MASO Yvan Christian
BARRY Ibrahima
HAIDARA Abdoul Wahab
IERMOLENKO Mariia
SOW Thierno Ousmane
SY Ndeye Oumoul Khairy

6th, December 2014

M1 MIAGE
ISTIC
University of Rennes 1

Product Comparison Matrices

Technical Document

PLAN

Illustrations Table

I. Preface

1. Purpose of this document :

This document is for use by PCM (Product Comparison Matrices) project developers. It provides guidance through the installation process and details about the project (its functionalities and structure) in order to help anyone involved in developing, monitoring, testing and deploying it to find a better documentation resource.

2. References:

The document's elaboration will mainly be based on our experience as involved developers in this project. Furthermore, some resources will be found on the project’s github webpage and in our teachers' lectures.

II. Introduction

1. Global function :

The product comparison matrix is designed to help determine which product is the right fit for people & their management needs. Product comparison matrices (PCMs) provide a convenient way to document the discriminant features of a family of related products.

2. Scope:

This project contains development artifacts used to perform research around product comparison matrices (PCM). PCM's repository on github is formed by nine (9) projects but the majors are:

• org.diverse.PCM.model: which contains the data (metamodel and model) and takes care of the code source generation and,

• org.diverse.PCM.io.Wikipedia: parses Wikipedia and creates PCMs.

This document will mainly study those two.

3. Definition, Anonyms & Abbreviation:

In the table below there are the main abbreviations used in the work.

PCM	*Product Comparison Matrices*
Class Diagram	*Describes the structure of a system*
Sequence Diagram	*Shows one or several sequences of messages sent among a set of objects*
Use-case Diagrams	*Illustrates the relationships between use cases*
Component Diagram	*A special case of a Class Diagram used to describe components within a software system*
System Block diagram	*A diagram showing the major components of the system with its interconnections and external interfaces*
KMF	*Kevoree Modeling Framework*

III. Installation

1. Pre-required

The minimum version of JDK - 1.7 should be installed, if it's missing download it here: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

2. Installation & Configuration

In order to get PCM's project work properly, we need to install its environment and make some configurations. Thus, we need to install development tools that are :

- **Git**

http://git-scm.com/downloads

Configuration task :

Unlike windows Systems, Linux and Mac OS need some additional configurations:

On Linux, follow these steps to get it work properly,

1) Unzip your file just downloaded

2) On a terminal type:

- git config --global color.diff auto

- git config --global color.status auto

git config --global color.branch auto

They will activate color on git to help read terminal messages more easily. Theses commands need to be type one time.

3) Configure our name and our email :

a) git config –-global user.name "a user name",

b) git config –-global user.emailmoi@email.com

4) Edit the configuation file: .gitconfig, here add an alias section at the end:

a) vim ~/.gitconfig

b) add this to the file:

[alias]

ci = commit

co = checkout

st = status

br = branch

The aim is to alias git commands, for instance instead of writing checkout one can write co.

- **Maven** – download the most recent bin.zip version here :

http://maven.apache.org/download.cgi

Configuration task :

- Unzip the zip file,
- Create an environment variable,
- Fix the value of the variable to the path to the maven bin directory ,
- Add the variable to the system PATH,

Test maven installation on a terminal: type : '**mvn –version**';

- **IntelliJ** - with the KMF plugin. Download it here:

https://www.jetbrains.com/idea/download/

Choose a community version according to the OS we have (Win, MAC or LINUX).

Download the project: Clone the project from its github repository:

1. For the project create a branch in our personal space on Github

2. Clone the project :

a) ***With GitBash :***

Type : git clone https://github.com/gbecan/PCM (destination file)

b) Using Graphical interface using ***INTELLIJ:***

VCS -> Checkout from version control -> Git -> copy

https://github.com/gbecan/PCM in Git Repository URL field.

In addition, we also need to install a framework PCM works with :

c) Kevoree Modeling Framework (***KMF***)

https://github.com/dukeboard/kevoree-modeling-framework

To get KMF:

1) clone it with:

GitBash type : git clone http://github.com/dukeboard/kevoree-modeling-framework (KEVOREE directory and PCM should be place in the same directory)

using Graphical interface using INTELLIJ :

VCS -> Checkout from version control -> Git -> copy

http://github.com/dukeboard/kevoree-modeling-framework in Git Repository URL field.

1) Review the last commit, type:

- get into the KEVOREE directory and
- git checkout '**a758cf07ded04e3dfc44b26bca7fcff8426d5495**'.

3. Troubleshooting:

Once the environment is set and the project cloned, open it with INTELLIJ and generate the maven project:

File -> Open -> Select the POM.xml file of the root's project in the dialog box -> OK

We can notice problems in the Maven project as shown here:

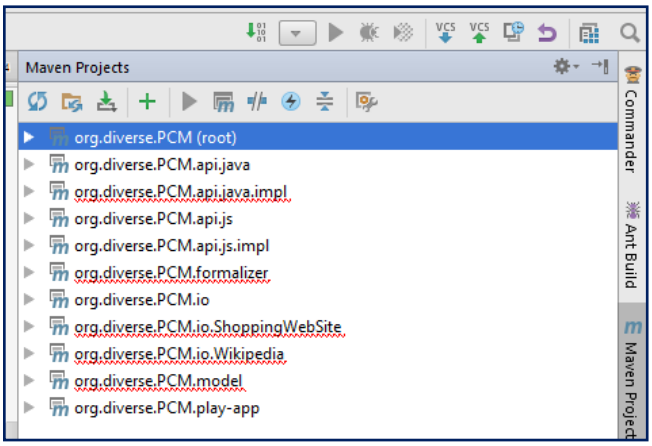

Image 1:Maven Project

Resolution Steps:

- ✓ For each underlined package, open its POM.XLM

It is recommended to start with the org.diverse.PCM.api.java.impl package.

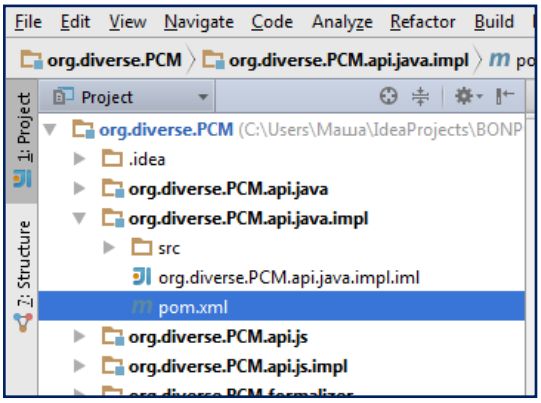

Image 2:Project Structure

- ✓ Look for <dependency> tags that are pointed out by the error messages when trying to clean install the Package (in the maven project).

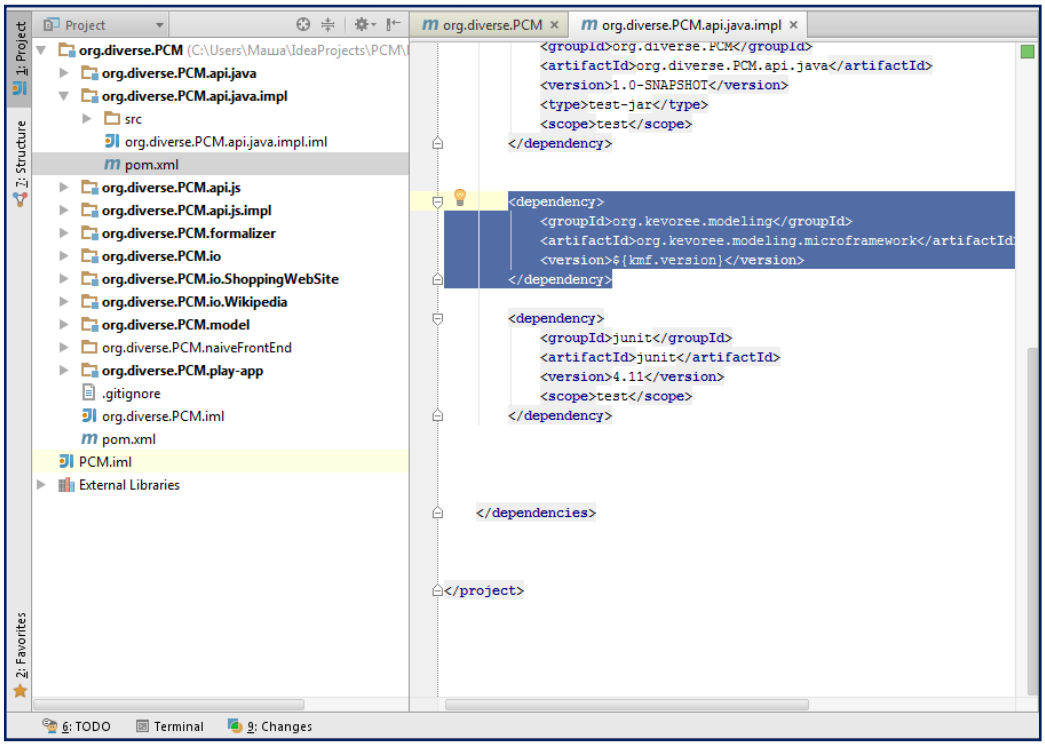

Image 3:Dependency

✓ Then, go to the Maven repository website (here : www.mvnrepository.com) and search for the artifact or the groupid of the dependency, choose the version we 're interested in.

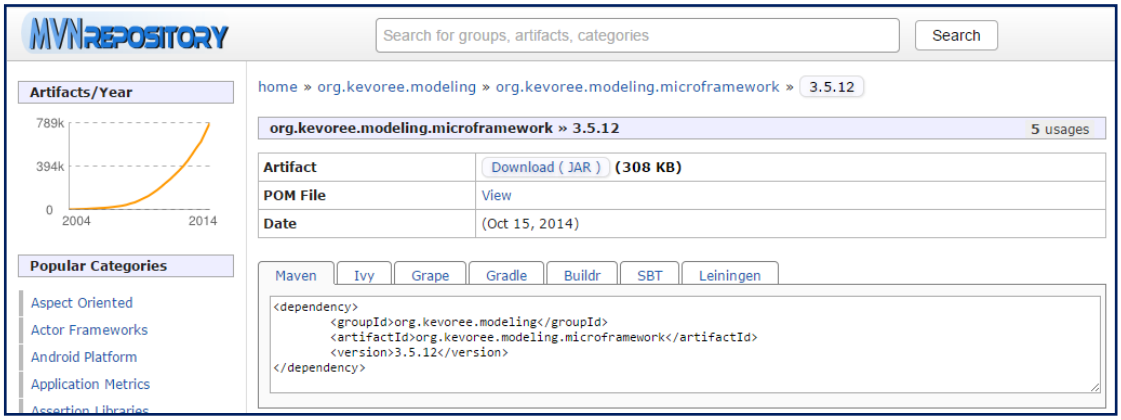

Image 4:Maven Repository

✓ Copy and paste its correct code, save the POM file, compile the package in the MAVEN project and import Maven Project's Changes.

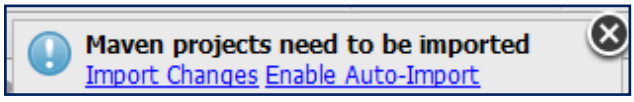

Image 5:: Imported Maven Project Message

And then the org.diverse.PCM.model. (In fact, other error packages depend on these ones, thus, their dependency problems will be fixed automatically).

- ✓ Now make a *clean install* on the project's root and we notice that the org.diverse.PCM.play-app has an activator bug. In fact, Play is not yet installed and the path to the activator is not precise either.

Play (to install once we get the project). To resolve it in the terminal (path - repository 'org.diverse.PCM.play-app') write '**activator.bat**'

IV. System Overview

1. System Architecture

The architecture defines structure of our PCM project in terms of organization of its functions. It describes how the different parts communicate with each other and its role in the project.

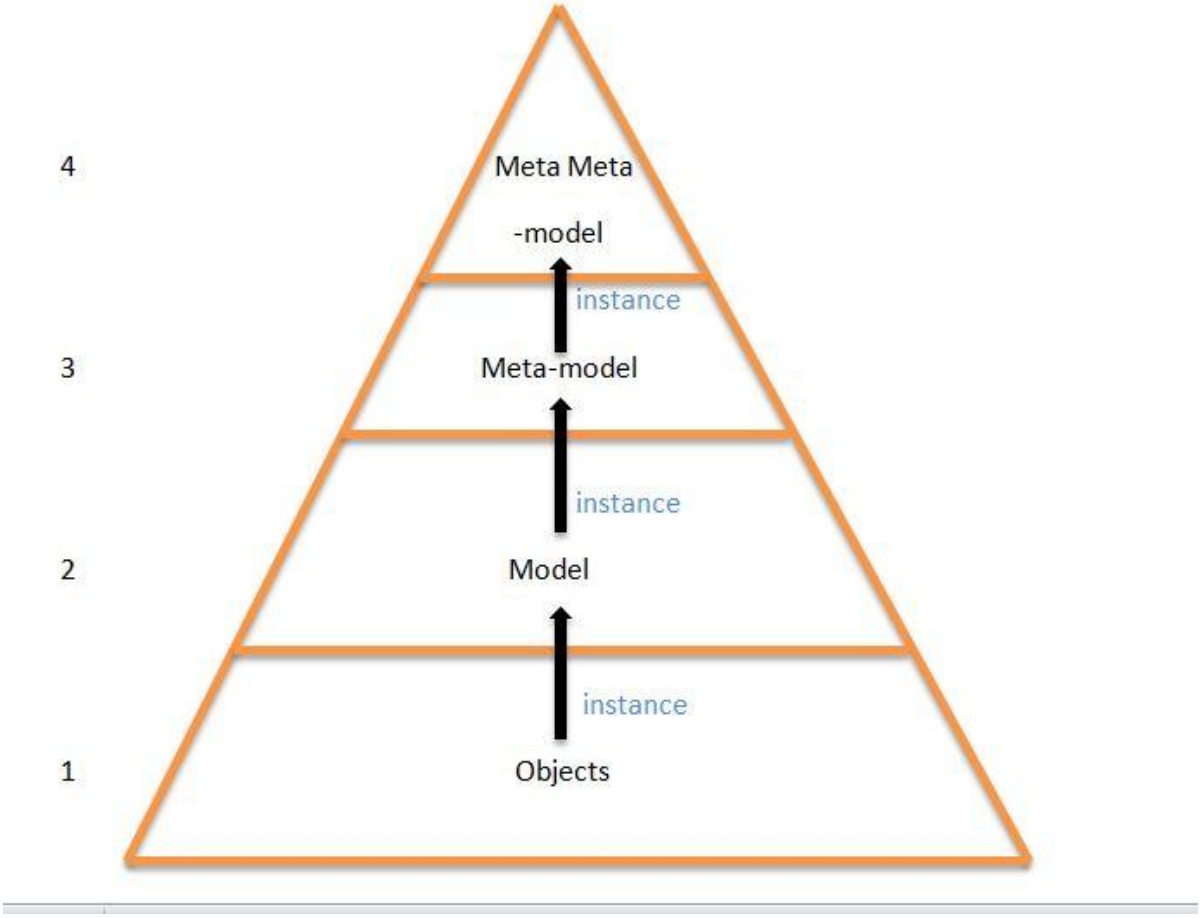

Diagram 1:MetaModel Pyramid

MDF (level 4): it is the most abstract level of the project. It is the core of ourproject. The MDF, pcm.ecore, shall in no case be changed because it is the Meta Meta-model of the PCM project.

Meta-model: it is the basis of our project. And all new features must obey these principles. Meta-model is an instance of MDF (namely Meta Meta-model).

The following the class diagram associated to Meta-model project:

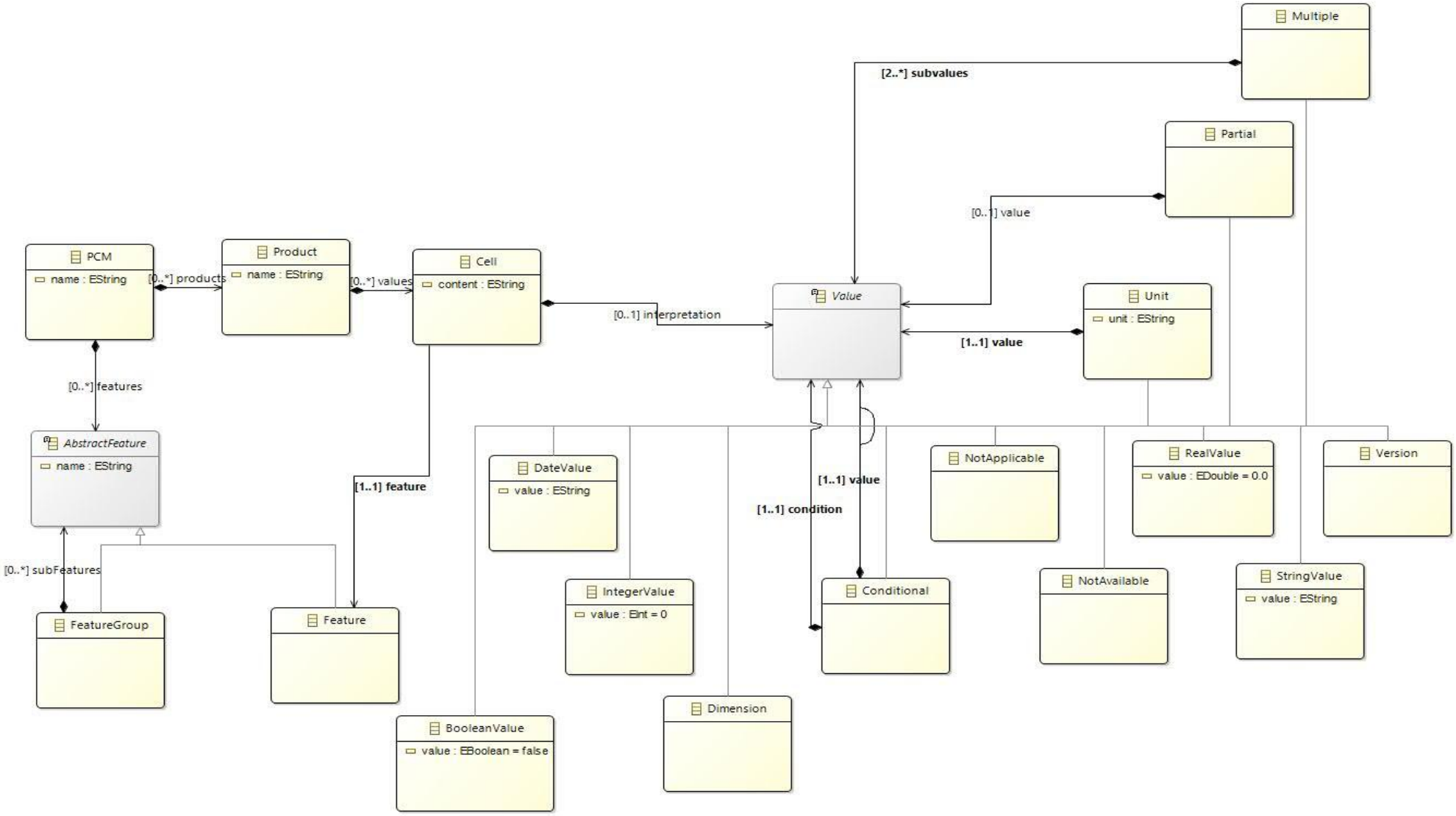

Diagram 2:MetaModel Diagram Class

Model: it is an instance of Meta-model. In our project it represents the different packages for example org.diverse.PCM.io.Wikipedia.

Project:

The project contains different packages:

- **org.diverse.PCM.api.java:** This package contains the different interfaces of our PCM project. It is basing on the Meta-model. It contains all the necessary interfaces to the different treatments on the matrices of comparison. In these different interfaces are operations that will allow to recover comparison matrices from PCM files.

The diagram represents an extract of this package is below.

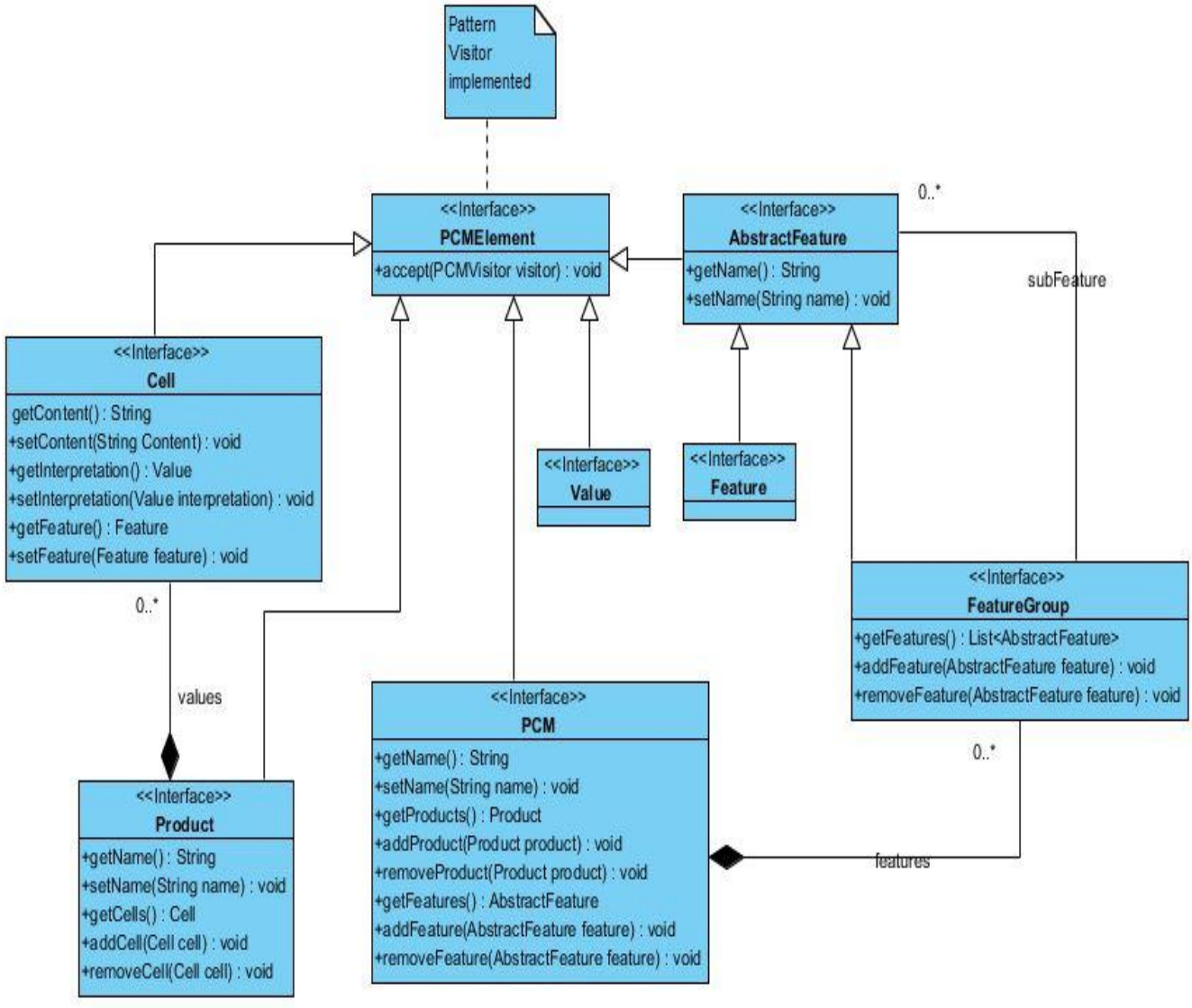

Diagram 3:API.PCM Class Diagram

- **org.diverse.PCM.api.js :** This package contains the JS interfaces.
- **org.diverse.PCM.api.java.impl:** It contains all the classes implementations of package org.diverse.PCM.api.java. We have concrete objects in this package. We base ourselves on this package for doing tests on org.diverse.PCM.io.Wikipedia in the concrete class WikipediaMinerTest.
- **org.diverse.PCM.api.js.impl**: Jsimplementation
- **org.diverse.PCM.model:** It contains the meta-model, all the project follows this model.
- **org.diverse.PCM.naiveFrontEnd**: a naive website to show how to manipulate model in a browser.

The aim of Front-End is to make web pages more ergonomic (visual aesthetics) and make more accessible the functional part (navigation on web pages).

He also takes care of portability, quality and referencing website by taking into account the different platforms and browsers used and applied with respect integration (for better visibility) web pages with new techniques.

- **org.diverse.PCM.io.ShoppingWebSite:** It parses shopping.com and creates PCMs
- **org.diverse.PCM.io.Wikipedia**: parses wikipedia and creates PCMs

We work at this level of the project. This package allows us to recover matrices of comparison from the web site of Wikipedia. The parser is in charge of parsing different web pages wikipedia for the final one recover only the matrices and put it into PCM file in script form containing only an array. This table (script PCM) contains different data matrix that we recovered on the web page.

- **org.diverse.PCM.play-app**: will contain the web editor

This is used to prepare or edit this page. The parsing is performed at the web editor. There are more details on the play in the part that talks about the role of play in the project

2. Infrastructure Service

PCM's project uses Java and Scala as programming language on INTELIJJ, Git for the version management and Maven for building. Futhermore, two additional developments tools (Play and Kevoree framework) are used.

INTELIJJ IDEA

Intellij is an Integrated Development Environment(IDE) which we use in its last Community released version(Version 14). This version supports :

- standard language programming (Java, Scala, Groovy, Clojure, JavaScript, CoffeeScript, HTML/XHTML/CSS, XML/XSL, ActionScript/MXML, Python, Ruby/JRuby, Structured Query Language, PHP),
- the use of version tools (Git, Svn, Mercurial et Gradle),
- use of build tools (Maven, Ant, Gradle et Gant).
- and Frameworks (Spring, JEE, Play, etc)

http://kendos-kenlen.olympe.in/intellij-idea-lun-meilleurs-ide-du-marche/ (For more details).

GIT

Git is a fast, recent and powerful development tool for version management. Projects are represented as branches thus developers can work in parallel. In addition, all previous versions are saved. However, Git may be a little complex, it may take time to understand and manipulate it.

Git has collaborative websites like GitHub (http://github.com/, the one used in this project) and Gitorious (http://gitorious.org/) where one can follow projects' evolution and participate.

MAVEN

Maven is an open source built tool for Java projects. It manages transitive dependencies, life cycle and personalized builds.

- The heart of a Maven project is the project Object model(POM), which is a XML file placed in the root directory of the project. It contains detailed description of the project with information about the versions, the configuration management, the dependencies, the application resources and the tests.
- The main goals of Maven life cycle are either:
  - to compile(compile the source code of the project),
  - to test (run unitary testing with mainly JUNIT in the src/test directory)
  - to pack (shape the compiled code in its format of diffusion(JAR, WAR, etc)

- to install (to install products in the local repository in order to be used as dependency by other local projects)
- to deploy (we be performed in a productive or integrated environment, it consists in copying the final product in a distant repository so that it can be shared).

➢ Dependencies Management:

In Maven, dependencies may have 4 types of visibility:

- compile (a dependency with this type of visibility is available in all phases. It's the default value),
- provided(in this case, it will be used to compiled the application but won't be deployed,
- use this when we want the JDK or the application server to make available the JAR)
- runtime (these types of dependencies are not required for the compilation but are for the execution just like JDBC drivers)
- test (these dependencies are necessary only to run test).

We can add functionalities to Maven using plugins available on Maven website or developing them. In either case, declare them in the POM file so that we will be able to use them.

PLAY

Play in a web open source framework used to write web application in Java or Scala quickly. Play is not based on Java motor 's Servlet. It's then easy to have a project's skeleton and then run the play server to get a home page.

Play gives all the necessary links to the website (database conations, mapping),manages errors(by giving the page causing it), allows quick re-compilation in case of saved modification on a file, has extensible modules for security(secure) and automatic generation of administrative data pages (crud).

Our play project's main directory are:

- app/ (contains the heart of the application with the MVC design pattern. The Java classes will be stored in here).
- conf/ (stores the configuration files, routes definition and files for internationalization)
- lib/ (contains optional Java libraries)
- public/ (stores public resources like images, Javascript or CSS files)
- test/ (contains test files. Theses latest may be Java Test with JUNIT or SELENIUM)

KEVOREE

KMF supplies runtime-oriented features such as: Memory optimized object oriented modeling API JS (Browser, NodeJS) and JVM cross-compiled models Efficient visitors for models traversal Unique path for each model element Optimized query language to lookup model elements

Trace operators to atomize model operations into low-level primitive sequences :

- Built-in load/save operation in JSON/XMI format
- Built-in clone strategies (mutable elements only, copy on write)
- Built-in merge and diff operators
- Persistence layer for BigModels with lazy load policy
- Distributed data store for BigData models

V. System Functionalities

1. Design method

PCM's project contains several projects, but the main one is the Wikipedia projet.The Visitor design pattern is used in that project. It can be found in the ***org.diverse.PCM\org.diverse.PCM.io.Wikipedia\src\main\scala\org.diverse.PCM.io.Wikipedia\pcm\parser*** directory.

Why is this pattern used in that project?

Classes in Wikipedia follow the closed principle in Object-Oriented programming. In addition, they are stored in a format which is difficult to understand. Methods in the PCM project need to work with those classes, thus, the Visitor design pattern must be used to gather information about the exact type of their instances and find out the treatment that best fits their public details.

In practice, Visitor Design Pattern works as follows: each "visited" class should provide a public method "accept" that can take the "visitor" as an argument.

In this case, the visitor classes are PageVisitor, TableVisitor, PreprocessVisitor, NodeToTextVisitor.

The sequence is as follows:

- PageVisitor visits a page then,
- TableVisitor visits the tables in that page then,
- calls the NodeToTextVisitor to visit the cells on the tables then,
- PreprocessVisitor makes the preprocessed code of the page and the client is the WikipediaPCMParser class.

The "accept" method calls the "visit" method of the visitor that takes the visited object as an argument. Thus, the visitor knows the object's reference and is able to get data it needs to proceed.

In our specific case, Visitor will add new virtual methods to Wikipedia's classes so that they can be manipulated.

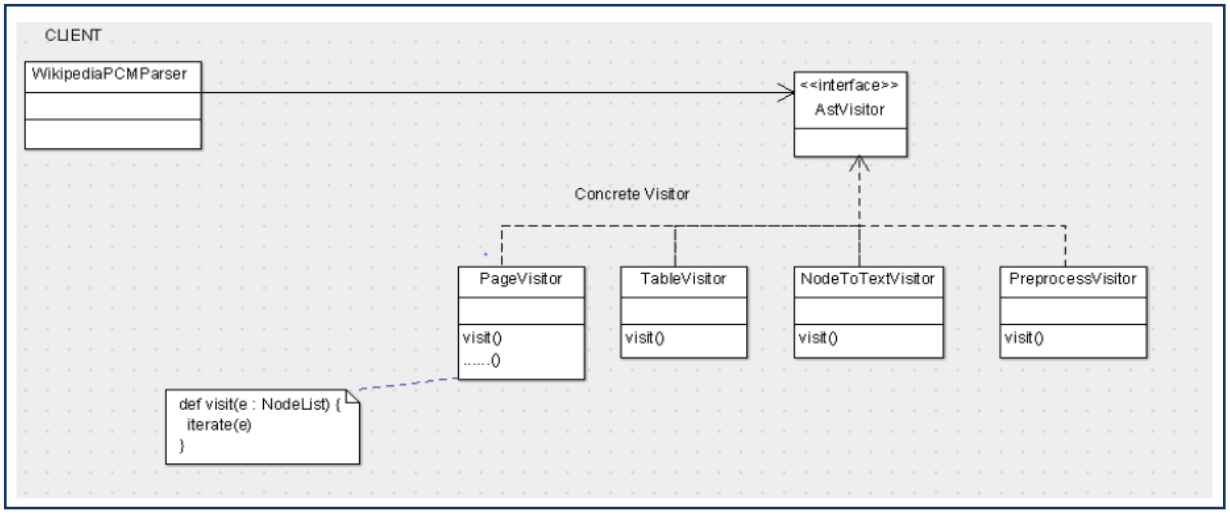

Diagram 4:Visitor Diagram Class

2. Naming Convention

The principle of naming is based on Java.

- ***Packages***

The package name must satisfy the following requirements:

a) All packages must begin by 'org.diverse.PCM' for the project PCM.

b) Use only [a-z], [0-9] and a point '**.**'. Don't use dashes '-', underscores '_', spaces ' ', and other characters ($, *, ...).

- ***Classes***

The names of the classes must meet the following criteria:

a) The first letter is a capital letter.

b) LowerCase letters mix with uppercase first letter of each word capitalized

c) Give simple and described names.

d) Avoid abbreviations apart from such as : XML, URL, HTML, ...

e) Use only [a-z], [[A-Z] and [0-9]. Don't use dashes '-', underscores '_', spaces ' ', and other characters ($, *, ...).

- ***Interface***

Interfaces follow the same rules as names of the classes. Correspond with the existing principles : don't use a letter 'I' and other characters for determining the interface.

- ***Methods***

The methods are called in accordance with a method camelCase and usually they include an action verb. For example : open(), insert text(), cut() etc.

- ***Variables***

The names of the variables must satisfy the following requirements :

a) The first letter is a small letter.

b) LowerCase letters mix with uppercase first letter of each word capitalized

c) Give simple and described names.

d) Use only [a-z], [A-Z] and [0-9]. Don't use dashes '-', spaces ' ', and other characters ($, *, ...).

e) Variables are denoted by a single letter for local application.

   - **int** : i, j, k, m, and n
   - **char** : c, d, and e
   - **boolean** : b

- ***Constants***

The names of the variables must satisfy the following requirements :

a) All the letters is the capital letters.
b) Separate words by an underscore '_'.
c) Give simple and described names.
d) Don't use letters [A-Z], [0-9] and an underscore '_'. Also don't use dashes '-', spaces ' ', and other characters ($, *, ...).

- ***Comments***

Comments are frequently used by programmers who rework the code or provide the documentation for the program. Suggested to use Javadoc which is a system with tags indicating the things such as: date, author, version, arguments and others.

3. Wikipedia Functionnality

Wikipedia's functionality can be describe as follows:

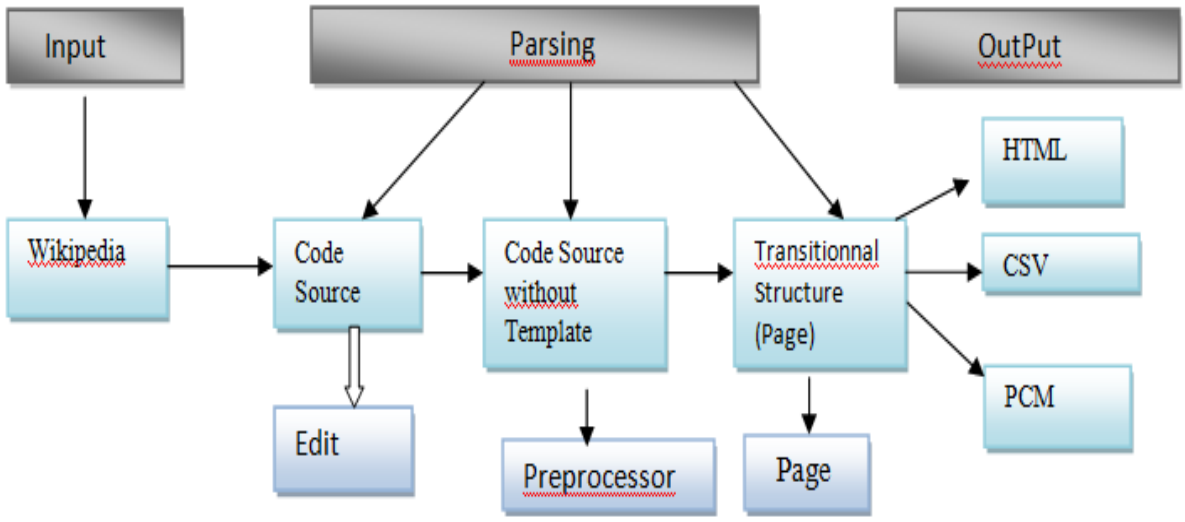

Diagram 5:Wikipedia Diagram

We can divide this process in 2 stages:

1)*The recovery stage*:

The WikipediaPager class is used during this stage.The input parameter is the Wikipedia page title and the output is the source code of the page that one can obtain using the Edit link in The Wikipedia web page.Then, the "preprocessed" method uses this output as an input to take off the templates from the source code.

2) *The parsing stage*:

The Pager class is used here. During this stage, the Pager Class uses the source code generated by the preprocessed method to create an intermediate structure "Page".Then, Page is used to produce PCM, CSV or HTML representation as output. The PCM representation which shows products and their features is the one we are interested in.

Wikipedia Sequence Diagram

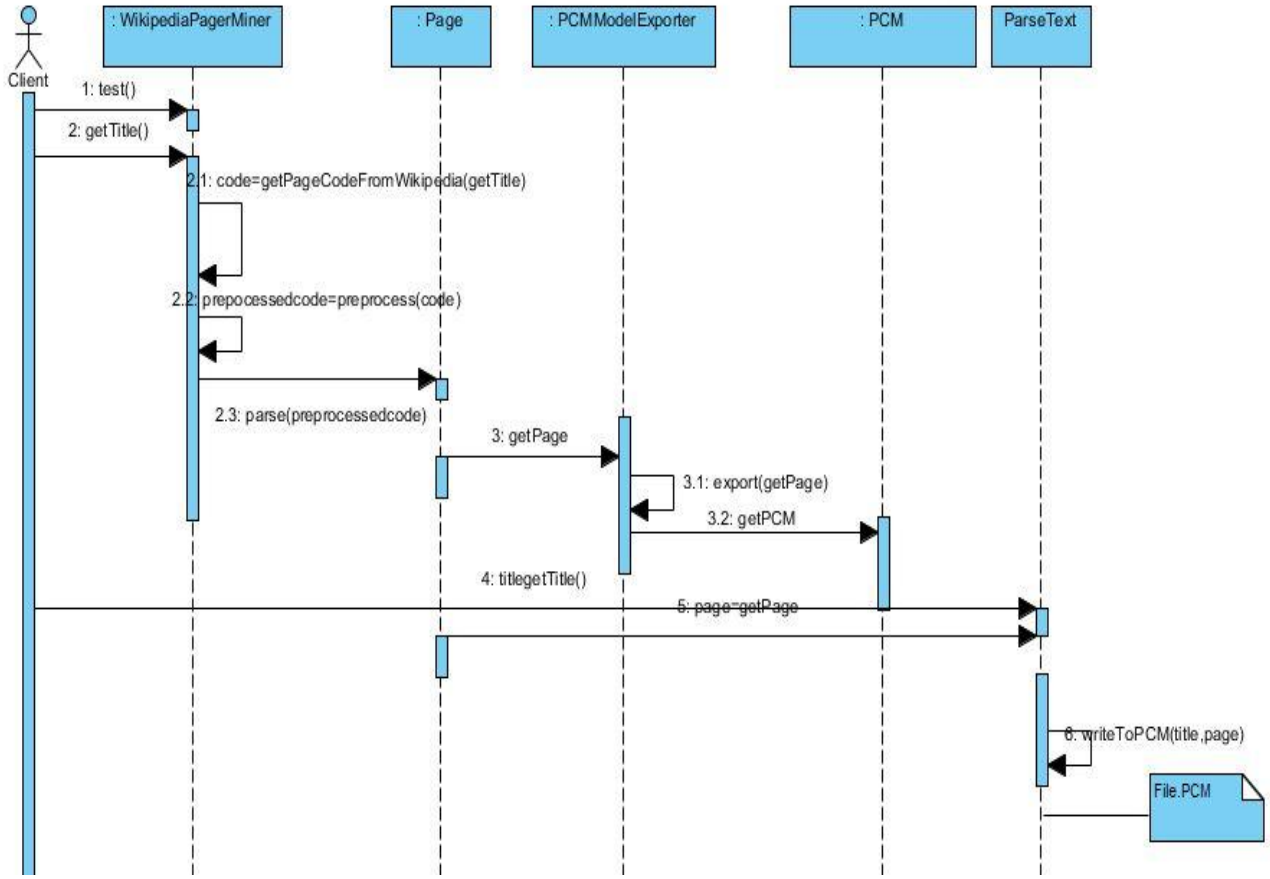

Diagram 6: Wikipedia Sequence Diagram

VI. Test

Testing of our PCM project was carried out at org.diverse.io.PCM.Wikipedia, which is a main part of our application. The tests belong to 2 classes such as java and scala types.

The 2 classes are:

- WikipediaMinerTest.java
- ParseTest.scala

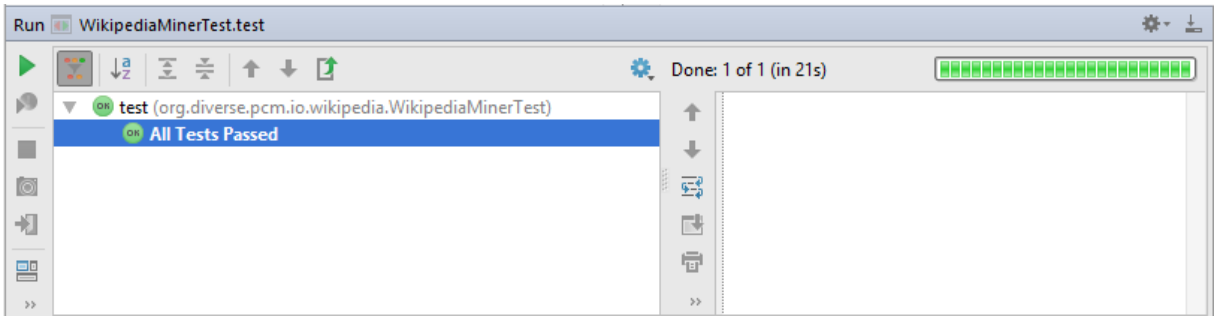

Image 6 : Test Result

The tests were carried out successfully. It is necessary to focus on file "WikipediaMinerTest.java". A following diagram shown below presents this.

The test monitors this file in the following way:

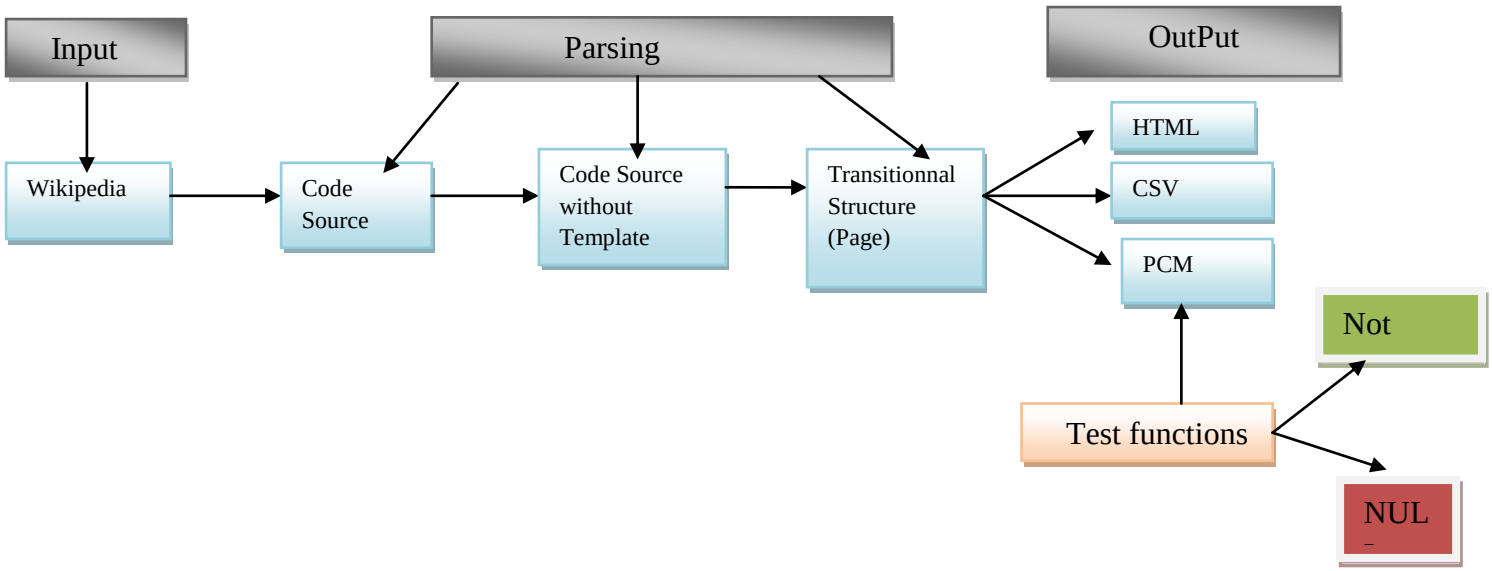

Diagram 7: Test Diagram

The tests show that the output files obtained after the parsing phase in 3 formats (CSV, HTML, PCM) are not nulls.

Control of no-empty PCM

In order to the test that the PCM returns will be ‘not null’, it is necessary to browse beginning with the products and the features, next check if they really exist the corresponding cells to the products and features.

Below it’s the java code for this control:

```
for(Product p:pcm.getProducts())
    {

        for(AbstractFeature f :pcm.getFeatures())
        {
             for (Cell c:p.getCells())
             {
                 System.out.println(c.getContent());
             }

        }

    }
```

Image 7:PCM process

The result for the "Comparison_of_assemblers" of the PCM is:

```
1 of 1 (in 34s)
"C:\Program ...
0
Proprietary
1979-1983
No
No
86-DOS, DOS DEBUG
Proprietary
1979-1983
No
No
86-DOS, DOS DEBUG
Proprietary
1979-1983
No
No
86-DOS, DOS DEBUG
Proprietary
1979-1983
No
No
86-DOS, DOS DEBUG
```

Image 8: PCM process Results

VII. ANNEXES

org.diverse.pcm.api.java :PCMTEST CLASS

```
package org.diverse.pcm.api.java;

import static org.junit.Assert.*;
import static org.junit.Assert.assertEquals;

import org.diverse.pcm.api.java.value.BooleanValue;
import org.hamcrest.CoreMatchers;
import org.junit.Before;
import org.junit.Test;

import java.util.List;

/**
 * Created by gbecan on 09/10/14.
 */
public abstract class PCMTest {

    protected PCMFactory factory;

    @Before
    public abstract void setUp();

   //method that tests adding ,the deletion of a   feature , a
feature of goupe in pcm object,
    //adding ,the deletion of a product in a pcm object,
    //the name of the pcm

    @Test
    public void testPCM() {
        //it creates an object of type pcm
        PCM pcm = factory.createPCM();


        //tested, if you look there's two type of feature in
Elements' pcm object
        assertEquals(pcm.getFeatures().size(), 0);


        //we test if there is any type of feature element in the
object pcm
        assertEquals(pcm.getProducts().size(), 0);

        //we give it the name "pcm name"
        pcm.setName("pcm name");
        // we test, we look back if the name of the products is
the same as the name to "pcm name"
        assertEquals(pcm.getName(), "pcm name");
```

```
        //adding a feature in the pcm object
        pcm.addFeature(factory.createFeature());
        //adds featuregroup in pcm object
        pcm.addFeature(factory.createFeatureGroup());
        //tested, if you look there's two type of feature in
Elements' pcm object
        assertEquals(pcm.getFeatures().size(), 2);

        //it removes the first feature type element in the object
pcm
        pcm.removeFeature(pcm.getFeatures().get(0));
        //it is tested whether the deletion was effected, if there
is only one element of such feature in pcm object
        assertEquals(pcm.getFeatures().size(), 1);

        //adding a product to the pcm object
        pcm.addProduct(factory.createProduct());
        //we test the poduct is added, it has an element (1) type
of product in the pcm object
        assertEquals(pcm.getProducts().size(), 1);

        //we remove the first product type element in the object
pcm
        pcm.removeProduct(pcm.getProducts().get(0));
        //we test the poduct is delete, ny no (0) product type
element in the object pcm
        assertEquals(pcm.getProducts().size(), 0);

    }
 //method that tests whether the name given to the feature is the
same return

    @Test
    public void testFeature() throws Exception {
        //I create a feature
        Feature feature = factory.createFeature();
        //we give it the name "feature name"
        feature.setName("feature name");
        // we test, we look back if the name of the feature is the
same as the name to "feature name"
        assertEquals(feature.getName(), "feature name");
    }


    //method that tests the feature of group if it returns the
name of the group of feature,
    // testing the addition and deletion if well done

    @Test
    public void testFeatureGroup() throws Exception {
        // I created my feature and I verified that there is no
element (0) in the featuregroup.
        FeatureGroup featureGroup = factory.createFeatureGroup();
```

```
        assertEquals(featureGroup.getFeatures().size(), 0);

        //I give the name "feature name" of the feature group
        featureGroup.setName("feature group name");
        //I test if the name is the same for return feature group
        assertEquals(featureGroup.getName(), "feature group
name");

        //I created a feature
        Feature feature = factory.createFeature();
        //I add the feature to create the feature group
        featureGroup.addFeature(feature);
        featureGroup.addFeature(factory.createFeatureGroup());
        //I test if I have actually two Elements in featuregroup
        assertEquals(featureGroup.getFeatures().size(), 2);

        // I delete a feature on featuregroup to watch if the
removal is going well
        featureGroup.removeFeature(feature);
        // I look if the removal is done well: there is only one
element in the group
        assertEquals(featureGroup.getFeatures().size(), 1);
        assertThat(featureGroup.getFeatures().get(0),
CoreMatchers.instanceOf(FeatureGroup.class));
    }



    //test method on the addition, deletion of a Product in a
cell,
    //testing name return

    @Test
    public void testProduct() throws Exception {
        //I create a Product
        Product product = factory.createProduct();
        //I tested that it  hasn't no element in the cell of
product
        assertEquals(product.getCells().size(), 0);

        //I give the name "product name" in the product
        product.setName("product name");

        // I test if the name returned is the same as the name
given
        assertEquals(product.getName(), "product name");

       // I created my cell
        Cell cell = factory.createCell();
        // I add the cell to the Product
        product.addCell(cell);
// I test if the product was add, if there is one (1) element in
```

```
the cell
        assertEquals(product.getCells().size(), 1);


        // I delete the cell
        product.removeCell(cell);
        // I test if the product has been deleted, if there is no
(0) element in the celulle
        assertEquals(product.getCells().size(), 0);
    }

    // Method that tests the cell contents and data
    @Test
    public void testCell() throws Exception {
        // I created my cell
        Cell cell = factory.createCell();

        // I put a content in the cell
        cell.setContent("content");
       // I test, I verified if it is the same content that is
returned
        assertEquals(cell.getContent(), "content");

        //I created a variable of type boolean
        BooleanValue value = factory.createBooleanValue();
        //I give the true value
        value.setValue(true);

        //I give a true interpretation of the cell
        cell.setInterpretation(value);

        // I test, I verified if it is true that is returned in
the interpretation of the cell
        assertThat(cell.getInterpretation(),
CoreMatchers.instanceOf(BooleanValue.class));
        assertEquals(((BooleanValue)
cell.getInterpretation()).getValue(), true);


        // I created a Feature
        Feature feature = factory.createFeature();
        //we give it the name "feature name"
        feature.setName("feature name");
        //I change the name of the cell with the feature create
"feature name"
        cell.setFeature(feature);
        // I test, I verified if it is the same name that is
returned
        assertEquals(cell.getFeature().getName(), "feature name");

}
}
```