FP7-ICT-2011-8

MARKOS

The MARKet for Open Source

An Intelligent Virtual Open Source Marketplace

WP4 – Code Analyser

D4.1.1a– Requirements for information extraction from source code v1

Due date: 31.3.2013 **Delivery Date**: 16.4.2013

Author: Gabriele Bavota, Massimiliano Di Penta, Unisannio

Partners contributed:

Dissemination level: CO* **Nature of the Deliverable:** Report

Internal Reviewers: Pawel Kedziora, PSNC (27.03.2013), Davide Galletti, GEEKNET (23.03.2013), Maria Saguino, ATOS (26.03.2013)

Davide Galletti, GEEKNET (29.03.2013) – Second Review

* **CO (**Confidential, only for members of the Consortium, including the Commission Services)

VERSIONING		
VERSION	DATE	NAME, ORGANIZATION
0.1	**20/02/2013**	**GABRIELE BAVOTA AND MASSIMILIANO DI PENTA,UNISANNIO**
0.2	**06/04/2013**	**GABRIELE BAVOTA AND MASSIMILIANO DI PENTA,UNISANNIO**

Executive Summary

The general goal for the Code Analyser is to extract from software repositories information about software artefacts to be used for supporting their retrieval in the context of a code search activity. This document describes the requirements for the source code analysis performed by the Code Analyser.

D4.1.1a– Requirements for information extraction from source code v1

Table of Contents

1. INTRODUCTION

This document describes the requirements for the source code analysis performed by the Code Analyser. The general goal for the Code Analyser is to extract from software repositories information about software artefacts to be used for supporting their retrieval in the context of a code search activity. The code search can be performed at different levels of granularity, e.g., entire packages/libraries able to fulfil a given piece of functionality, a source code file/class, but also a small source code snippet/fragment that can be relevant for a specific task, e.g. a snippet useful to implement a particular sorting algorithm. Moreover, the Code Analyser must extract and process the source code licensing information that will be used in WP3 to determine whether the discovered artefact is compliant from a legal point of view. In addition, it is necessary to extract dependencies, to identify requirements necessary to be fulfilled when one wants to use a discovered code artefact, and to determine whether the dependencies would create legal issues, e.g., because of interconnection with artefacts that are not compatible-from a licensing point of view-with the system.

This first version of the Code Analyser's requirements starts to identify the facts to extract from source code in order to support the aforementioned functionalities.

1.1. Document organization

The document is organized in five sections including this introduction. Section 2 summarizes the main features that must be provided by the Code Analyser together with the functional requirements (in form of user stories) related to them. Section 3 presents the data dictionary showing the information that the Code Analyser will extract from the analysed project releases while Section 4 describe the non-functional requirements of the Code Analyser. Section 5 briefly describes the control flow of the Code Analyser when performing code analysis, while Section 6 concludes the document.

2. FEATURES OF THE CODE ANALYSER AND FUNCTIONAL REQUIREMENTS

In this section we describe the main features provided by the Code Analyser. Each feature is overviewed in a dedicated subsection together with the requirements derived by it.

2.1. Download data related to projects identified by the Crawler

The Code Analyser exposes a web service to allow the Crawler to notify it when new projects or new project releases are available. For each new release available, the Code Analyser expects from the Crawler the following info:

1. The name of the project (e.g., Apache Tomcat)
2. The name of the release (e.g., 6.0)
3. The release's date (e.g., 25/10/2011) [if available]
4. A list of possible repositories available to download the source code of the new release to analyse.

This info will be passed from the Crawler to the Code Analyser through an HTTP request.Then, the Code Analyser downloads the source code of the new project releases and starts the source code analysis.

2.1.1. Requirements and user stories derived from this feature

a) The **Code Analyser** must be able to download source code from a given software repository, so that it can start the code analysis.

b) the**Code Analyser** must be able to decompress an archive containing source code, so that it can start the code analysis. The supported formats must be at least:

   a. .tar
   b. .gz
   c. .tar.gz
   d. .zip
   e. .jar.

   Other archive formats will be added in the future (e.g., .rar, .war).

2.2. Code analysis

The Code Analyser analyses the downloaded source code through the following steps:

1. Firstly, it converts the source code of each new release in srcML[1] format, an XML-like representation of source code.

2. Then, a parser based on Java DOM[2] is used to extract structural information about the source code. During this stage all the info reported in Section 3 of this document are extracted and stored in an internal database (described in Section 3), except: info related to the licenses, and dependencies between code components.

3. Dependencies between code components are extracted in the third step of code analysis. There are two possible kinds of dependencies existing between code entities:

    - Dependencies between code entities belonging to the same project (e.g., a method m1 implemented in class C1 of Apache Tomcat calls a method m2 implemented in class C2 of Apache Tomcat). From now on we refer to these dependencies as *internal dependencies*.

    - Dependencies between code entities belonging to different projects (a method m1 implemented in class C1 of Apache Tomcat calls a method m3 implemented in class C3 of an imported library). From now on we refer to these dependencies as *external dependencies*.

    The analysis of the internal dependencies is performed just for the new (just analysed) releases. Examples of dependencies extracted include calls between methods, inheritance and extend relationships, import/include relationships.

    Concerning the external dependencies, the Code Analyser will periodically look for them in the entire database, trying to improve the stored info. For instance, there could be dependencies stored in the database for which we know the *source* entity without having identified the *target* entity. As example, it is possible that during the analysis of a project a call from a method m1 to a method m2 imported from an external library is identified. However, if the source code of the external library is not available, it will not be possible to identify the project to which m2 belongs. To solve this problem the Browsing&Querying component will expose an interface allowing the Code Analyser to require the provenience of a code element (at least at source file level). If also the Browsing&Querying component is not able to retrieve this information, the dependency will be marked by the Code Analyser as "not solved" and will be analysed in a future stage when more projects are available in the semantic store.

Just after the analysis of the internal dependencies, the Code Analyser starts performing the licensing statement analysis and classification (see Section 3.3).

[1] http://www.sdml.info/projects/srcml/

[2] http://docs.oracle.com/javase/1.4.2/docs/api/org/w3c/dom/package-summary.html

2.2.1. Requirements and user stories derived from this feature

c) The **Code Analyser** must be able to automatically convert the source code in srcML format, so that fact extraction from source code is possible.

d) The **Code Analyser** must support as programming languages at least:

   a. **Java**

   In future other programming languages (starting from C and C++) will be supported.

e) The **Code Analyser** must be able to extract information about source code reported in the data dictionary depicted in Section 3.

2.3. Licensing statements analysis and classification

The license analysis is performed on each new release processed by the Code Analyser. In particular, each source code file is parsed by Ninka[3], a lightweight license identification tool. The license's info extracted from each source code file include: the licensing statement reported in the file, its contributors, the licenses specified in it, and the covered copyright years.

2.3.1. Requirements in form of user stories derived from this feature

f) The **Code Analyser** must be able to extract from source code all the information related to licenses reported in the data model presented in Section 3 of this document.

2.4. Web-services offered by the Code Analyser

All the above described features will be possible thanks to the following RESTful web-services exposed by the Code Analyser:

1. analyseNewProjectReleases(List<Projects>infoAboutNewReleases) [exposed to the Crawler]. This service will allow the Crawler to notify the Code Analyser about new project releases to analyse.

2. analyseAprojectProvidedByUser(URL srcArchiveURL) [exposed to the Front-end]. A Markos' user can ask Markos to analyse a project not yet indexed by the system. In this case the front end can pass to the Code Analyser a URL pointing to an archive containing the source code to analyse.

[3]http://ninka.turingmachine.org/

3. getProjects(Date lastUpdate) [exposed to the Repository]. The semantic store can invoke this service time by time to receive the new available data from the Code Analyser.

4. notifyMedatataChange(List<Projects> projects) [exposed to the Crawler]. This service will allow the Crawler to notify the Code Analyser about changes in the metadata of already indexed projects.

5. notify(intbatchId) [exposed to the Crawler]. This service will allow the Crawler to notify the Code Analyser when new data about a specific project requested by the Code Analyser are ready.

3. DATA DICTIONARY OF THE CODE ANALYSER

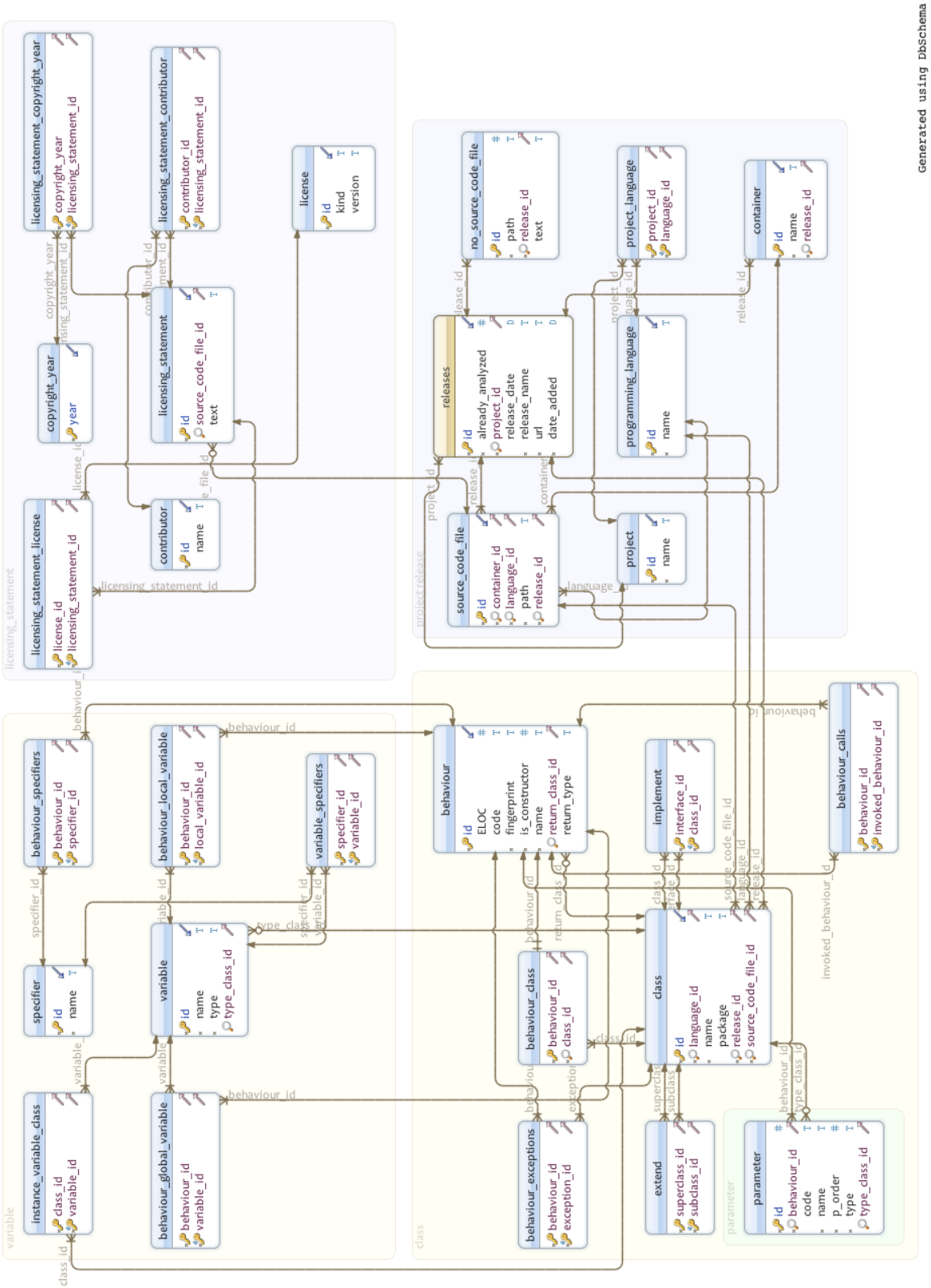
Generated using DbSchema
licensing_statement
licensing_statement_copyright_year
copyright_year
licensing_statement_id
licensing_statement_contributor
contributor_id
licensing_statement_id
license
id
kind
version
licensing_statement
id
source_code_file_id
text
contributor
id
name
licensing_statement_license
license_id
licensing_statement_id
project release
releases
id
already_analyzed
project_id
release_date
release_name
url
date_added
no_source_code_file
id
path
release_id
text
project_language
project_id
language_id
container
id
name
release_id
programming_language
id
name
source_code_file
id
container_id
language_id
path
release_id
project
id
name
variable
behaviour_specifiers
behaviour_id
specifier_id
behaviour_local_variable
behaviour_id
local_variable_id
variable_specifiers
specifier_id
variable_id
specifier
id
name
variable
id
name
type
type_class_id
instance_variable_class
class_id
variable_id
behaviour_global_variable
behaviour_id
variable_id
class
behaviour
id
ELOC
code
fingerprint
is_constructor
name
return_class_id
return_type
implement
interface_id
class_id
behaviour_calls
behaviour_id
invoked_behaviour_id
behaviour_class
behaviour_id
class_id
class
id
language_id
name
package
release_id
source_code_file_id
behaviour_exceptions
behaviour_id
exception_id
extend
superclass_id
subclass_id
parameter
parameter
id
behaviour_id
code
name
p_order
type
type_class_id

Figure 1: Code Analyser data dictionary

In this Section we describe the Code Analyser data dictionary reported in Figure 1. Note that the information stored in the Code Analyser database at this stage of the project (i.e., first sprint) is mainly designed for Object-Oriented programming languages. In the version v2 of this document the complete data dictionary will be presented.

project Stores generic information about a project			
Attribute	**Type**	**Description**	**Example**
name	varchar	The name of the project	Sweet Home 3D
id	int	Primary Key	1

programming_language Represents a programming language that could be used in different projects			
Attribute	**Type**	**Description**	**Example**
name	varchar	The name of the programming language	Java
Id	Int	Primary Key	1

project_language Stores the programming languages used in a project			
Attribute	**Type**	**Description**	**Example**
project_id	int	The id of the project	5
language_id	int	The id of the programming language	1

source_code_file Represents a file of a project release containing source code			
Attribute	**Type**	**Description**	**Example**
path	varchar	The path of the code element	/src/ManagerUser.java
language_id	int	The programming language used in the source code file	Java
release_id	int	The project release to which the source code file belong to	4
container_id	int	The container in which the source code file is placed	6
id	int	Primary Key	1

releases Stores the different releases analysed by Markos			
Attribute	**Type**	**Description**	**Example**
id	int	Primary Key	3
project_id	int	The project to which the release belongs	8
release_date	Date	The release date	10-10-2012
release_name	varchar	The name of the release	2.0.1
url	varchar	The url from where the source code for this release has been downloaded	http://svn.argouml.it/ Argo_2.0.0.tar.gz
already_analyzed	int	1 if source code analysis has been performed for this release, 0 otherwise	0
date_added	Date	The date in which the Code Analyser has added this project in the	21-10-2012

		database.	

container A container (e.g., a Package) contained in a project release			
Attribute	**Type**	**Description**	**Example**
id	int	Primary Key	3
name	varchar	The container's name	org.argouml
release_id	int	The id of the release to which the container belongs	7

no_source_code_file Represents a non-source code file contained in a project release			
Attribute	**Type**	**Description**	**Example**
id	int	Primary Key	3
path	varchar	The path of the file	/src/config/conf.txt
release_id	int	The id of the release to which the file belong	6
text	varchar	The textual content of the file	"This file contains the configuration parameters of the system.."

licensing_statement Represents a licensing statement contained in a file			
Attribute	**Type**	**Description**	**Example**
id	int	Primary Key	3
text	varchar	The licensing statement	This class is released under the license…
source_code_file_id	int	The id of the file in which the licensing statement is contained	7

licensing_statement_license Stores the licenses contained in a licensing statement			
Attribute	**Type**	**Description**	**Example**
license_id	int	The id of the license	3
licensing_statement_id	int	The id of the licensing statement	5

licensing_statement_copyright_year Stores the copyright years contained in a licensing statement			
Attribute	**Type**	**Description**	**Example**
copyright_year	int	The copyright year	2001
licensing_statement_id	int	The id of the license	3

licensing_statement_contributor Stores the contributors specified in a licensing statement			
Attribute	**Type**	**Description**	**Example**
contributor_id	int	The id of the contributor	3
licensing_statement_id	int	The id of the license	8

contributor Represents a contributor of one or more licensing statements			
Attribute	**Type**	**Description**	**Example**
id	int	Primary Key	3
name	varchar	The contributor name	Eclipse Foundation

copyright_year Represents a copyright year			
Attribute	**Type**	**Description**	**Example**
year	int	Primary Key	1998

class Represents a class contained in a source code file			
Attribute	**Type**	**Description**	**Example**
id	int	Primary Key	3
language_id	int	The id of the programming language used	Java
name	varchar	The name of the class	ManagerUser
release_id	int	The id of the release to which the class belong	6
source_code_file_id	int	The id of the source code file in which the class is defined	8
package	varchar	The name of the package containing the class	userManagement.db

behavior Can represent a method or a function			
Attribute	**Type**	**Description**	**Example**
id	int	The primary key	5
ELOC	int	The lines of code of the method, excluding comments and blank lines	78
code	varchar	The code of the method. Will not be stored in the final version of Markos, is just used to test the Analyser	public static…
fingerprint	varchar	Is composed by a set of metrics measured for the method	4170942183290321
is_constructor	Boolean	true if the method is a constructor, false otherwise	false
name	varchar	The name of the method	getUser
return_class_id	int	The id of the returned class (if the returned class is a class of the system, NULL otherwise)	5
return_type	varchar	A string reporting the return type of the method. It is useful if return_class_id is NULL	String

behavior_class Stores the methods contained in a class			
Attribute	**Type**	**Description**	**Example**
behaviour_id	int	The id of the method	3
class_id	int	The id of the class	8

behavior_exceptions Stores the exceptions thrown by a method			
Attribute	**Type**	**Description**	**Example**
behaviour_id	int	The id of the method	3
exception_id	int	The id of the class representing the exception	14

behavior_calls Stores the behaviors invoked by other behaviors			
Attribute	**Type**	**Description**	**Example**
behaviour_id	int	The id of the method	9
invoked_behaviour_id	int	The id of the invoked behabiour	11

extends Stores extend relationships between classes of a release			
Attribute	**Type**	**Description**	**Example**
superclass_id	int	The id of superclass	5
subclass_id	int	The id of the subclass	12

implements Stores implement relationships between classes of a release			
Attribute	**Type**	**Description**	**Example**
interface_id	int	The id of the interface	1
class_id	int	The id of the class implementing the interface	90

parameter Represents a parameter used in a Method or in a Function			
Attribute	**Type**	**Description**	**Example**
name	varchar	The name of the parameter	firstName
behavior_id	int	The id of the behavior to which the parameter belongs	6
code	varchar	Just used to test the analyzer. Will not be part of the final Markos project.	code..
type_class_id	int	The id of the class representing the parameter type (if it is a class of the system, NULL otherwise)	7
type	varchar	The type of the parameter. Useful if type_class_id is NULL.	String
p_order	int	The position of the parameter in the behaviour's parameter list. Starts from 0.	0

variable Represents a variable. Could be both a local variable of a method or an instance variable			
Attribute	**Type**	**Description**	**Example**
id	int	Primary Key	3
name	varchar	The name of the variable	user
type_class_id		The id of the class representing the variable type (if it is a class of the system, NULL otherwise)	7
type	varchar	The type of the variable. Useful if type_class_id is NULL.	String

specifier Stores the specifiers (e.g., public, static)			
Attribute	**Type**	**Description**	**Example**
id	int	Primary Key	3
name	varchar	The name of the specifier	public

behaviour_specifiers Stores the specifiers associated to a behaviour			
Attribute	**Type**	**Description**	**Example**
behaviour_id	int	The id of the method	9
specifier_id	int	The id of the specifier	8

variable_specifiers Stores the specifiers associated to a variable			
Attribute	**Type**	**Description**	**Example**
variable_id	int	The id of the variable	9
specifier_id	int	The id of the specifier	8

behaviour_local_variable Stores the local variables defined in a behaviour			
Attribute	**Type**	**Description**	**Example**
local_variable_id	int	The id of the variable	5
behaviour_id	int	The id of the behaviour	3

behaviour_global_variable Stores the global variables used by a behaviour			
Attribute	**Type**	**Description**	**Example**
variable_id	int	The id of the variable	1
behaviour_id	int	The id of the behaviour	4

instance_variable_class Stores the global variables defined in a class			
Attribute	**Type**	**Description**	**Example**
variable_id	int	The id of the variable	1
class_id	int	The id of the class	4

4. NON-FUNCTIONAL REQUIREMENTS

The Code Analyser is a heavy computational process. Its main non-functional requirement is related to the executions time. To date, we do not have reliable estimations of the Code Analyser execution time. However, we are confident that we should be able to satisfy *at least* the following performance requirement:

a) No more than 1 minute needed to analyse 30 Kilo Lines Of Code (KLOC).

A second non-functional requirement is related to the possibility of easily integrating in Markos new programming languages. We plan to limit the implementation of a new programming language to the implementation of a new parser, without impacting (i) the data dictionary and (ii) the classes managing the persistent information stored by the Code Analyser.

5. CODE ANALYSER CONTROL FLOW

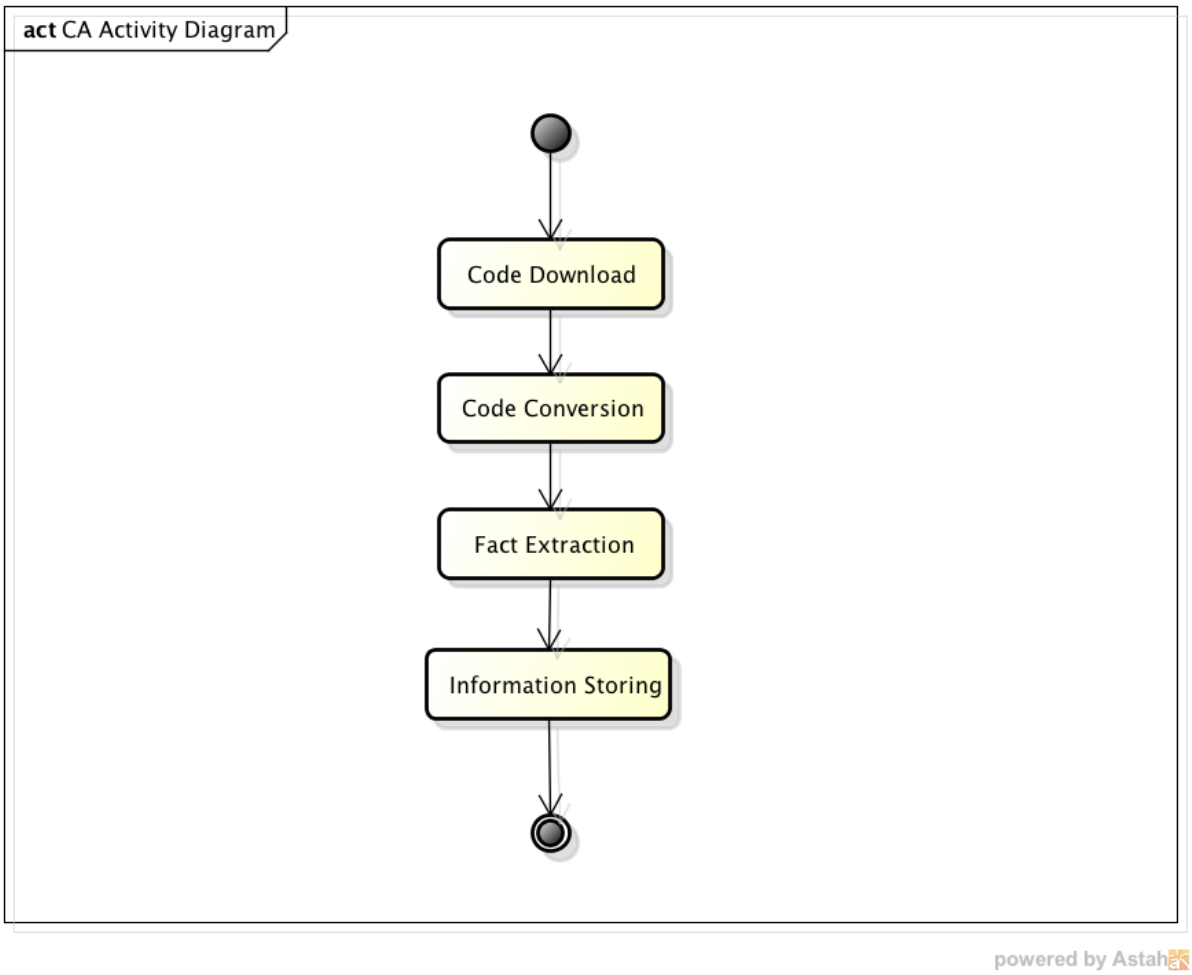

Figure 2: Code Analyser control flow

Figure 2 reports the control flow of the code analyser component when performing code analysis. In the following subsections we briefly discuss each of the performed actions.

5.1. Code Download

The *Code Downloader* is in charge of download new project releases indicated by the Crawler through the "analyseNewProjectReleases" service. The downloaded code is stored in a temporary folder. Once the code is ready to be parsed, the *Code Converter* component is notified.

5.2. Code Conversion

The *Code Converter* converts the downloaded project releases to be parsed in and XML-like format. When the *Code Converter* converts a release it (i) notifies the *Fact Extractor* that a new project is ready to be parsed and (ii) deletes the source code of the converted release.

5.3. Fact Extraction

The *Fact Extractor* extracts all info reported in the data model presented in Section 3, including those related to the license statements.

5.4. Information Storing

The *Storage Management* manages all the information in the Code Analyser database. It provides a set of interfaces (not exposed by the Code Analyser to other components) allowing to insert/delete/update/query all the info stored in the Code Analyser database. The information is mainly inserted/deleted/updated by the *Code Parser* and the *License Parser*, while the *Exposed Services* component is the one mainly exploiting the querying interfaces.

6. CONCLUSION

This document overviews the requirements behind the source code information extraction performed by the Code Analyser. In the second version of this document scheduled for the 12th month we will provide the final version of the Code Analyser data dictionary together with more reliable information about the performance requirements that the Code Analyser is able to satisfy.