4 A Plataforma de Extração

Neste capítulo, vamos apresentar a plataforma de mineração de repositórios construída neste trabalho. Esta plataforma foi desenvolvida com o intuito de endereçar algumas das limitações existentes em outras abordagens de mineração de repositórios de software. Como dito anteriormente, no escopo deste trabalho endereçamos o tratamento de grande volume de informações, a expressividade das informações relacionadas ao código-fonte e a extensibilidade das soluções propostas. Escolhemos extrair informações relacionadas ao código-fonte dos projetos de modo a testar esta nova abordagem.

Em primeiro lugar, vamos descrever os casos de uso da plataforma. Eles representam o modo como os usuários interagem com a plataforma para minerar e consultar informações de repositórios de software. Logo após, apresentamos a arquitetura da plataforma, seus módulos e como estão relacionados com as ontologias apresentadas no capítulo anterior. Depois, descrevemos como e quais tecnologias foram utilizadas na implementação da plataforma.

4.1. Casos de Uso da Plataforma

Nesta seção apresentamos e detalhamos os principais casos de uso da plataforma de extração. Abaixo, o diagrama de casos de uso:

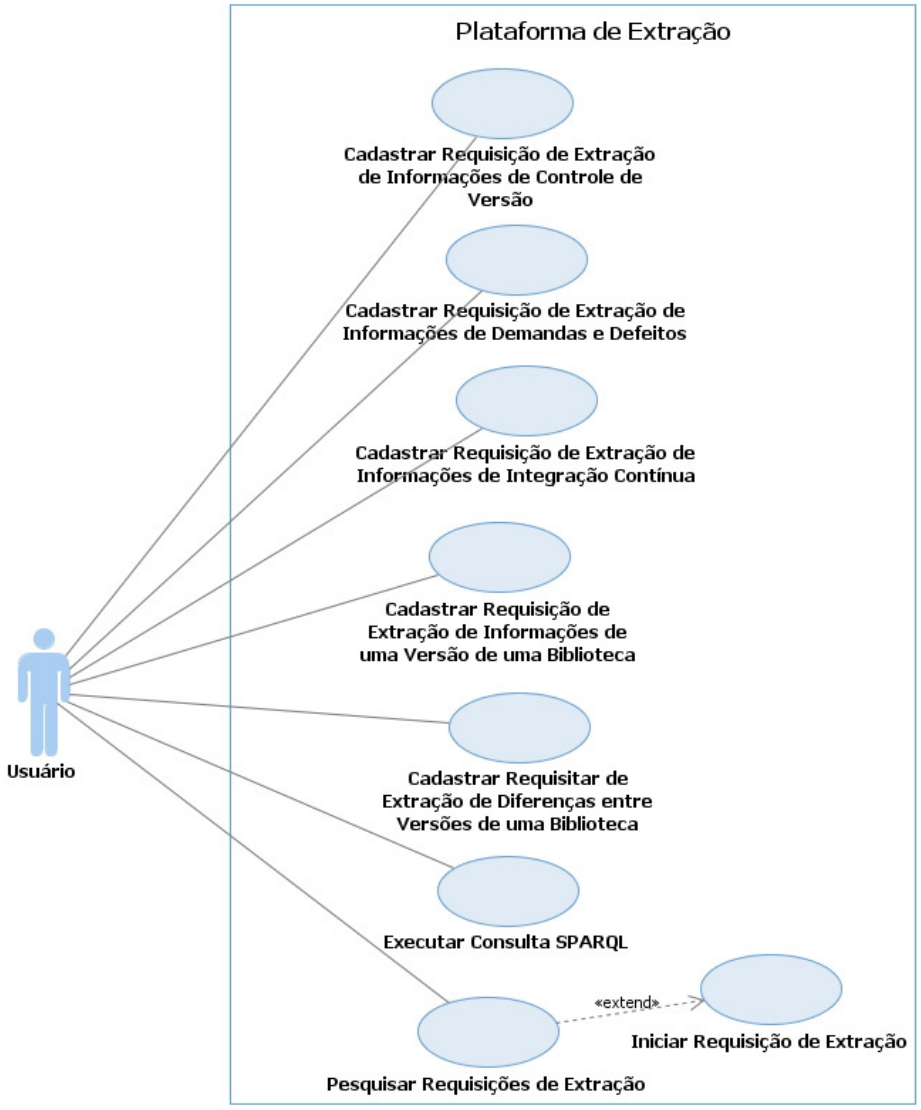

Figura 22 Diagrama de Casos de Uso da Plataforma

4.1.1. Cadastrar Requisição de Extração de Informações de Controle de Versão

Este caso de uso tem por objetivo permitir que um usuário cadastre uma solicitação da extração de informações de repositórios de controle de versão. As seguintes informações são necessárias para este procedimento: a URL do repositório a ser extraído, diretório local onde os arquivos serão armazenados, tipo de ferramenta de controle de versão e a data de início e de fim para filtrar as informações retornadas.

4.1.2. Cadastrar Requisição de Extração de Informações de Demandas e Defeitos

Este caso de uso tem por objetivo permitir que um usuário cadastre uma solicitação da extração de informações de demandas/defeitos presentes em ferramentas gerenciadoras de demandas e defeitos. As seguintes informações são necessárias para este procedimento: a URL do repositório que será acessado, os identificadores das demandas/defeitos a serem extraídas e o tipo de Gerenciador de Demandas e Defeitos.

4.1.3. Cadastrar Requisição de Extração de Informações de Integração Contínua

Este caso de uso tem por objetivo permitir que um usuário cadastre uma solicitação da extração de informações de ferramentas de Integração Contínua. As seguintes informações são necessárias para este procedimento: a URL do repositório que será acessado, a construção queque será extraída e o tipo de servidor de integração contínua.

4.1.4. Cadastrar Requisição de Extração de Informações de Versão de uma Biblioteca

Este caso de uso tem por objetivo permitir que um usuário realize o cadastro da solicitação da extração de informações sobre a versão de uma biblioteca utilizada como dependência no projeto. As seguintes informações são necessárias para este procedimento: a URL do repositório de artefatos que será acessado e a versão da biblioteca que terá informações extraídas.

4.1.5. Cadastrar Requisitar de Extração de Diferenças entre Versões de uma Biblioteca

Este caso de uso tem por objetivo permitir que um usuário realize o cadastro de uma solicitação da extração de diferenças entre duas versões de uma biblioteca. As seguintes informações são necessárias para este procedimento: a URL do repositório de artefatos e as versões que serão utilizados na extração das diferenças.

4.1.6. Executar Consulta SPARQL

Este caso de uso permite o usuário executar uma consulta SPARQL na plataforma de extração. A partir da consulta especificada, a plataforma retorna uma tabela com os resultados ou uma mensagem informando um erro quando a consulta está incorreta.

4.1.7. Pesquisar Requisições de Extração

Este caso de uso tem por objetivo permitir que o usuário pesquise as solicitações de requisição cadastradas na plataforma. As seguintes informações podem ser utilizadas para pesquisar as requisições: status, tipo de ferramenta, tipo de repositório e URL do repositório. O sistema retorna uma lista de requisições que atendam ao filtro informado. Para cada requisição retornada o sistema informa o seu status, a URL do repositório, o tipo de ferramenta, o tipo de repositório e o progresso da extração. Além disso, é possível solicitar o início do processamento de uma requisição.

4.1.8. Iniciar Requisição de Extração

Este caso de uso permite um usuário iniciar um processo de Requisição de Extração previamente cadastrado e encontrado através do cenário "Pesquisar Requisições de Requisições". Os seguintes processos podem ser iniciados:

- Processo de Extração de Informações de Controle de Versão
- Processo de Extração de Informações de uma Versão de uma Biblioteca
- Processo de Extração de Diferenças entre Versões de uma Biblioteca
- Processo de Extração de Informações de Demandas e Defeitos
- Processo de Extração de Informações Integração Contínua

4.2. Arquitetura da Plataforma

Nesta seção apresentamos a arquitetura da plataforma desenvolvida. Seus principais componentes são os Conectores de Repositórios, os Transformadores RDF e os Coordenadores de Extração. Estes elementos interagem entre si para extrair informações dos diversos repositórios de software. Na figura abaixo, apresentamos uma visão geral da plataforma e da interação destes módulos.

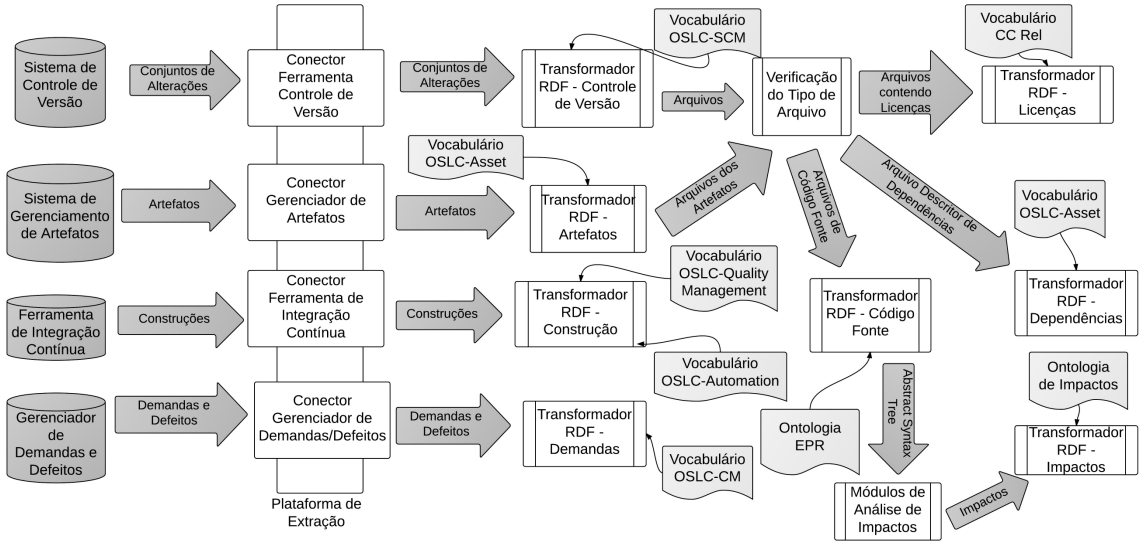

Figura 23 Visão Geral dos Processos de Extração implementados na Plataforma de Extração

4.2.1.1. Módulos Conectores de Repositórios

Os módulos conectores fornecem mecanismos para permitir o acesso de forma transparente às informações presentes em diferentes tipos e versões de repositórios de software, como os sistemas de controle de versão, as ferramentas de integração contínua, as linguagens de programação e os gerenciadores de demandas. Como mencionamos anteriormente, um dos objetivos deste trabalho é criar uma infraestrutura que seja extensível e flexível para suportar outras ferramentas e versões de forma mais simples e transparente, por este motivo cada tipo de repositório de software tem um módulo conector comum responsável pelo registro e recuperação de módulos conectores específicos quando estes são necessários para outros módulos. Abaixo listamos os módulos conectores comuns

que foram desenvolvidos e no Apêndice A descrevemos em detalhes cada um destes módulos e os conectores específicos.

4.2.1.1.1. Conector de Ferramentas de Controle de Versão

Este tipo de conector é responsável por fornecer uma interface entre as ferramentas de controle de versão e os outros módulos da plataforma que necessitam de informações destas ferramentas. Nesta versão, o conector para a ferramenta Git foi implementado.

4.2.1.1.2. Conector de Ferramentas de Integração Contínua

Este tipo de conector é responsável por fornecer a comunicação necessária entre as ferramentas de integração contínua e a plataforma de extração. Nesta versão, os conectores para as ferramentas Jenkins e Hudson foram implementados.

4.2.1.1.3. Conector de Gerenciadores de Artefatos

Este tipo de conector fornece os mecanismos necessários para a plataforma acessar artefatos e outras informações armazenados em ferramentas gerenciadoras de artefatos. Nesta versão, o conector para o ferramenta Nexus foi implementado.

4.2.1.1.4. Conector de Ferramenta de Gerenciamento de Defeitos e Demandas

Este tipo de conector é responsável por permitir que a plataforma acesse informações de ferramentas gerenciadoras de defeitos e demandas. Nesta versão, o conector para a ferramenta Jira foi implementado.

4.2.1.1.5. Conector de Linguagem de Programação

Este tipo de conector é responsável por permitir que a plataforma acesse informações relacionadas ao código-fonte escrito em linguagens de programação. Nesta versão, o conector para a linguagem Java foi implementado.

4.2.1.1.6. Conector de Ferramentas de Gerenciamento de Dependências

Este tipo de conector fornece meios de acesso para informações relacionadas às dependências de um projeto de software ou de uma versão de uma biblioteca. Nesta versão, o conector para a ferramenta Maven foi implementado.

4.2.1.1.7. Conector de Licenças

Este tipo de conector permite que informações relacionadas a licenças de software presentes em arquivos sejam recuperadas pelos módulos da plataforma. Nesta versão, o conector para a ferramenta Maven foi implementado.

4.2.1.2.Módulos Transformadores RDF

Os módulos transformadores RDF são os responsáveis por converter as informações recuperadas dos repositórios de software em dados RDF. Estes módulos utilizam as ontologias e vocabulários apresentados anteriormente para realizar esta transformação. Além disso, fornecem mecanismo para que outros módulos possam recuperar estas informações de forma transparente através de classes Java. Abaixo listamos os transformadores que foram implementados para os respectivos tipos de repositórios de software. No Apêndice B descrevemos em detalhes cada um dos transformadores desenvolvidos.

4.2.1.2.1. Transformador RDF de Controle de Versão

Este transformador é responsável por converter os dados recuperados pelos conectores de controle de versão para triplas RDF. Utiliza o vocabulário OSLC SCM em conjunto com as ontologias FOAF e Dublin Core neste processo.

4.2.1.2.2. Transformador RDF de Licença

Este transformador é responsável por converter os dados de licenciamento recuperados pelos conectores de Licenças para triplas RDF. Utiliza o vocabulário CC REL no processo de transformação.

4.2.1.2.3.Transformador RDF de Artefatos

Este transformador é responsável por converter as informações recuperadas pelos Conectores de Artefatos em triplas RDF. Utiliza o vocabulário OSLC Asset Management no processo de transformação.

4.2.1.2.4. Transformador RDF de Dependências

Este transformador é responsável por converter as informações recuperadas pelos Conectores de Dependências em triplas RDF. Utiliza o vocabulário OSLC Asset Management no processo de transformação.

4.2.1.2.5. Transformador RDF de Código-Fonte

Este transformador é responsável por converter as informações da AST de um determinado código-fonte recuperadas pelos Conectores de Linguagem de Programação para triplas RDF. Utiliza neste processo a Ontologia de Entidades, Propriedades e Relacionamentos.

4.2.1.2.6. Transformador RDF de Impacto

Este transformador é responsável por transformar as informações dos impactos computados entre duas versões da AST de um código-fonte em triplas RDF. Utiliza a Ontologia de Impactos neste processo.

4.2.1.2.7. Transformador RDF de Integração Contínua

Este transformador é responsável por transformar as informações recuperadas pelos Conectores de Ferramentas de Integração Contínua em triplas RDF. Utiliza o vocabulário OSLC Automation e Quality Management neste processo.

4.2.1.2.8. Transformador RDF de Demandas/Defeitos

Este transformador é responsável por transformar em RDF as informações recuperadas pelos Conectores de Ferramentas de Gerenciamento de

Demandas/Defeitos. Utiliza o vocabulário OSLC Change Management em seu processo de conversão.

4.2.1.3. Módulos Auxiliares

Os módulos auxiliares têm o papel de auxiliar os módulos coordenadores durante a execução de suas atividades. Atualmente existem dois módulos deste tipo: Módulo de Verificação de Tipo de Arquivo e Módulo de Análise de Impactos. No Apêndice C detalhamos estes dois módulos.

4.2.1.4. Módulos Coordenadores

Os módulos coordenadores são responsáveis por coordenar a execução dos diferentes tipos de processo de extração apresentados nos Casos de Uso da plataforma, como por exemplo, extrair informações de uma ferramenta de integração contínua ou extrair informações de um sistema de controle de versão. Estes módulos recebem um conjunto de informações necessárias para iniciar um determinado processo extração, como por exemplo, o endereço do repositório de controle de versão ou o caminho de um arquivo de código-fonte. A partir disto, eles podem interagir com módulos conectores e transformadores RDF para os auxiliaram durante o processo. Abaixo listamos os módulos deste tipo que foram construídos. No Apêndice D apresentamos os detalhes de cada um destes módulos.

4.2.1.4.1. Módulo de Coordenação de Extração de Informações de Ferramentas de Controle de Versão

Este módulo tem a responsabilidade de coordenar o acesso a um repositório de controle de versão para extrair as informações presentes nos conjuntos de alterações armazenados. Dentre estas informações destacam-se: o autor, a data de criação, as issues associadas e os arquivos afetados pelo commit. Logo após, este módulo realiza o processamento destes arquivos para extrair informações subsequentes, como as mudanças na AST de um arquivo de código-fonte, as licenças associadas com o projeto e as dependências. Abaixo a visão em alto nível do processo de extração coordenado por este módulo

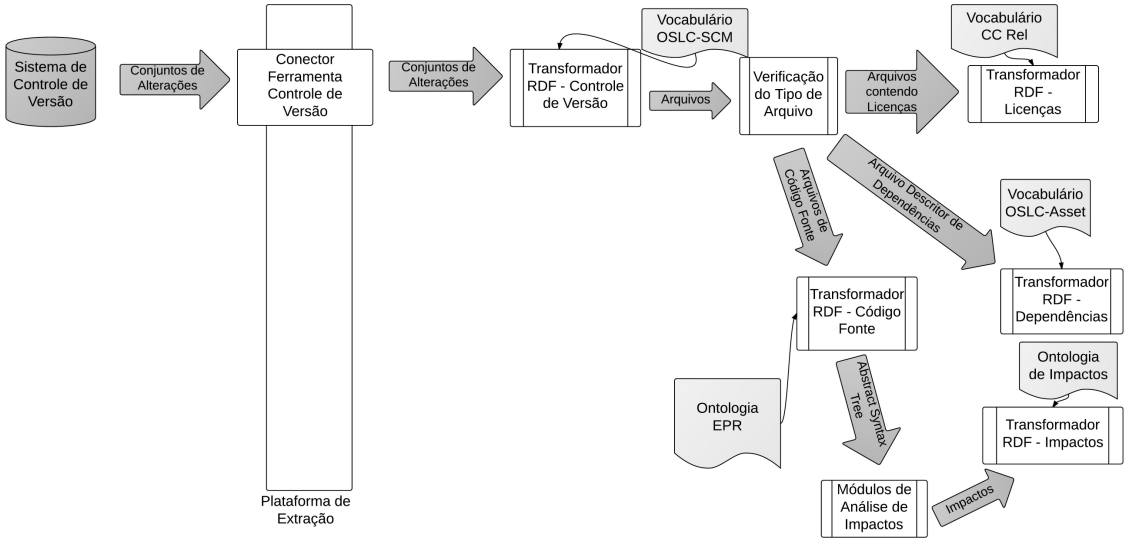

Figura 24 Visão Geral do Processo de Extração de Informações de Ferramentas de Controle de Versão

4.2.1.4.2. Módulo Coordenador do Processamento da Extração de Informações de uma Versão de uma Biblioteca

Este módulo tem a responsabilidade de coordenar o processamento de informação de uma versão de uma biblioteca utilizada como dependência de um projeto. Ele tem a função de recuperar o arquivo físico que representa uma determinada biblioteca e a partir disto recuperar seu código-fonte e suas dependências para obter outras informações. Utiliza o Conector de Artefatos para recuperar o arquivo de uma biblioteca e delega ao Conector de Dependências a responsabilidade de recuperar as dependências que a biblioteca possa ter. Abaixo representamos uma visão geral do processo de extração coordenado por este módulo.

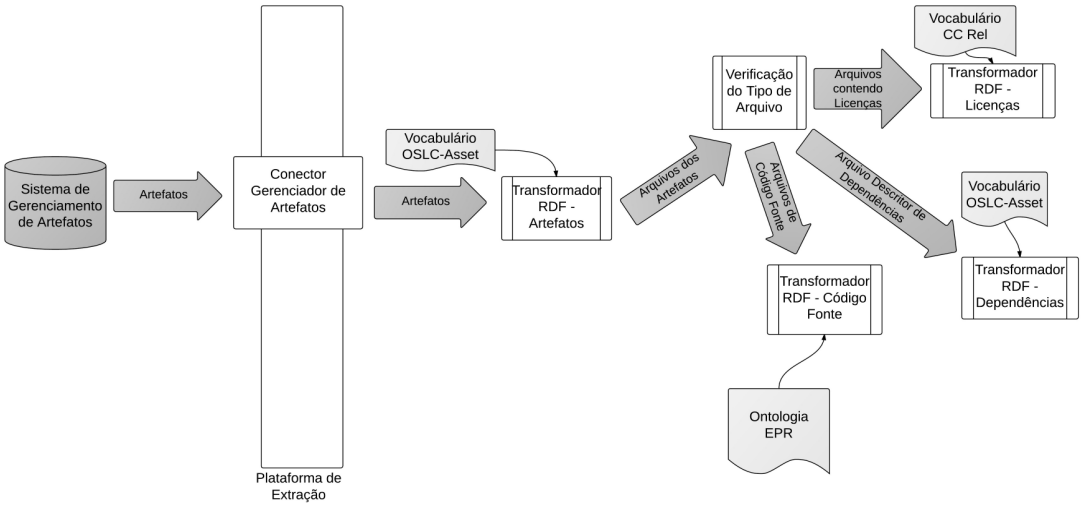

Figura 25 Visão Geral do Processo de Extração de Informações de uma Versão de uma Biblioteca

4.2.1.4.3. Módulo Coordenador de Extração de Informações de Ferramentas de Integração Contínua

Este módulo é o responsável por coordenar o processo de extração de informações de servidor de integração contínua e recuperar informações como construções executadas, conjuntos de alterações incluídos e testes executados em cada construção. Abaixo representamos uma visão geral do processo de extração coordenado por este módulo.

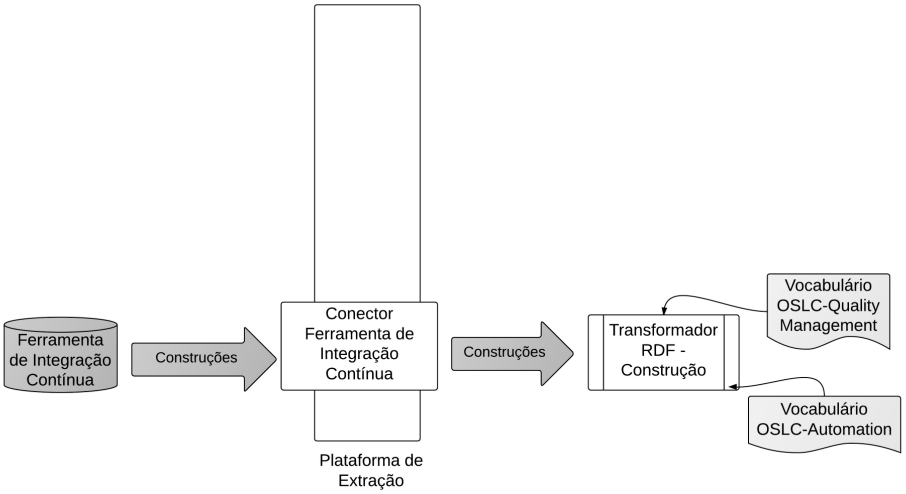

Figura 26 Visão Geral do Processo de Extração de Informações de Ferramentas de Integração Contínua

4.2.1.4.4. Módulo Coordenador de Extração de Informações de Diferenças entre Versões de uma Biblioteca

Este módulo é o responsável por coordenar o processo de extração das diferenças (impactos) na AST dos elementos de código-fonte presentes em duas versões de uma biblioteca. Abaixo representamos uma visão geral do processo de extração coordenado por este módulo.

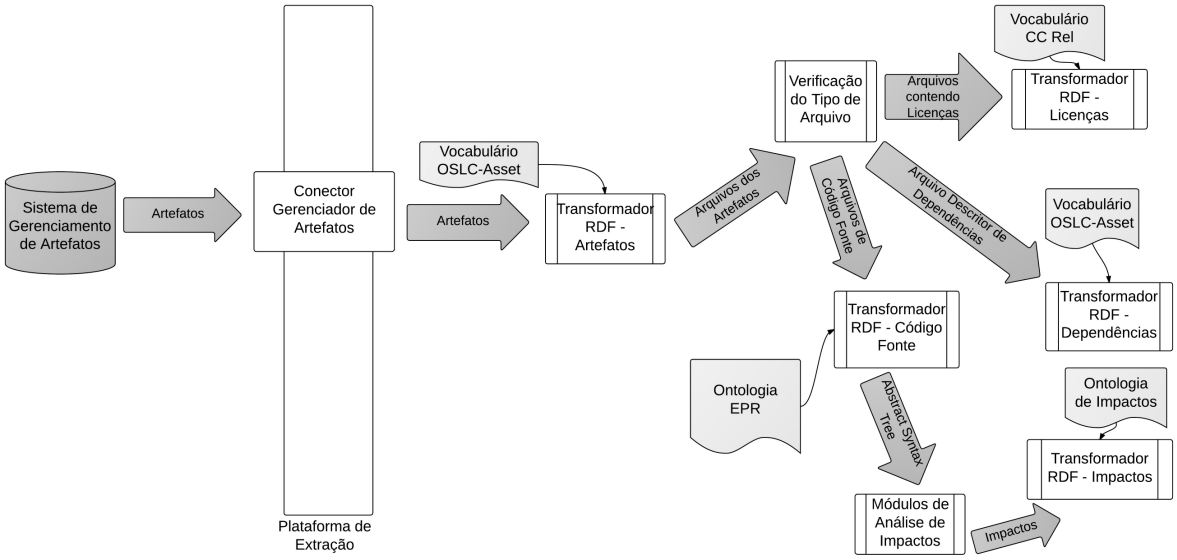

Figura 27 Visão Geral do Processo de Extração de Informações de Diferenças entre duas Versões de uma Biblioteca

4.2.1.4.5. Módulo Coordenador de Extração de Informações de Ferramentas de Gerenciamento de Demandas/Defeitos

Este módulo é o responsável por coordenar o processo de extração das demandas e defeitos presentes em ferramentas de gerenciamento de demandas/defeitos. Abaixo representamos uma visão geral do processo de extração coordenado por este módulo.

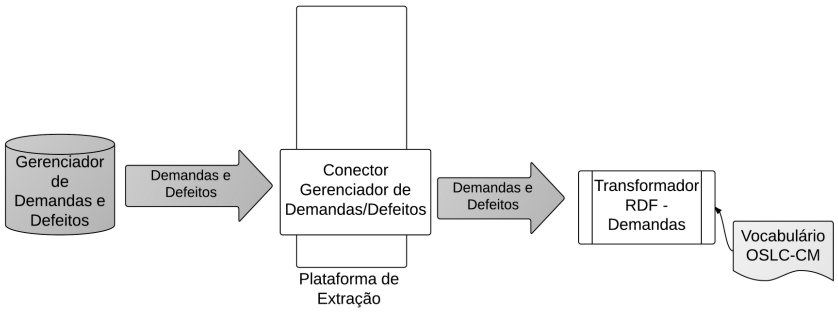

Figura 28 Visão Geral do Processo de Extração de Informações de Ferramentas de Gerenciamento de Demandas/Defeitos

4.3. Arquitetura de Implementação

Nesta seção vamos descrever as principais tecnologias utilizadas no desenvolvimento da plataforma de extração. Além disso, vamos apresentar como estas tecnologias foram utilizadas na construção dos módulos apresentados na seção anterior.

4.3.1. Tecnologias de Infraestrutura

Nesta seção vamos descrever as tecnologias utilizadas na infraestrutura da plataforma, ou seja, as tecnologias que permeiam todos os módulos construídos.

4.3.1.1. Java

Java é uma linguagem de programação orientada a objetos, multipropósito e criada originalmente pela Sun Microsystems, que é conhecida por sua premissa *Write Once*, *Run Anywhere* (Escreva uma vez, rode em qualquer lugar). Isto significa que o código Java escrito e compilado em uma plataforma não necessita ser recompilado quando executado em outra.

A plataforma Java consiste das interfaces de programação de aplicação (APIs)[60] e da máquina virtual Java (JVM). As primeiras são bibliotecas de código compilado que podem ser utilizadas nos programas escritos em Java. Elas fornecem funcionalidades normalmente utilizadas na grande maioria dos programas, como manipulação de arquivos, gerenciamento de concorrência e etc.

Já a máquina virtual Java, permite que o *bytecode*[61] gerado pela compilação de uma aplicação seja interpretado e possa ser executado em qualquer plataforma que tenha uma implementação de JVM. Pois ela é a responsável por converter as instruções contidas nos *bytecodes* para instruções nativas do sistema operacional da plataforma.

4.3.1.2. OSGi[62]

OSGi é um conjunto de especificações que definem um sistema de componentes e plataforma de serviços para a linguagem Java. Estas especificações permitem a criação de aplicativos componentizados e dinâmicos que podem ser remotamente instalados, instalados, iniciados, parados, atualizados e desinstalados sem a necessidade de reiniciar a JVM onde estão executando. Este gerenciamento do ciclo de vida das aplicações é feito através de um conjunto de APIs que além de fornecerem estas funcionalidades, permitem que as aplicações (ou seus componentes) possam registrar serviços, receber notificações de mudanças no ambiente de execução e etc.

Atualmente existem diversas implementações as especificações OSGi, sendo que as mais conhecidas são os frameworks Apache Felix[63] e Eclipse Equinox[64].

Camadas

A arquitetura de qualquer framework que implementa o padrão OSGI pode ser dividida em três grandes camadas: Módulos, Serviços e Ciclo de Vida. Abaixo apresentamos uma visão geral da arquitetura definida por esta especificação e os detalhes de cada uma destas camadas:

[60] http://en.wikipedia.org/wiki/Application_programming_interface
[61] http://en.wikipedia.org/wiki/Bytecode
[62] http://www.osgi.org/Technology/WhatIsOSGi
[63] http://felix.apache.org/
[64] http://www.eclipse.org/equinox/

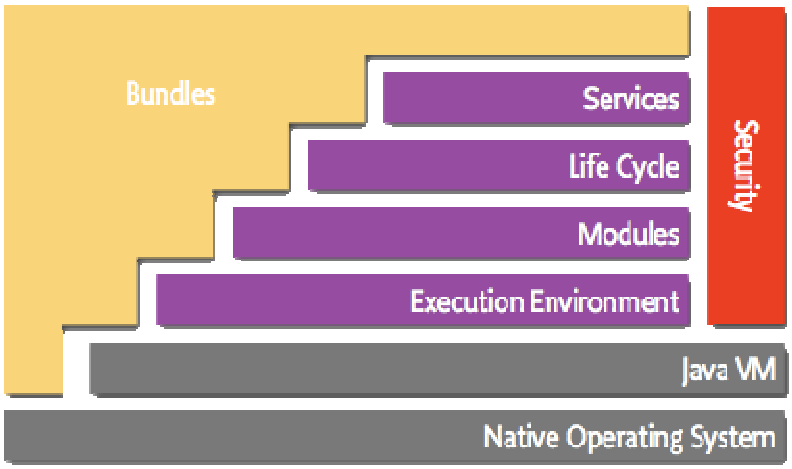

Figura 29 Arquitetura definida pela especificação OSGi

1. Módulos

Modularidade é um dos principais objetivos desta especificação e é representada através do conceito de *bundle*. Um Bundle é um Java Archive (JAR)[65] com determinadas características que permitem definir que classes e recursos são disponibilizados por aquele arquivo Jar para outros bundles/jars. Apesar de existirem diversas boas práticas e recomendações para o desenvolvimento de aplicativos modulares com baixo acoplamento e alta coesão, a plataforma Java não fornece um mecanismo para reforçar isto em tempo de execução. Isto significa que Jar presente no *classpath*[66] de uma aplicação fornece acesso a todas as suas classes, até mesmo as classes que não fazem parte da API inicialmente pensada pelo desenvolvedor do pacote. Isto pode criar um alto acoplamento entre os módulos desenvolvidos durante um projeto e funcionalidades que tem alta chance de serem modificadas sem aviso prévio pelo fato de não terem sido pensadas para uso externo. A possibilidade de ocultar código e explicitamente definir o que é visível externamente fornece diversas vantagens para o desenvolvimento de aplicações modulares, corroborando, por exemplo, com um dos princípios dos padrões de projeto que é programar para interface não para sua implementação. Abaixo se pode observar o diagrama de

[65] http://docs.oracle.com/javase/6/docs/technotes/guides/jar/
[66] http://docs.oracle.com/javase/6/docs/technotes/tools/windows/classpath.html

estrutura composta[67] do Módulo Conector de Ferramentas de Controle de Versão, que é o responsável por fornecer a interface que os vários conectores de ferramentas de controle de versão devem implementa de forma que possam ser acessados transparentemente pela plataforma.

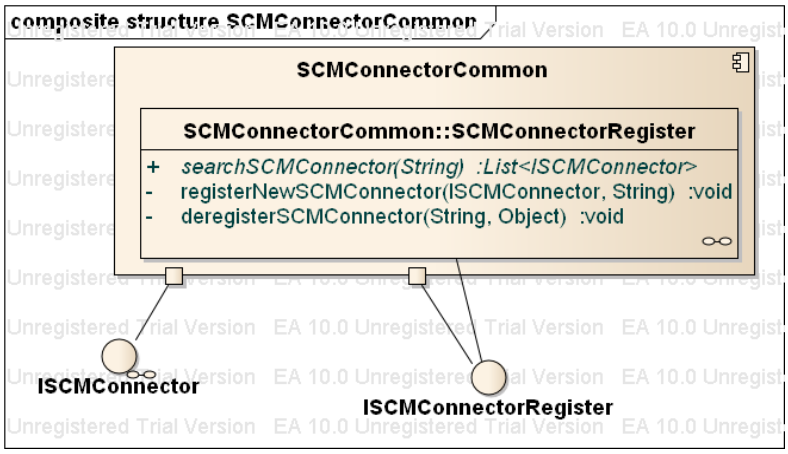

Figura 30 Interfaces que o módulo Conector de Ferramentas de Controle de Versão fornece

A forma de um *bundle* definir estas e outras informações é através de um arquivo conhecido como MANIFEST.MF. Este arquivo permite definir dentre outras coisas, o nome do *bundle*, sua versão, que pacotes são disponibilizados, que pacotes são necessários para seu funcionamento e etc. Abaixo apresentamos um exemplo do MANIFEST.MF do *bundle* que define o módulo Conector de Ferramentas de Controle de Versão:

Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: com.semanticminer.scm.connector
Bundle-SymbolicName: com.semanticminer.scm.connector
Bundle-Version: 1.0.0.qualifier
Export-Package: com.semanticminer.scm.connector.api;version="1.
0.0.qualifier"

Figura 31 Exemplo de um MANIFEST.MF de um bundle

[67] http://www.uml-diagrams.org/composite-structure-diagrams.html

2. Serviços

Além de permitir a modularização das aplicações, OSGi introduz o conceito de serviços à plataforma Java. Cada *bundle* pode criar um objeto e registrá-lo no Service Registry da plataforma OSGi sob a forma de uma ou mais interfaces. Com isso, outros *bundles* podem ir a este registro e recuperar o objeto que representa determinado serviço que desejam acessar, ou então, podem utilizar mecanismos para aguardar o aparecimento de um determinando serviço para então utilizá-lo. Abaixo apresentamos uma figura que ilustra o mecanismo de serviços oferecidos por esta especificação:

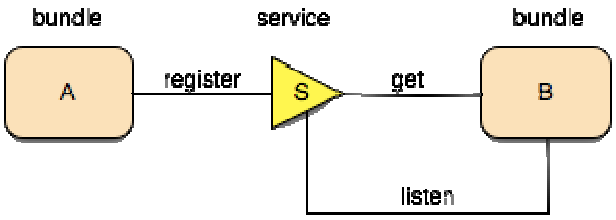

Figura 32 Exemplo da dinâmica entre dois bundles e um serviço fornecido por um deles

A especificação também possibilita que diversos *bundles* forneçam serviços sob a mesma interface, como ocorre nos módulos conectores, que disponibilizam uma interface de serviço para que outros módulos possam registrar diferentes implementações e assim suportar o acesso a diversos tipos e versões da mesma categoria de repositório de software, como os sistemas de controle de versão. Abaixo apresentamos o trecho de código do *SCMConnectorRegister* onde o bundle do Modulo Conector de Ferramentas de Controle de Versão recebe a notificação de um novo serviço conector e armazena sua referência:

```
@Reference(multiple = true, service = ISCMConnector.class, dynamic = true,
unbind="removeService")
public void addScmConnector(ServiceReference reference) {


            String repositoryType = (String)
reference.getProperty("scm_type");


      registerNewSCMConnector((ISCMConnector)reference.getTarget(),repo
sitoryType);
      }


public void registerNewSCMConnector(ISCMConnector connector,String type
) {

            if (this.sRScmRepositories == null) {
                  this.sRScmRepositories = new HashMap<String,
ServiceReference>();
            }
            this.sRScmRepositories.put(repositoryType, connector);
      }
```

Figura 33 Código que registra novos serviços do tipo ISCMConnector

Como o método *addSCMConnector* está anotado com a anotação @Reference, em tempo de compilação, a ferramenta aQute BND[68] gera um arquivo XML que é lido pela plataforma OSGi e que faz com que toda vez que um novo serviço com a interface ISCMConnector seja registrado, este método deve ser invocado passando a referência do serviço. Esta referência, além de fornecer acesso ao objeto do serviço propriamente dito, possibilita acesso a diversas propriedades que contém as características do serviço disponibilizado. Estas propriedades podem então ser utilizadas durante a pesquisa de algum serviço que tenha diversas implementações na aplicação, como é o caso dos conectores. No exemplo acima, a propriedade customizada "scm_type" foi utilizada para identificar os diversos conectores de acordo com a ferramenta de controle de versão a qual fornecem acesso.

Entretanto, o serviços na plataforma OSGi são entidade dinâmicas, ou seja, podem ser registrados ou removidos durante a execução da aplicação. Isto está intrinsicamente relacionado com a dinamicidade do próprio bundle na plataforma, pois ele pode ser instalado, removido ou atualizado em tempo de execução. Por este motivo, um framework que implementa a especificação OSGi deve fornece

[68] http://aqute.biz/Code/Bnd

meios para que os *bundles* possam ser notificados quando um serviços é removido da plataforma. Abaixo apresentamos o trecho de código responsável por remover a referência para um Conector de Controle de Versão que não será mais utilizado na plataforma:

```
public void removeScmConnector(ServiceReference reference) {


            String repositoryType = (String)
reference.getProperty("scm_type");

            deregisterSCMConnector(repositoryType,
(ISCMConnector)reference.getTarget());
      }


public void deregisterSCMConnector(String type, Object o ) {

            if (this.sRScmRepositories == null) {
                  return;
            }

            this.sRScmRepositories.remove(repositoryType);
      }
```

Figura 34 Código que recebe notificação da remoção de um serviço do tipo ISCMConnector e remove do SCMConnectorRegister.

Além dos serviços que os desenvolvedores podem registrar na plataforma, a especificação define diversos serviços comuns que devem ser fornecidos por um framework que implemente a especificação, como por exemplo, serviços de Log, acesso JDBC, JMX e etc.

3. Ciclo de Vida

A camada de ciclo de vida é a responsável pela dinamicidade dos *bundles*, ou seja, permite que eles possam ser dinamicamente instalados, iniciados, parados, atualizados e desinstalados. Para isto ela fornece uma infraestrutura de suporte e uma API de ciclo de vida que pode ser acessada tanto internamente, como externamente, através do protocolo Telnet, por exemplo.

4.3.1.3. Blueprint OSGi[69]

Blueprint é uma especificação que define um modelo de programação não invasivo de injeção de dependências para a plataforma OSGi. Esta especificação tem por objetivo facilitar a manipulação de serviços dentro do ambiente dinâmico do OSGi, onde os serviços podem ser registrados e removidos a qualquer momento. Esta especificação define um conjunto de arquivos XML onde os serviços requeridos e disponibilizados por um bundle são descritos. Quando um bundle que contém estes arquivos é iniciado, o bundle responsável pelo container Blueprint faz a leitura destes arquivos e realiza todas as operações necessárias para instanciar e registrar os serviços disponibilizados e por recuperar todos os serviços necessários para o funcionamento do bundle.

Toda a comunicação entre um bundle Blueprint e um determinado serviço utilizado é interceptado por este container, que é o responsável por tratar as situações que normalmente ocorrem dentro de um sistema OSGi, como o surgimento de um novo serviço de maior prioridade ou a remoção de um serviço que estava sendo utilizado. Isto permite que os bundles utilizem meio uniforme para lidar com a dinâmica dos serviços na plataforma OSGi.

Abaixo um exemplo do XML de configuração do serviço fornecido pelo módulo Conector Git. Este serviço fornece acesso às informações de repositórios criados com a ferramenta de Controle de Versão Git através da interface comum fornecida pelo Módulo Conector SCM.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:osgi="http://www.eclipse.org/gemini/blueprint/schema/blueprint"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.eclipse.org/gemini/blueprint/schema/blueprint
       http://www.eclipse.org/gemini/blueprint/schema/blueprint/gemini-
blueprint.xsd">

<osgi:service  ref="gitSCMImpl"
interface="com.semanticminer.scm.common.ISCMConnector" >

</osgi:service>

    </beans>
```

Figura 35 Trecho do Arquivo XML de Configuração do serviço fornecido pelo Módulo Conector Git

[69] http://wiki.osgi.org/wiki/Blueprint

4.3.1.4. Akka[70]

O Modelo de Atores é uma abstração que foi concebida como uma forma de modelar problemas que exploraram grande paralelismo computacional através de dezenas ou centenas de processadores trabalhando independentemente. A figura do ator neste tipo de sistema é a sua principal primitiva, porém, do mesmo modo que os atores humanos, eles agem seguindo um comportamento pré-definido em reposta a uma mensagem recebida, como um roteiro. Isto guia todas as decisões que este ator deve tomar, como por exemplo, se é necessária a criação de mais atores ou se ele deve responder a mensagem recebida.

Akka é um framework Java para a criação de aplicações concorrentes, escaláveis e tolerantes a falhas e que fornece uma arquitetura baseada em atores. Na plataforma de extração, alguma das funcionalidades foram implementadas a partir da utilização dos mecanismos fornecidos por este framework, como a possibilidade de iniciar diversos processos de extração que executam de forma paralela ou realizar a pesquisa e o download das diversas dependências de um projeto.

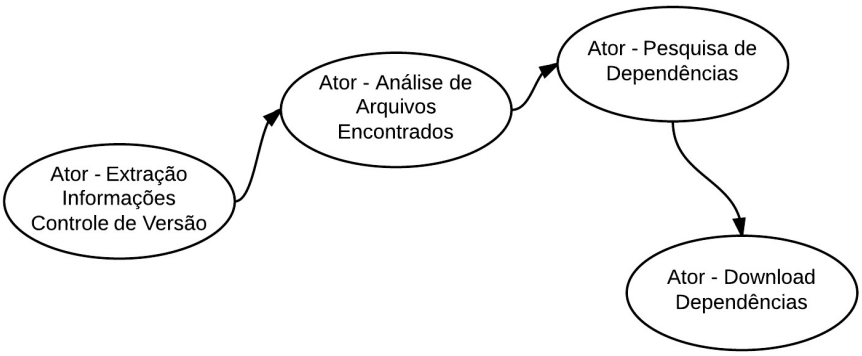

Figura 36 Parte dos Atores criados para o processo de Extração de Informações de Ferramentas de Controle de Versão

4.3.2. Tecnologias para a persistência dos Dados RDF

Nesta seção vamos descrever as tecnologias utilizadas na persistência dos dados RDF que são coletados durante um processo de extração.

[70] http://akka.io/

4.3.2.1. Sesame OpenRDF[71]

Sesame OpenRDF é um framework de código-aberto para o processamento de RDF. Ele inclui mecanismos para *parsing*, armazenamento, inferência e consultas de dados RDF. Esta ferramenta pode ser implantada em diversos sistemas de armazenamento, como banco de dados relacionais, banco em memória, sistema de arquivos e etc., pois fornece uma API, chamada de SAIL, que permite abstrair as funcionalidades do framework com o tipo de armazenamento utilizado. Além disso, suporta a execução de consultas descritas no padrão SPARQL ou no formato interno do próprio framework, o SeRQL.

4.3.2.2.OWLIM[72]

OWLIM é um repositório semântico para armazenar e manipular grandes quantidades de dados RDF. Ele implementa a API SAIL definida pelo framework Sesame OpenRDF e é composto por três componentes distintos:

- Um banco de dados RDF para o armazenamento, recuperação e atualização de triplas RDF.
- Um mecanismo de inferência que utiliza regras para inferir novo conhecimento a partir de asserções existentes.
- Um mecanismo de consulta para acessar as informações armazenadas na base ou inferidas.

4.3.3.Tecnologias de Apoio aos Conectores

Nesta seção vamos descrever as tecnologias utilizadas nos conectores que fornecem acesso aos repositórios de software considerados nesta versão da plataforma.

4.3.3.1. Eclipse JGit[73]

Git[74] é um dos repositórios de software para o qual foi implementado um conector para extrair informações, conforme demonstrado anteriormente. Ele é um

[71] http://www.openrdf.org/
[72] http://www.ontotext.com/owlim
[73] http://www.eclipse.org/jgit/

sistema de controle de versão distribuído e de código-aberto que foi desenhado inicialmente para controlar o código-fonte do kernel do sistema operacional Linux. Atualmente, passou a ser bastante utilizado em projetos de código aberto, como os da fundação Eclipse[75], da fundação Apache[76] e projetos no Github.

JGit é um projeto da fundação Eclipse que implementa o controle de versão do Git na linguagem de programação Java. As seguintes funcionalidades são implementadas:

- Rotinas de acesso aos repositórios
- Protocolos de Rede
- Algoritmos para controle de versão

4.3.3.2. Jira Rest Client[77]

JIRA[78] é uma ferramenta proprietária desenvolvida pela Atlassian para o acompanhamento de defeitos e issues e para o gerenciamento de projetos. Atualmente é bastante popular em projetos de código-aberto, sendo o gerenciador de demandas oficial da fundação Apache.

O Jira fornece uma API REST[79] para qualquer aplicativo que necessite interagir com suas funcionalidades de forma programática, como a integração do Jira com determinada ferramenta de controle de versão ou a automatização de algum procedimento, como a criação automática de issues.

A biblioteca Jira Rest Client fornece uma implementação escrita em Java para a utilização desta API. Isto facilita a interação e integração de aplicativos com o Jira, pois esta biblioteca remove a necessidade de navegação pelas URIs da API, a preparação de requisições e o *parsing* do retorno da chamada da API.

.

4.3.3.3. Jenkins Rest API[80]

Jenkins[81] é uma ferramenta de integração contínua escrita em Java e de código aberto. Ela fornece suporte para diversas ferramentas de controle de

[74] http://git-scm.com/
[75] http://eclipse.org/org/community_survey/Survey_Final_Results_2012.xls
[76] http://git.apache.org/
[77] https://ecosystem.atlassian.net/wiki/display/JRJC/Home
[78] https://www.atlassian.com/software/jira
[79] http://en.wikipedia.org/wiki/Representational_state_transfer
[80] https://wiki.jenkins-ci.org/display/JENKINS/Remote+access+API

versão, como CVS, Subversion, Git, Mercurial, Perforce, Clearcase e Team Concert e pode executar desde scripts em Ant a projetos Maven ou scripts Shell e arquivos bat Windows. Existem diversas formas de se iniciar uma construção: através da detecção de alterações nos arquivos que estão armazenados no sistema de controle de versão associado, agendamento, ao término de outras construções ou manualmente.

O Jenkins fornece uma API de acesso REST que possibilita grande parte de suas funcionalidades serem utilizadas remotamente, isto inclui: requisição de construções, cópia de definições de construção e recuperação de informações da execução de uma determinada construção.

4.3.3.4. Biblioteca Eclipse JDT

O Eclipse JDT é um projeto para adicionar à plataforma Eclipse todas as funcionalidades necessárias de uma IDE para aplicações Java. O plug-in JDT fornece um compilador incremental e uma API que permite a navegação estruturada pela AST de códigos escritos na linguagem Java. Isto permite que ferramentas que utilizem esta biblioteca possam extrair informações de classes, métodos e atributos na forma de interações com esta API, sem a necessidade de mecanismos para inspecionar *bytecode* ou para compilar o código. Esta funcionalidade foi utilizada pelo conector da Linguagem Java para recuperar informações durante o processo de extração de informações do controle de versão e de informações de bibliotecas.

4.3.3.5. Apache Maven API[82]

O Apache Maven[83] é uma ferramenta de automação de construção projetada primariamente para projetos de software escritos em Java. Baseia-se no conceito de Project Object Model (POM), que é um arquivo XML onde são configuradas as principais informações do projeto, como suas dependências, os módulos que serão construídos, a ordem de construção destes componentes, plug-ins que devem ser executados e etc. Além disso, fornece tarefas pré-definidas para as

[81] http://jenkins-ci.org/
[82] http://maven.apache.org/ref/3.1.0/maven-plugin-api/
[83] http://maven.apache.org/

tarefas que normalmente são executadas em um projeto, como compilação e empacotamento.

O Maven foi construído utilizando uma arquitetura baseada em plug-ins, o que permite a integração com um grande número de ferramentas. Para isso, fornece uma API que permite estes plug-ins recuperarem informações do POM e interagirem com a construção. No contexto deste trabalho, a API foi utilizada para recuperar as dependências do projeto e os repositórios onde estão localizadas.

4.3.3.6. Eclipse Aether[84]

Eclipse Aether é uma biblioteca para trabalhar com repositórios de artefatos. Ele é um framework comum para lidar com a publicação e a recuperação de artefatos, deixando detalhes, como tipo de protocolo de transporte e especificidades do repositório remoto para extensões desenvolvidas para cada tipo de ferramenta gerenciadora de artefatos. Por exemplo, o acesso a repositórios que armazenam bibliotecas utilizadas por projetos Maven é implementado através do plugin Maven-Aether.

[84] http://eclipse.org/aether/