7 Referências Bibliográficas

ALWIS, B.; MURPHY, G. Answering conceptual queries with Ferret. **Proceedings of the 30th International Conference on Software Engineering (ICSE'08)**, Nova Iorque, p. 21-30, 2008.

BERNERS-LEE, T; HENDLER, JAMES; LASSILA, ORA. The Semantic Web. **Scientific American**, v. 284, n. 5, p. 34-43, Maio 2001.

CLARK, MIKE. **Pragmatic Project Automation: How to Build, Deploy, and Monitor Java Applications.** Raleigh: Pragmatic Bookshelf, 2004.

FRITZ, T.; MURPHY, G. Using information fragments to answer the questions developers ask. **Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE '10)**, Nova Iorque, p. 175-184, 2010.

HEBELER, J. et al. **Semantic Web Programming.** Nova Iorque: John Wiley & Sons, 2011.

HOLT, R.C.; WINTER, A.; SCHURR, A. GXL: toward a standard exchange format. **Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00)**, v. 1, Brisbane, p. 162-171, 2000.

HYLAND-WOOD, D; CARRINGTON, D; KAPLAN, S. Toward a Software Maintenance Methodology using Semantic Web Techniques. **Proceedings of the Second International IEEE Workshop on Software Evolvability (SOFTWARE-EVOLVABILITY'06)**, Washington, p. 23-30, 2006.

HYLAND-WOOD, D; CARRINGTON, D; KAPLAN, S. A Semantic Web Approach to Software Maintenance. **Proceedings of the 15th International Conference on Knowledge Engineering and Knowledge Management (EKAW)**, Podebrady, 2006.

HYLAND-WOOD, D. **Metadata Foundations for the Life Cycle Management of Software Systems**. Queensland, 2008. 221p. Tese - School of Information Technology and Electric Engineering, The University of Queensland.

KEIVANLOO, I. et al. Towards sharing source code facts using linked data. **Proceedings of the 3rd International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation (SUITE '11)**, Nova Iorque, p. 25-28, 2011.

KEIVANLOO, I. et al. A Linked Data platform for mining software repositories. **Proceedings of the 9th Working Conference on Mining Software Repositories**, Zurique, p. 32-35, 2012.

KIEFER, C.; BERNSTEIN, A.; TAPPOLET, J. Mining Software Repositories with iSPAROL and a Software Evolution Ontology. **Proceedings of the 4th Working Conference on Mining Software Repositories**, Minneapolis, p. 20-26, 2007.

LATOZA, T; MYERS, B. Hard-to-answer questions about code. **Proceedings of the Evaluation and Usability of Programming Languages and Tools (PLATEAU '10)**, Nova Iorque, p. 8-14, 2010.

LETHBRIDGE, T.; ANQUETIL, N. **Architecture of a Source Code Exploration Tool: A Software Engineering Case Study**. University of Ottawa. Ottawa. 2007.

LI, Y; ZHANG, H. Integrating software engineering data using semantic web technologies. **Proceedings of the 8th Working Conference on Mining Software Repositories (MSR '11)**. Nova Iorque, p. 211-214, 2011.

OSSHER, J. et al. SourcererDB: An aggregated repository of statically analyzed and cross-linked open source Java projects. **Proceedings of the 9th Working Conference on Mining Software Repositories**, Zurique, p. 183-186, 2012.

RIVA, C. Reverse architecting: an industrial experience report. **Proceedings of the Seventh Working Conference on Reverse Engineering**. Brisbane, p. 42-50, 2000.

SEGARAN, T.; EVANS, C.; TAYLOR, J. **Programming the Semantic Web**. Nova Iorque: O'Reilly Media, 2009.

SOMMERVILLE, I. **Software engineering**. 9. ed. University of St Andrews, Scotland: Wesley, Addison, 2011.

TAPPOLET, J. **Mining Software Repositories - A Semantic Web Approach**. Zurique, 2007. 86p. Tese - Department of Informatics, University of Zurich.

TAPPOLET, J. Semantics-aware Software Project Repositories, **Proceedings of the ESWC 2008 Ph.D. Symposium**. Tenerife, 2008.

TAPPOLET, J; KIEFER, C; BERNSTEIN, A. Semantic web enabled software analysis. **Journal of Web Semantics: Science, Services and Agents on the World Wide Web**, v. 8, p. 225-240, Julho 2010.

WITTE, R.; ZHANG, Y.; RILLING, J. Empowering software maintainers with semantic web technologies. **Proceedings of the 4th European conference on The Semantic Web: Research and Applications (ESWC '07)**. Tyrol, p. 37–52, 2007.

WÜRSCH, M. et al. Fostering synergies: how semantic web technology could influence software repositories. **Proceedings of 2010 ICSE Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation (SUITE '10).** Nova Iorque, p. 45-48, 2010.

Apêndice A – Módulos Conectores Comuns

Módulo Conector Comum de Ferramentas de Controle de Versão

Este módulo fornece acesso aos conectores existentes para ferramentas de controle de versão. Para isso, prove a interface *ISCMConnector* (**Figura 42**) que é implementada pelos módulos conectores específicos de determinada ferramenta, além de também fornecer meios para recuperar e registrar novos conectores de ferramentas de controle de versão. Isto garante transparência para os outros componentes da plataforma que necessitam acessar informações destes tipos de repositórios de software, não sendo necessário depender de bibliotecas ou APIs específicas de cada ferramenta.

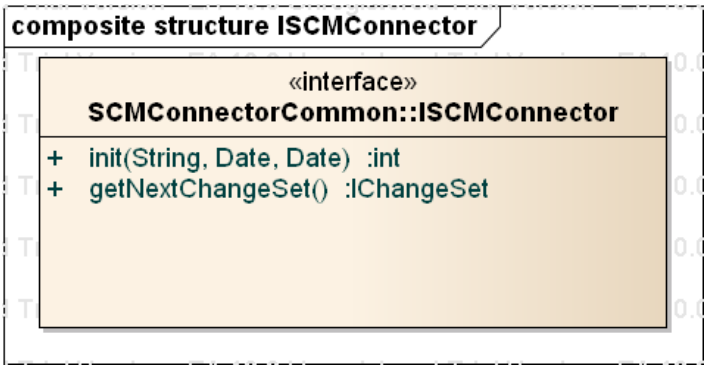

Figura 42 Interface ISCMConnector

A Interface *ISCMConnector* fornece dois métodos que devem ser implementados pelos conectores específicos:

init: Método responsável por inicializar a conexão com o repositório em questão. Recebe como parâmetro o endereço do repositório e uma data de início e de fim para filtrar os conjuntos de alterações que se deseja extrair. Retorna o total de conjuntos de alterações encontrados.

getNextChangeSet: Método utilizado para recuperar o próximo Conjunto de Alterações presente na listagem encontrada pelo método **init.**

Além desta interface, o módulo comum fornece mecanismos para o registro e recuperação de conectores. Isto é feito através da classe **SCMConnectorCommon** e de sua interface **ISCMConnectorCommon**.

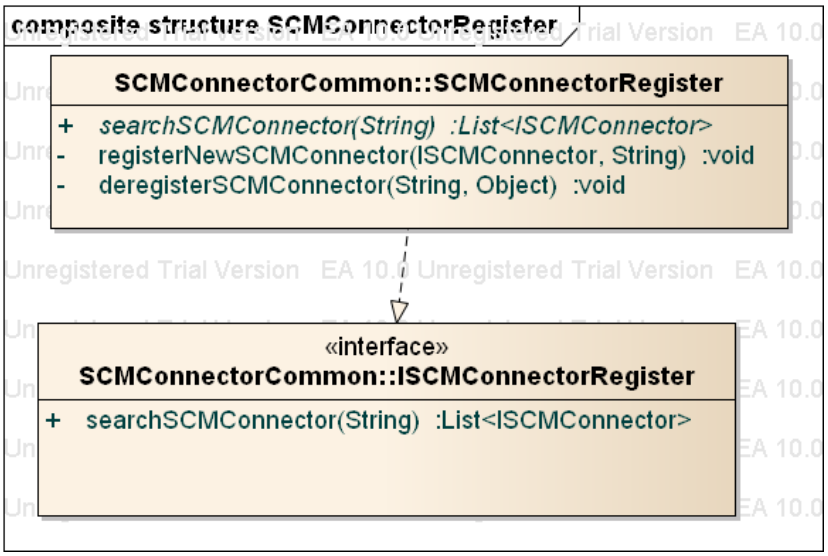

Figura 43 A classe SCMConnectorRegister e a interface ISCMConnectorRegister

A classe *SCMConnectorRegister* funciona como um listener do padrão de projetos Observer[91] que recebe notificações de novos módulos conectores que fornecem classes que implementam a interface *ISCMConnector* ou notificações de módulos conectores removidos. Utiliza o método registerNewSCMConnector para o primeiro caso e o método deregisterSCMConnector para o segundo.

Já a interface *ISCMConnectorRegister* implementada pela classe *SCMConnectorRegister* fornece o método searchSCMConnector para que os clientes externos possam encontrar um conector de ferramenta de controle de versão específico registrado na plataforma. Abaixo apresentamos o diagrama UML de estrutura composta deste módulo (**Figura 44**) e um diagrama UML de sequência (**Figura 45**) que demonstra um exemplo de interação entre um cliente, o registro de conectores e um conector específico.

[91] http://en.wikipedia.org/wiki/Observer_pattern

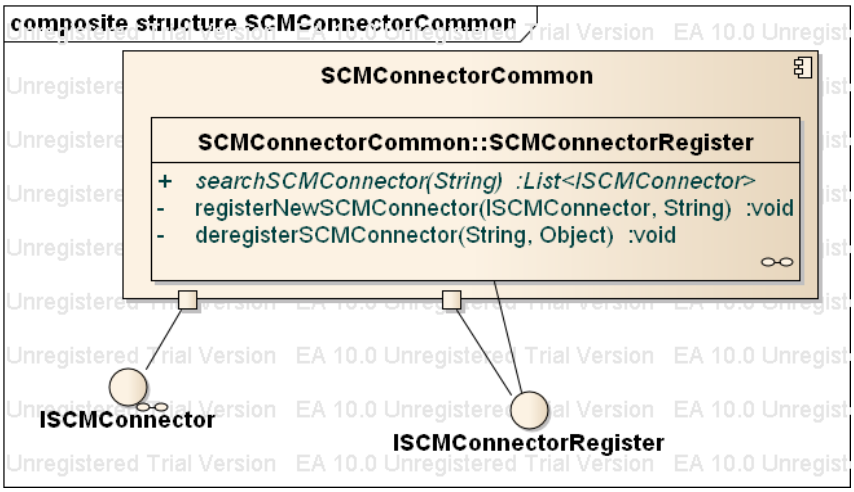

Figura 44 Diagrama de Estrutura Composta do Módulo Conector de Ferramenta de Controle de Versão

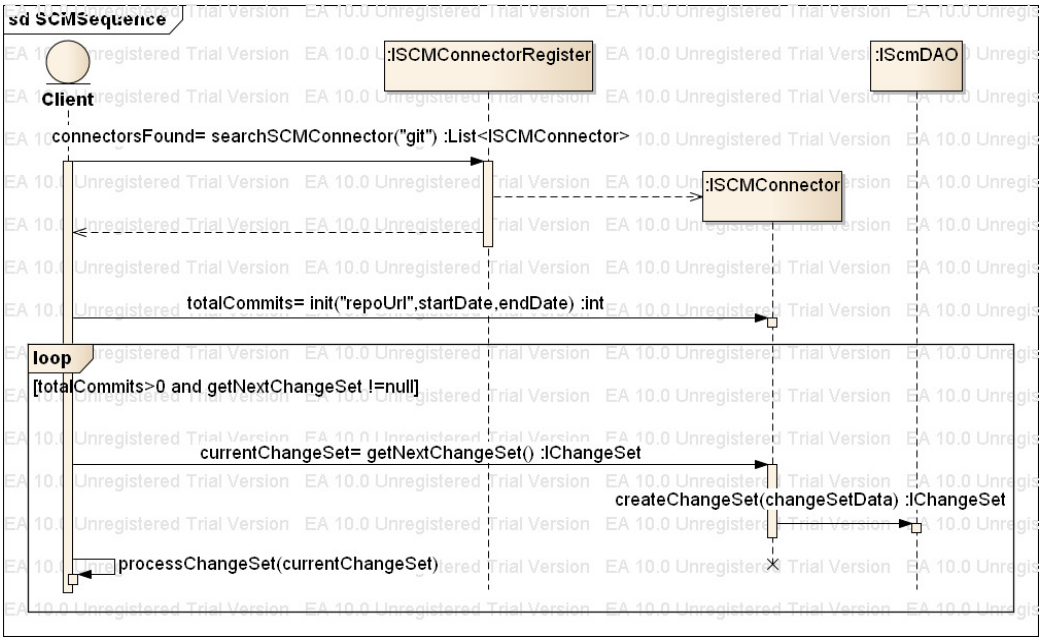

Figura 45 Diagrama de Sequência da integração entre o cliente o Módulo Conector de Ferramenta de Controle de Versão

Conectores de Ferramenta de Controle de Versão Implementados

Nesta versão da plataforma foi implementado um módulo que fornece um conector para acesso a conjuntos de alterações e arquivos armazenados em sistemas de controle de versão do tipo Git. A biblioteca Eclipse JGit foi utilizada

na implementação deste conector. Abaixo apresentamos o diagrama de estrutura composta deste componente.

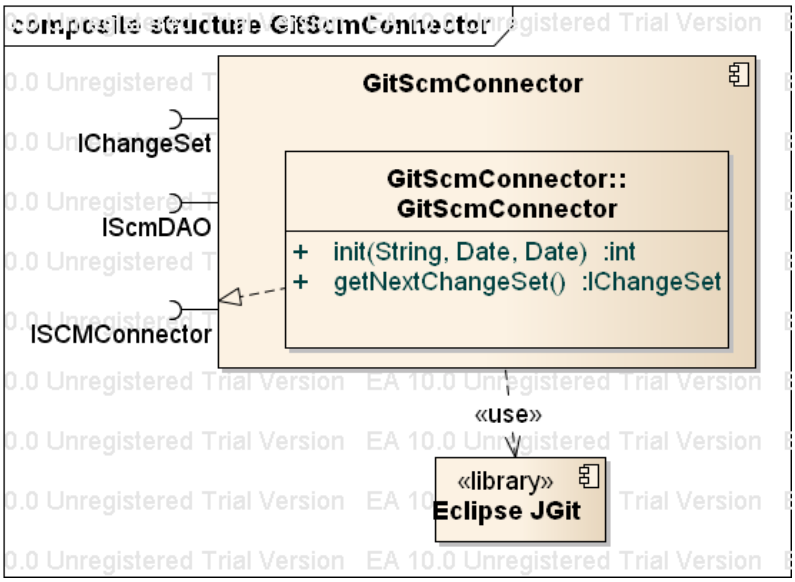

Figura 46 Diagrama de Estrutura Composta do Módulo Conector Git

Módulo Conector Comum de Ferramentas de Integração Contínua

Este módulo fornece acesso aos conectores existentes para ferramentas de integração contínua. Para isso, prove a interface *ICIConnector* (**Figura 47**) que é implementada pelos módulos conectores específicos de determinada ferramenta, além de também fornecer meios para recuperar e registrar novos conectores de ferramentas de integração contínua. Isto garante transparência para os outros componentes da plataforma que necessitam acessar informações destes tipos de repositórios de software, não sendo necessário depender de bibliotecas ou APIs específicas de cada ferramenta.

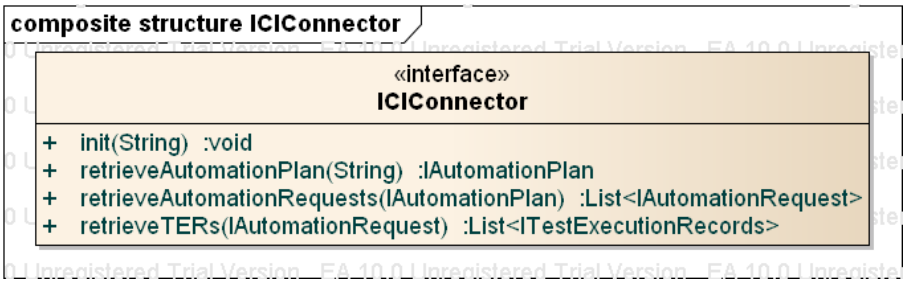

Figura 47 Interface ICIConnector

A interface *ICIConnector* fornece quatro métodos:

init: Método responsável por inicializar a conexão com o repositório em questão. Recebe como parâmetro a URL do repositório.

retrieveAutomationPlan: Método utilizado para recuperar um AutomationPlan pelo seu identificador**.**

retrieveAutomationRequests: Método utilizado para recuperar o conjunto de AutomationRequests executados em um AutomationPlan**.**

retrieveTERs: Método utilizado para recuperar o conjunto de testes executados no contexto de um AutomationRequest

Além desta interface, este módulo fornece mecanismos para o registro e recuperação de conectores. Isto é feito através da classe **CIConnectorCommon** e de sua interface **ICIConnectorCommon**.

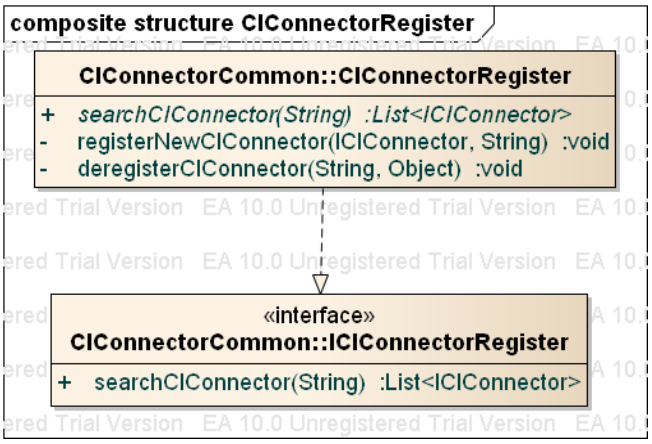

Figura 48 A classe CIConnectorRegister e a interface ICIConnectorRegister

A classe *CIConnectorRegister* funciona como um listener que recebe notificações de novos módulos conectores que fornecem classes que implementam a interface *ICMConnector* ou notificações de módulos que foram removidos.

Utiliza o método registerNewCIConnector para o primeiro caso e o método deregisterCIConnector para o segundo.

Já a interface *ICIConnectorRegister* implementada pela classe *CIConnectorRegister* fornece o método searchCIConnector para que os clientes externos possam encontrar um conector específico de ferramenta de integração contínua registrado na plataforma. Abaixo apresentamos o diagrama UML de estrutura composta deste módulo (**Figura 49**) e um diagrama UML de sequência (**Figura 50**) que demonstra um exemplo de interação entre um cliente, o registro de conectores e um conector específico.

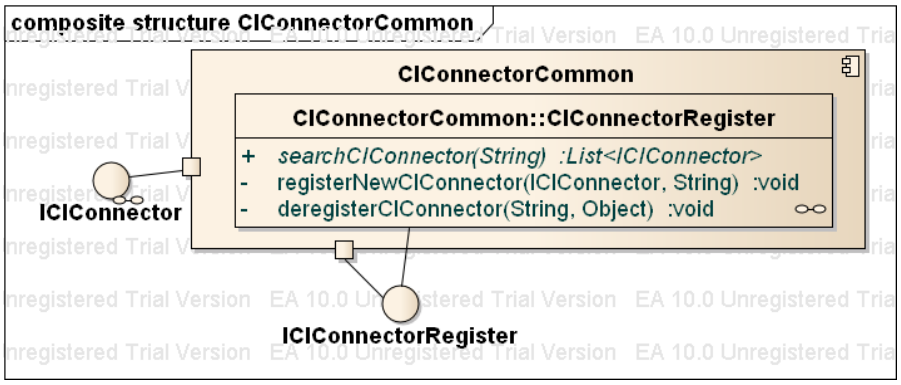

Figura 49 Diagrama de Estrutura Composta do Módulo Connector Comum de Ferramentas de Integração Contínua

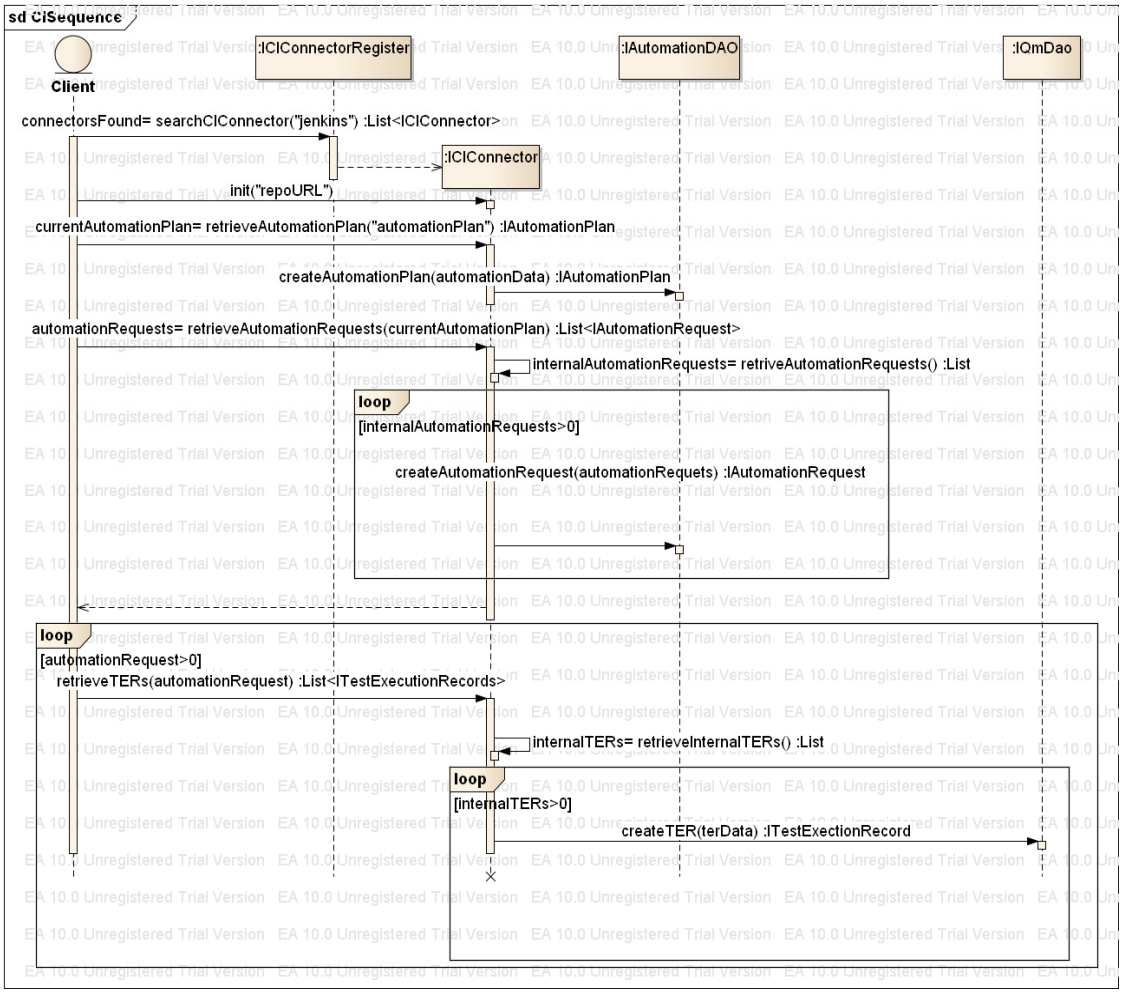

Figura 50 Diagrama de Sequência da integração entre o cliente o Módulo Connector de Ferramentas de Integração Contínua

Conectores de Ferramentas de Integração Contínua Implementados

Nesta versão da plataforma foi implementado um módulo que fornece um conector para acesso as informações armazenadas em ferramentas de integração contínua do tipo Jenkins. Utiliza a API Jenkins Rest para acessar este tipo de repositório. Abaixo apresentamos o diagrama UML de estrutura composta deste componente.

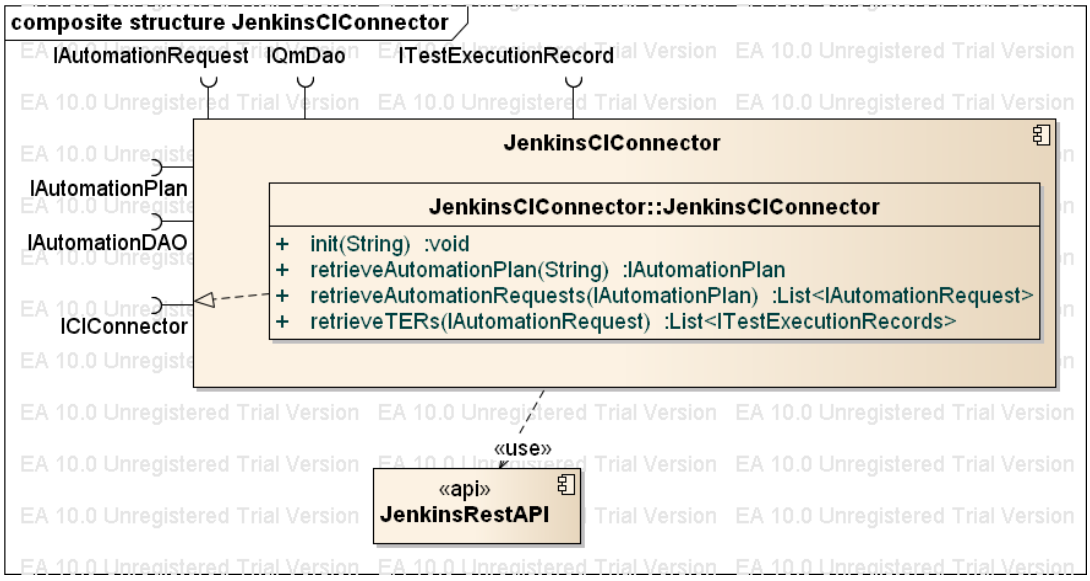

Figura 51 Diagrama de Estrutura Composta do Módulo Conector Jenkins

Módulo Conector Comum de Gerenciadores de Artefatos

Este módulo fornece acesso aos conectores existentes para ferramentas de gerenciamento de artefatos. Para isso, prove a interface *IAssetConnector* (**Figura 52**) que é implementada pelos módulos conectores específicos de determinada ferramenta, além de também fornecer meios para recuperar e registrar novos conectores. Isto garante transparência para os outros componentes da plataforma que necessitam acessar informações destes tipos de repositórios de software, não sendo necessário depender de bibliotecas ou APIs específicas de cada ferramenta.

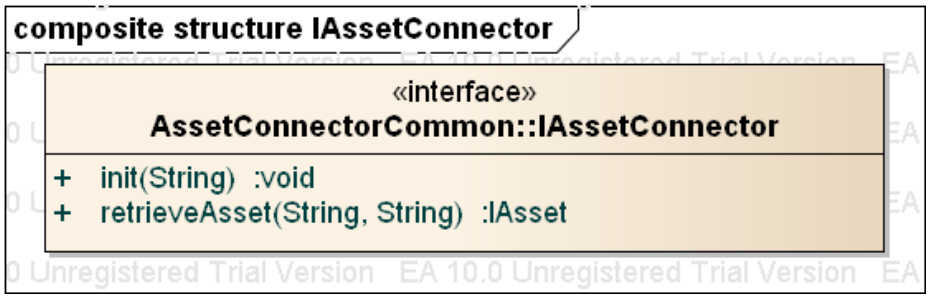

Figura 52 Interface IAssetConnector

A interface *IAssetConnector* fornece dois métodos:

init: Método responsável por inicializar a conexão com o repositório em questão. Recebe como parâmetro a URL do repositório.

retrieveAsset: Método utilizado para recuperar um artefato**.**

Além desta interface, este módulo fornece mecanismos para o registro e recuperação de conectores. Isto é feito através da classe **AssetConnectorCommon** e de sua interface **IAssetConnectorCommon**.

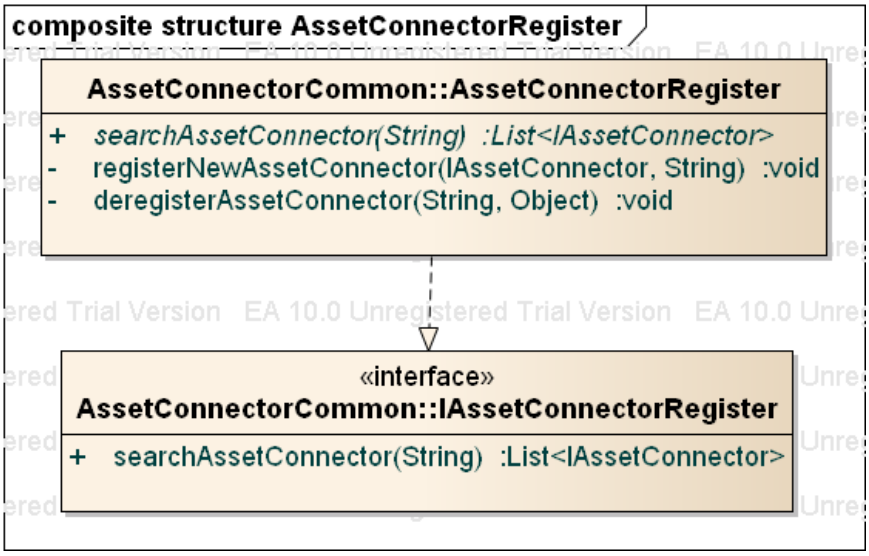

Figura 53 A classe AssetConnectorRegister e a interface IAssetConnectorRegister

A classe Asset*ConnectorRegister* funciona como um listener que recebe notificações de novos módulos conectores que fornecem classes que implementam a interface *IAssetConnector* ou notificações de módulos que foram removidos. Utiliza o método registerNewAssetConnector para o primeiro caso e o método deregisterAssetConnector para o segundo.

Já a interface *IAssetConnectorRegister* implementada pela classe *AssetConnectorRegister* fornece o método searchAssetConnector para que os clientes externos possam encontrar um conector específico de repositório de artefatos registrado na plataforma. Abaixo apresentamos o diagrama UML de

estrutura composta deste módulo (**Figura 54**) e um diagrama UML de sequência (**Figura 55**) que demonstra um exemplo de interação entre um cliente, o registro de conectores e um conector específico.

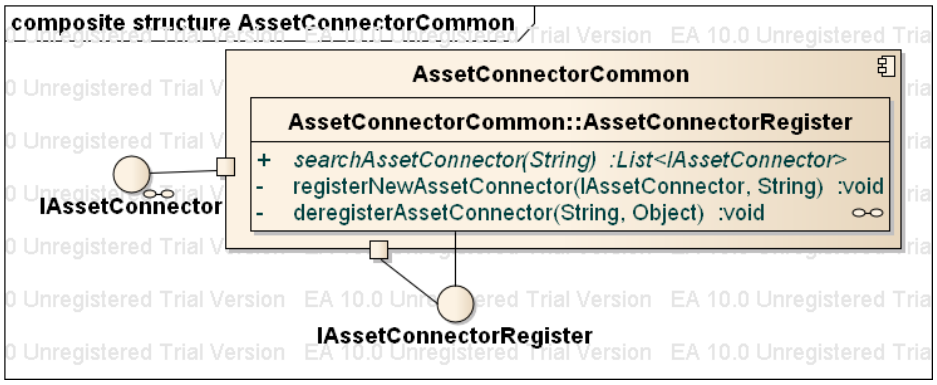

Figura 54 Diagrama de Estrutura Composta do Módulo Connector de Ferramentas de Gerenciamento de Artefatos

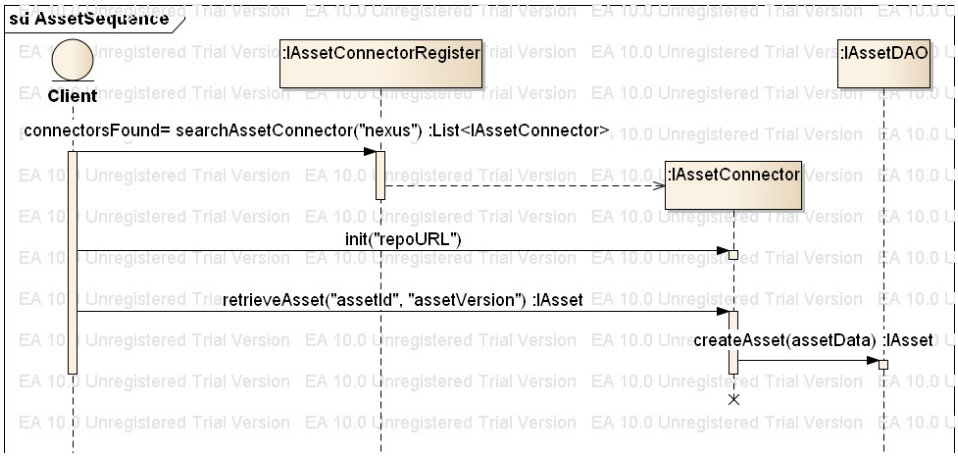

Figura 55 Diagrama de Sequência da integração entre o cliente o Módulo Connector de Ferramentas de Gerenciamento de Artefatos

Conectores de Ferramentas de Gerenciamento de Artefatos Implementados

Nesta versão da plataforma foi implementado um módulo que fornece um conector para acesso as informações armazenadas repositórios de artefatos do tipo Nexus. Utiliza a biblioteca Eclipse Aether para acessar este tipo de repositório. Abaixo apresentamos o diagrama de estrutura composta deste componente.

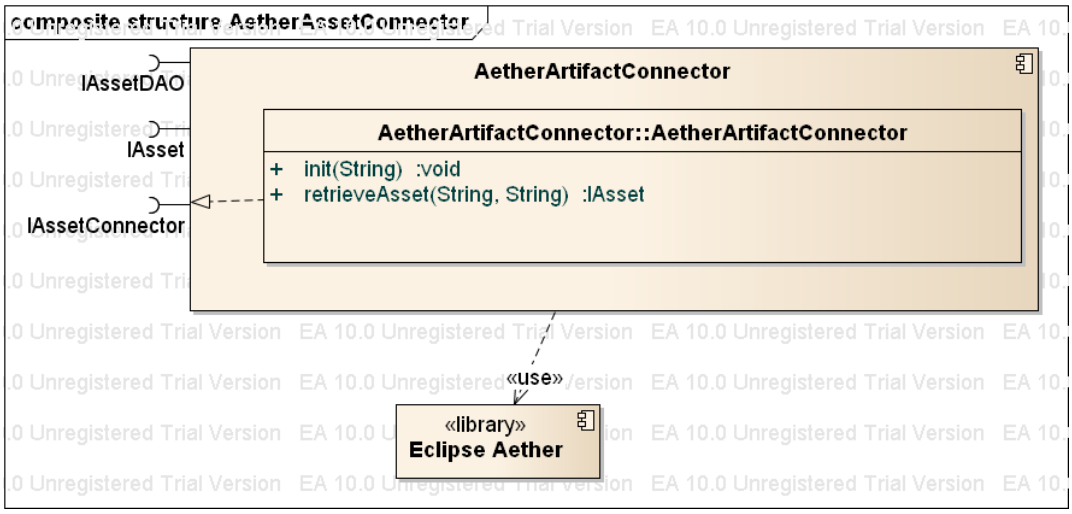

Figura 56 Diagrama de Estrutura Composta do Módulo Conector Aether

Módulo Conector Comum de Ferramentas de Gerenciamento de Defeitos e Demandas

Este módulo fornece acesso aos conectores existentes para ferramentas de gerenciamento de defeitos e demandas. Para isso, prove a interface *ICMConnector* (**Figura 57**) que é implementada pelos módulos conectores específicos de determinada ferramenta, além de também fornecer meios para recuperar e registrar novos conectores de ferramentas deste tipo. Isto garante transparência para os outros componentes da plataforma que necessitam acessar informações destes tipos de repositórios de software, não sendo necessário depender de bibliotecas ou APIs específicas de cada ferramenta.

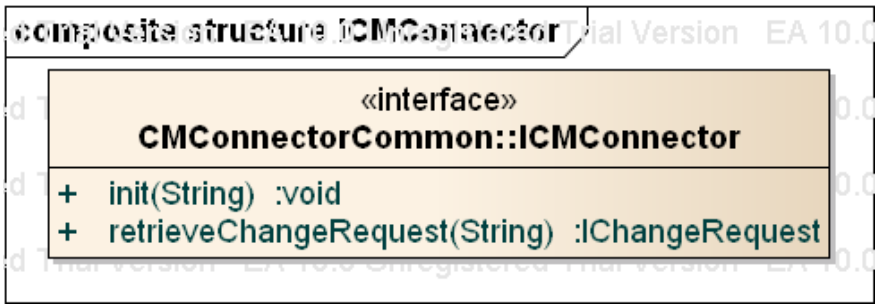

Figura 57 Interface ICMConnector

Esta interface fornece dois métodos:

init: Método responsável por inicializar a conexão com o repositório em questão. Recebe como parâmetro a URL do repositório.

retrieveChangeRequest: Método utilizado para recuperar uma demanda/defeito pelo seu identificador**.**

Além desta interface, este módulo fornece mecanismos para o registro e recuperação de conectores. Isto é feito através da classe **CMConnectorCommon** e de sua interface **ICMConnectorCommon**.

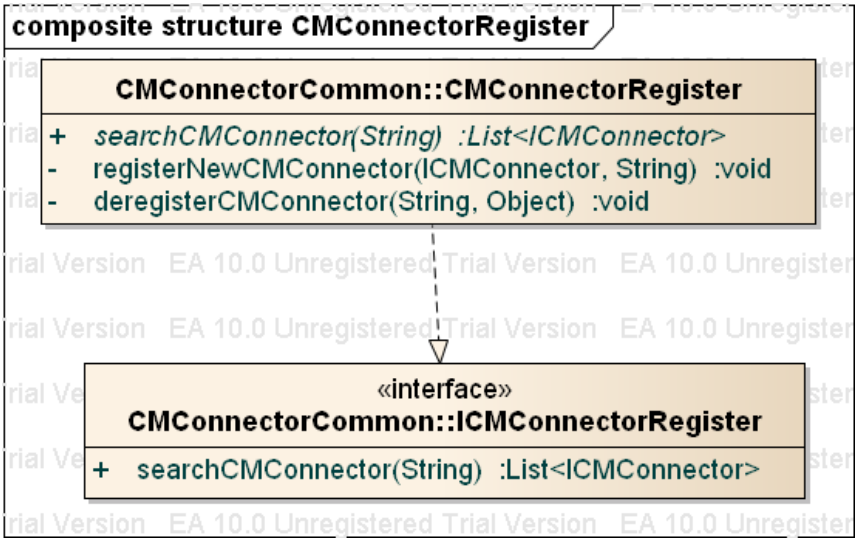

Figura 58 A classe *CMConnectorRegister* e a interface *ICMConnectorRegister*

A classe *CMConnectorRegister* funciona como um listener que recebe notificações de novos módulos conectores que fornecem classes que implementam a interface *ICMConnector* ou notificações de módulos conectores removidos. Utiliza o método registerNewCMConnector para o primeiro caso e o método deregisterCMConnector para o segundo.

Já a interface *ICMConnectorRegister* implementada pela classe *CMConnectorRegister* fornece o método searchCMConnector para que os clientes externos possam encontrar um conector de uma ferramenta específica registrado

na plataforma. Abaixo apresentamos o diagrama UML de estrutura composta deste módulo (**Figura 59**) e um diagrama UML de sequência (**Figura 60**) que demonstra um exemplo de interação entre um cliente, o registro de conectores e um conector específico.

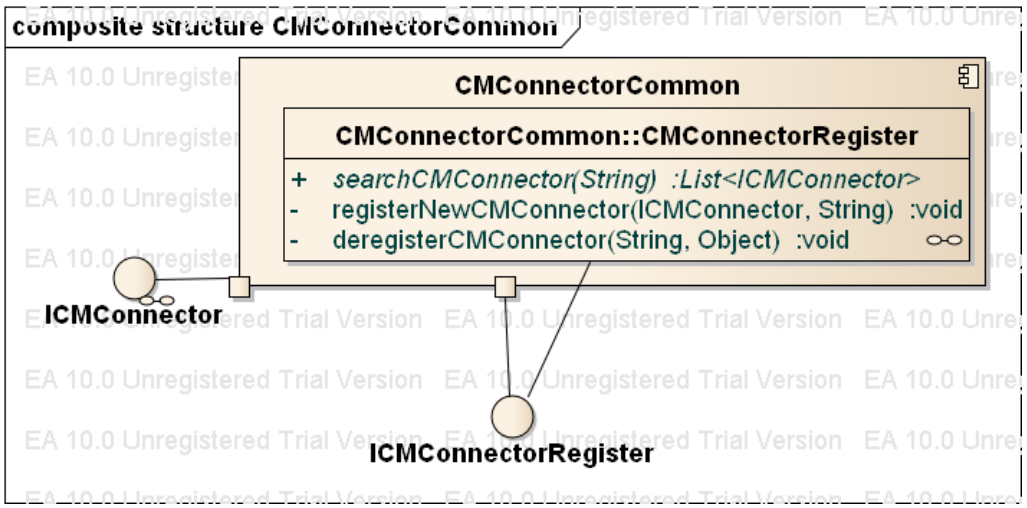

Figura 59 Diagrama de Estrutura Composta do Módulo Conector Comum de Ferramentas de Gerenciamento de Defeitos e Demandas.

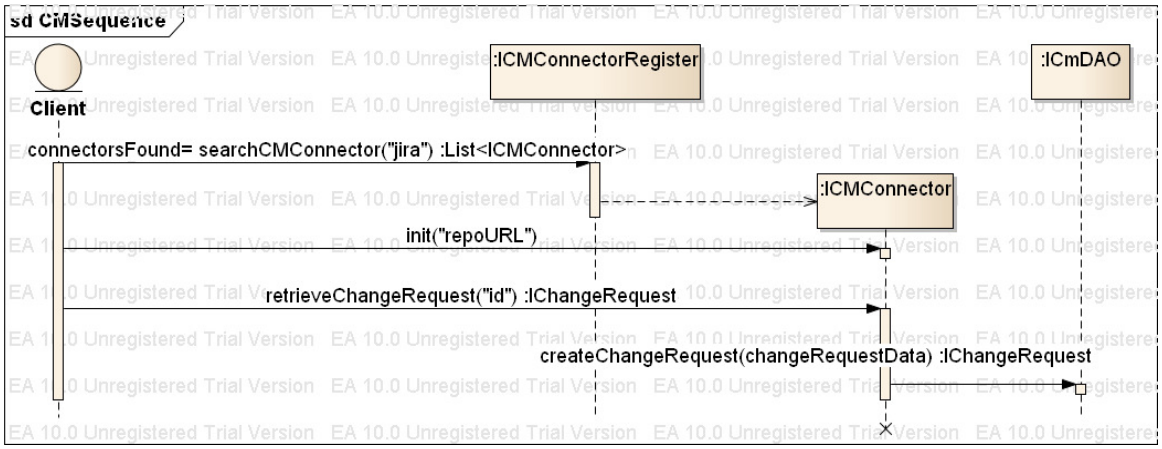

Figura 60 Diagrama de Sequência da integração entre o cliente o Módulo Conector de Ferramentas de Gerenciamento de Demandas e Defeitos

Conectores de Ferramentas de Gerenciamento de Defeitos e Demandas Implementados

Nesta versão da plataforma foi implementado um módulo que fornece um conector para acesso aos defeitos e demandas armazenados em ferramentas do tipo Atlassian Jira. Utiliza a biblioteca Jira Rest Client para acessar este tipo de repositório. Abaixo apresentamos o diagrama de estrutura composta deste componente.

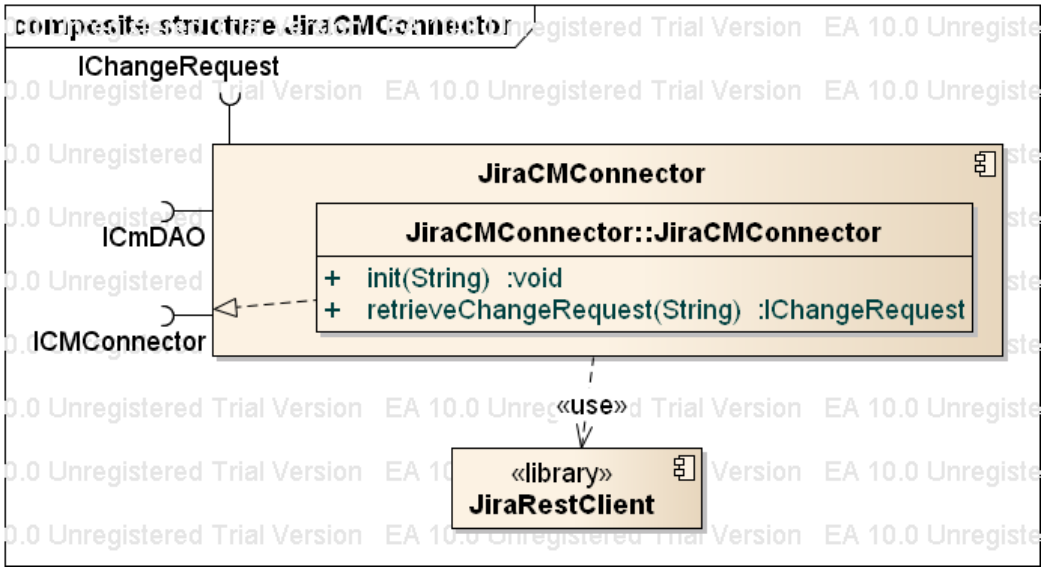

Figura 61 Diagrama de Estrutura Composta do Módulo Conector Jira

Módulo Conector Comum de Linguagens de Programação

Este módulo fornece acesso aos conectores existentes para informações da *Abstract Syntax Tree* de arquivos de código-fonte. Para isso, prove a interface *IASTConnector* (**Figura 62**) que é implementada pelos módulos conectores específicos de determinada linguagem de programação, além de também fornecer meios para recuperar e registrar novos conectores. Isto garante transparência para os outros componentes da plataforma que necessitam acessar informações de arquivos de código-fonte escritos em determinada linguagem, não sendo necessário depender de bibliotecas específicas de cada linguagem.

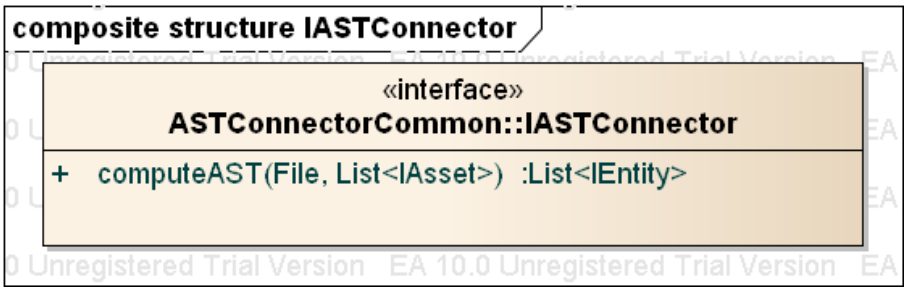

Figura 62 Interface IASTConnector

A interface *IASTConnector* fornece um único método:

computeAST: Método utilizado para computar a AST de um arquivo de código-fonte e representar as informações utilizando a Ontologia EPR**.** Recebe como parâmetros o arquivo e uma lista de ativos que compõe a dependência deste arquivo. Retorna a listagem de entidades encontradas pela computação da AST.

Além desta interface, este módulo fornece mecanismos para o registro e recuperação de conectores. Isto é feito através da classe **ASTConnectorCommon** e de sua interface **IASTConnectorCommon**.

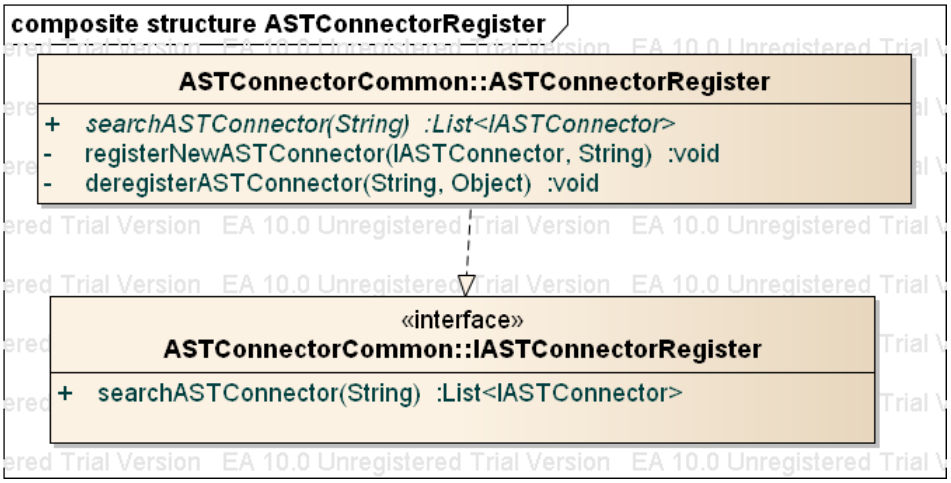

Figura 63 A classe *ASTConnectorRegister* e a interface *IASTConnectorRegister*

A classe AST*ConnectorRegister* funciona como um listener que recebe notificações de novos módulos conectores que fornecem classes que implementam a interface *IASTConnector* ou notificações de módulos que foram removidos.

Utiliza o método registerNewASTConnector para o primeiro caso e o método deregisterASTConnector para o segundo.

Já a interface *IASTConnectorRegister* implementada pela classe *ASTConnectorRegister* fornece o método searchASTConnector para que os clientes externos possam encontrar um conector específico de extração de AST registrado na plataforma. Abaixo apresentamos o diagrama UML de estrutura composta deste módulo (**Figura 64**) e um diagrama UML de sequência (**Figura 65**) que demonstra um exemplo de interação entre um cliente, o registro de conectores e um conector específico.

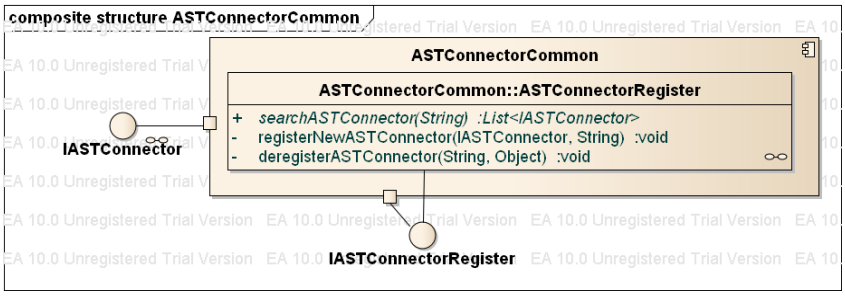

Figura 64 Diagrama de Estrutura Composta do Módulo Conector Comum de Linguagem de Programação

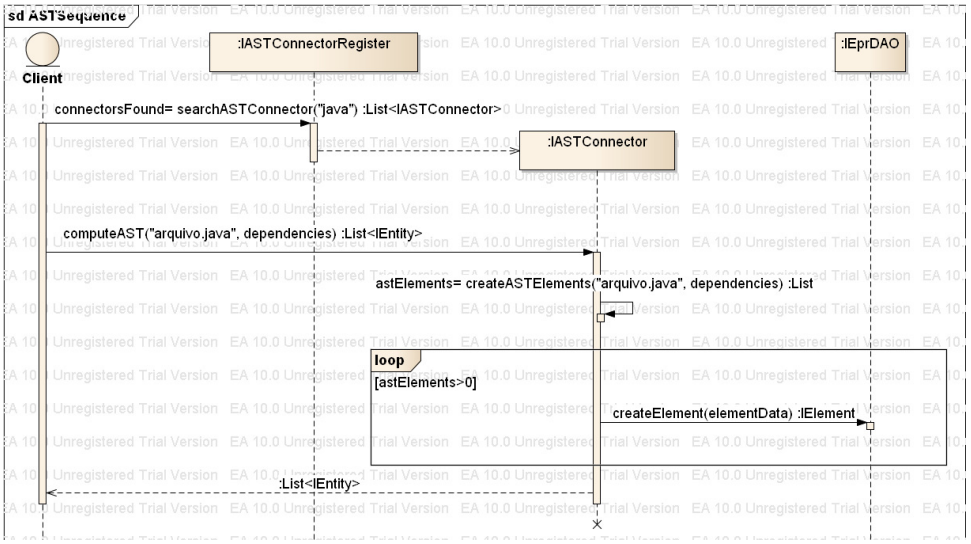

Figura 65 Diagrama de Sequência da integração entre o cliente o Módulo Conector de Linguagem de Programação

Conectores de Linguagem de Programação Implementados

Nesta versão da plataforma foi implementado um módulo que fornece um conector para acesso as informações de *Abstract Syntax Tree* definidas na linguagem de programação Java. Utiliza a biblioteca Eclipse JDT computar a AST. Abaixo apresentamos o diagrama de estrutura composta deste componente.

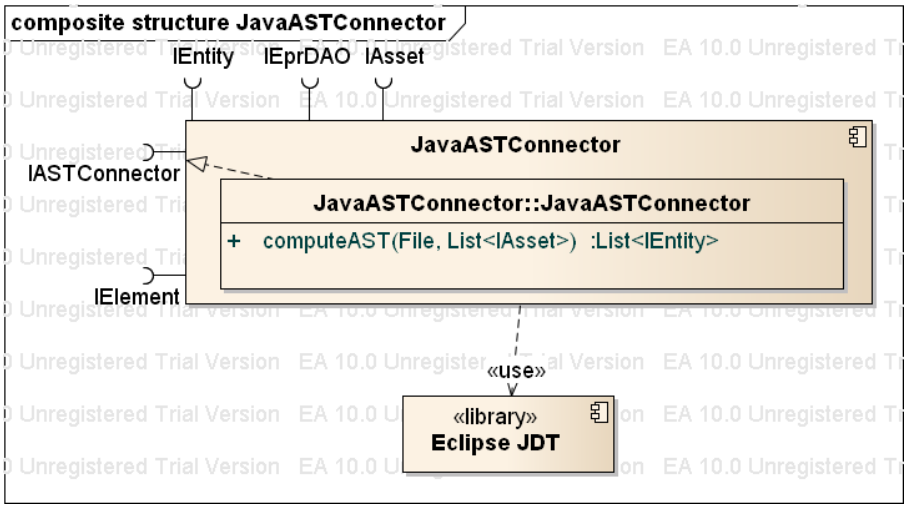

Figura 66 Diagrama de Estrutura Composta do Módulo Conector de Linguagem de Programação Java

Módulo Conector Comum de Ferramentas de Gerenciamento de Dependências

Este módulo fornece acesso aos conectores existentes para ferramentas de gerenciamento de dependências. Para isso, prove a interface *IDependencyConnector* (**Figura 67**) que é implementada pelos módulos conectores específicos de determinada gerenciador de dependências, além de também fornecer meios para recuperar e registrar novos conectores deste tipo. Isto garante transparência para os outros componentes da plataforma que necessitam acessar informações destes tipos de repositórios de software, não sendo necessário depender de bibliotecas ou APIs específicas de cada ferramenta.

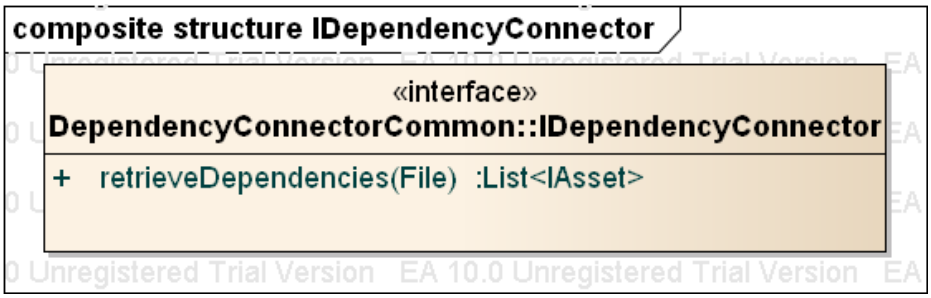

Figura 67 Interface IDependencyConnector

A interface *IDependencyConnector* fornece um único método:

retrieveDependencies: Método utilizado para as informações de dependências presentes em um arquivo**.**

Além desta interface, este módulo fornece mecanismos para o registro e recuperação de conectores. Isto é feito através da classe **DependencyConnectorCommon** e de sua interface **IDependencyConnectorCommon**.

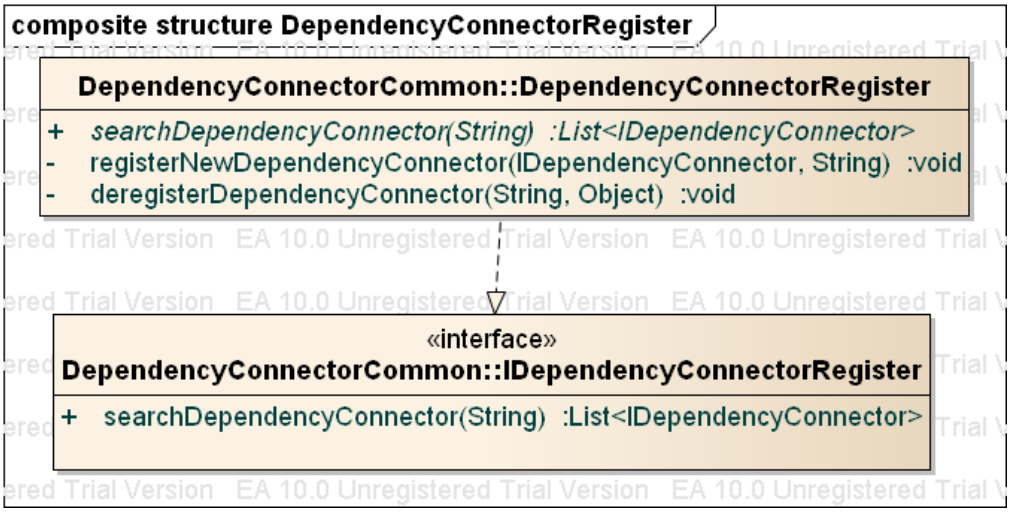

Figura 68 A classe DependencyConnectorRegister e a interface IDependencyConnectorRegister

A classe Dependency*ConnectorRegister* funciona como um listener que recebe notificações de novos módulos conectores que fornecem classes que implementam a interface *IDependencyConnector* ou notificações de módulos que foram removidos. Utiliza o método registerNewDependencyConnector para o primeiro caso e o método deregisterDependencyConnector para o segundo.

Já a interface *IDependencyConnectorRegister* implementada pela classe *DependencyConnectorRegister* fornece o método searchDependencyConnector para que os clientes externos possam encontrar um conector específico de repositório de dependências registrado na plataforma. Abaixo apresentamos o diagrama UML de estrutura composta deste módulo (**Figura 69**) e um diagrama UML de sequência (**Figura 70**) que demonstra um exemplo de interação entre um cliente, o registro de conectores e um conector específico.

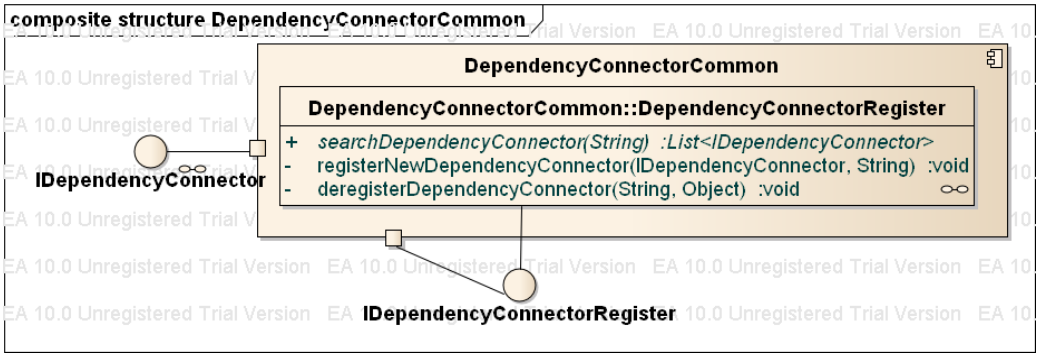

Figura 69 Diagrama de Estrutura Composta do Módulo Comum Conector de Ferramentas de Gerenciamento de Dependências

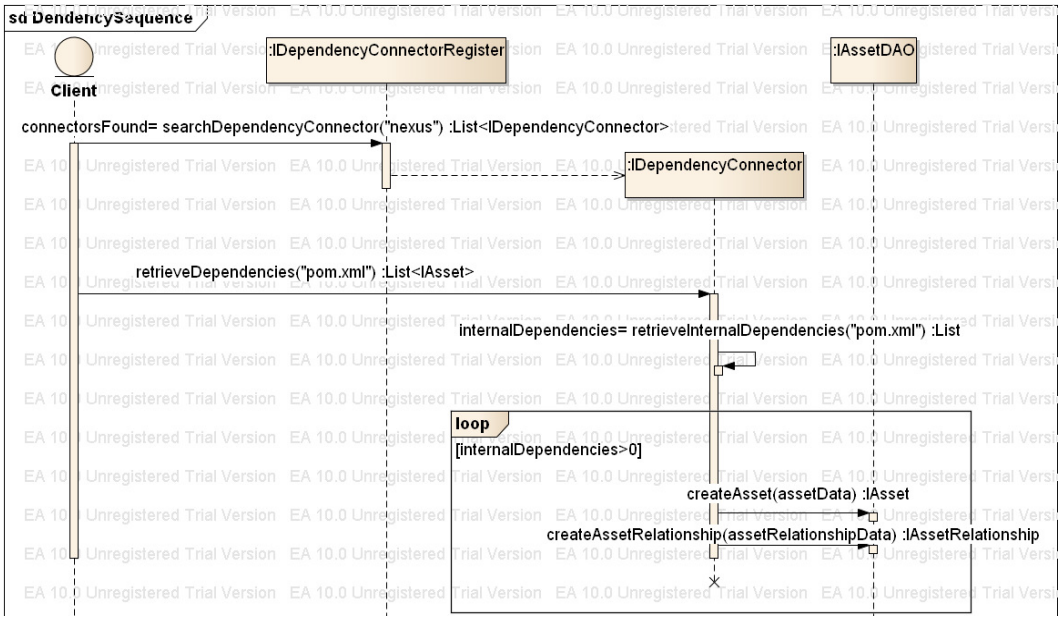

Figura 70 Diagrama de Sequência da integração entre o cliente o Módulo Conector de Ferramentas de Gerenciamento de Dependência

Conectores de Ferramentas de Gerenciamento de Dependências Implementados

Nesta versão da plataforma foi implementado um módulo que fornece um conector para acesso as informações de dependência presente em arquivos da ferramenta Maven. Utiliza a biblioteca Maven API para acessar este tipo de arquivo. Abaixo apresentamos o diagrama de estrutura composta deste componente.

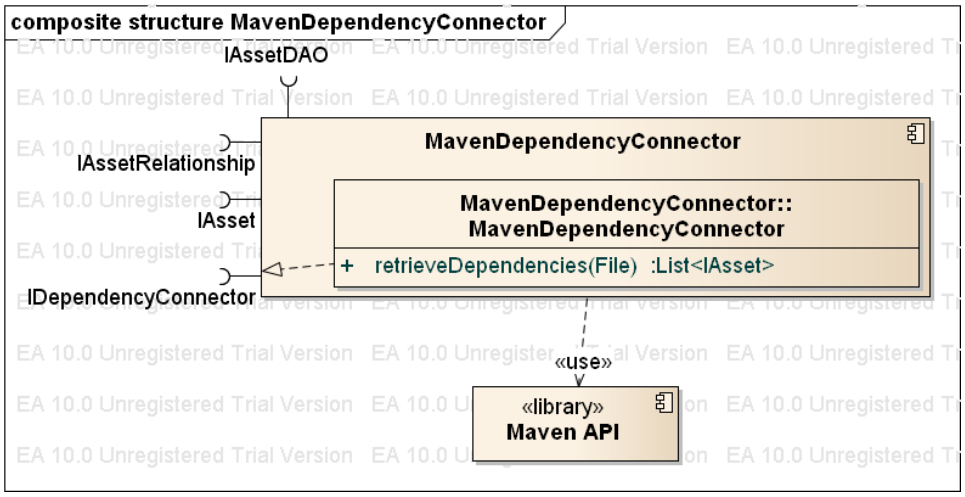

Figura 71 Diagrama de Estrutura Composta do Módulo Conector da Ferramenta Maven

Módulo Conector Comum de Licenças

Este módulo fornece acesso aos conectores existentes para recuperação de informações de licenças. Para isso, prove a interface *ILicenseConnector* (**Figura 72**) que é implementada pelos módulos conectores específicos de determinado arquivo que contém informações de licenças, além de também fornecer meios para recuperar e registrar novos conectores deste tipo. Isto garante transparência para os outros componentes da plataforma que necessitam acessar informações de licenças, não sendo necessário depender de bibliotecas ou APIs específicas.

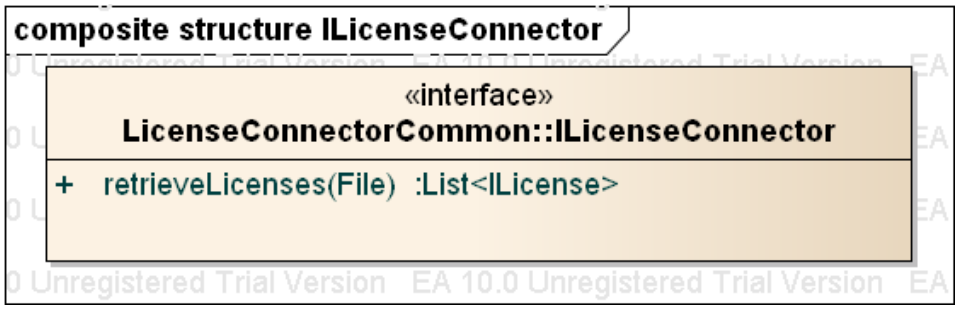

Figura 72 Interface ILicenseConnector

A interface *ILicenseConnector* fornece um único método:

retrieveLicenses: Método utilizado para as informações de licenças presentes em um arquivo**.**

Além desta interface, este módulo fornece mecanismos para o registro e recuperação de conectores. Isto é feito através da classe **LicenseConnectorCommon** e de sua interface **ILicenseConnectorCommon**.

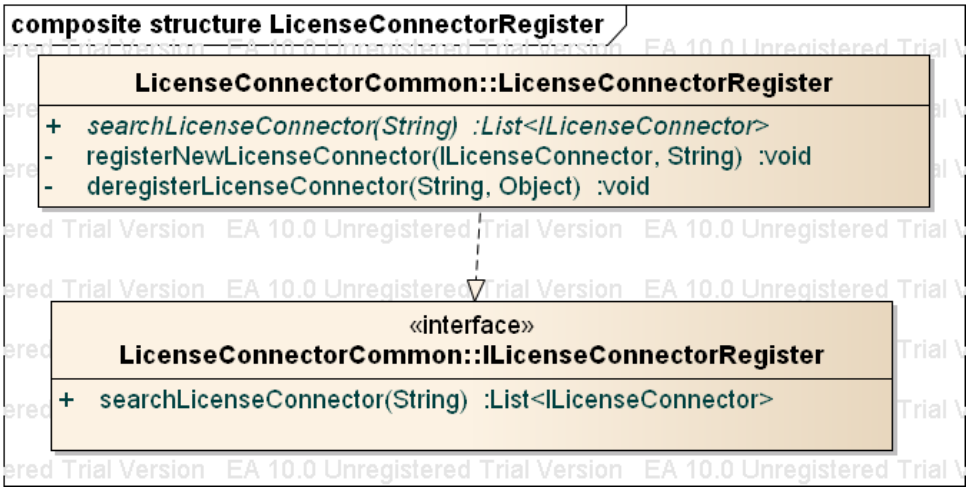

Figura 73 A classe LicenseConnectorRegister e a interface ILicenseConnectorRegister

A classe License*ConnectorRegister* funciona como um listener que recebe notificações de novos módulos conectores que fornecem classes que implementam a interface *ILicenseConnector* ou notificações de módulos que foram removidos. Utiliza o método registerNewLicenseConnector para o primeiro caso e o método deregisterLicenseConnector para o segundo.

Já a interface *ILicenseConnectorRegister* implementada pela classe *LicenseConnectorRegister* fornece o método searchLicenseConnector para que os clientes externos possam encontrar um conector específico de extração de licenças registrado na plataforma. Abaixo apresentamos o diagrama UML de estrutura composta deste módulo (**Figura 74**) e um diagrama UML de sequência (**Figura 75**) que demonstra um exemplo de interação entre um cliente, o registro de conectores e um conector específico.

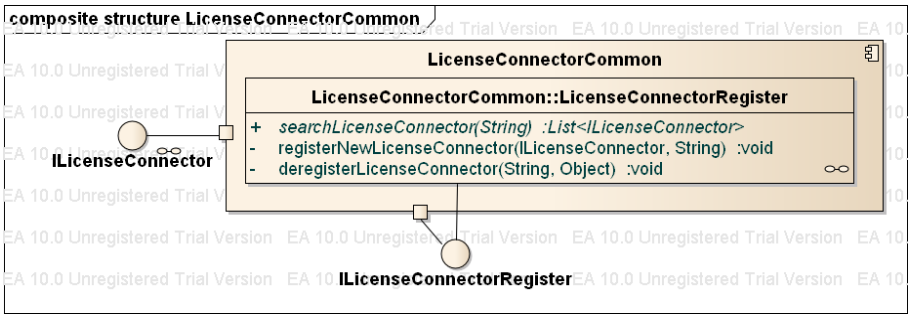

Figura 74 Diagrama de Estrutura Composta do Módulo Comum Connector de Licenças

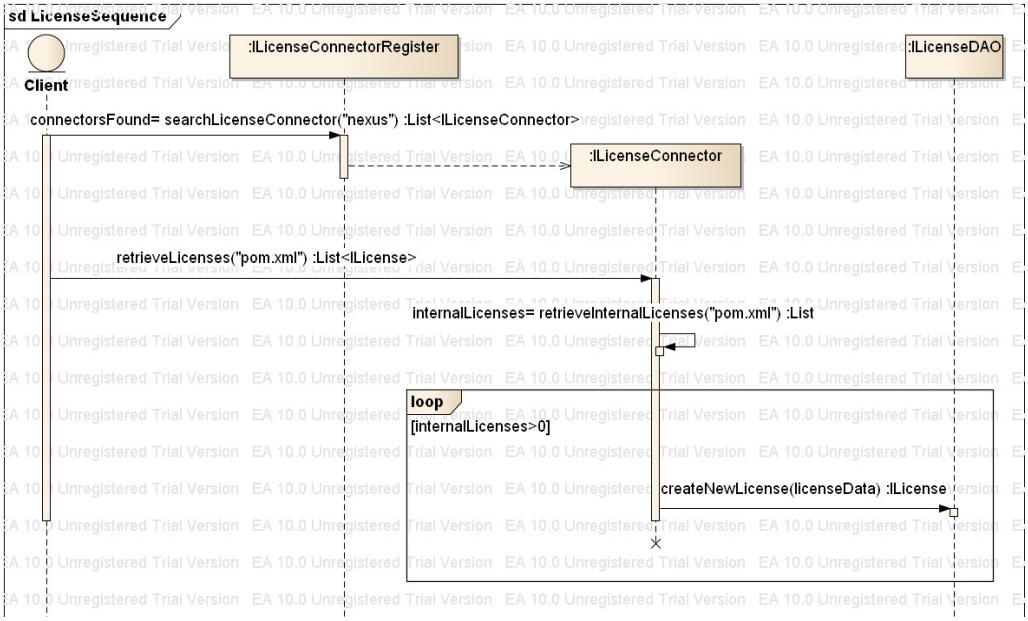

Figura 75 Diagrama de Sequência da integração entre o cliente o Módulo Conector de Licenças

Conectores de Licenças Implementados

Nesta versão da plataforma foi implementado um módulo que fornece um conector para acesso as informações de licenças presente em arquivos da ferramenta Maven. Utiliza a biblioteca Maven API para acessar este tipo de arquivo. Abaixo apresentamos o diagrama de estrutura composta deste componente.

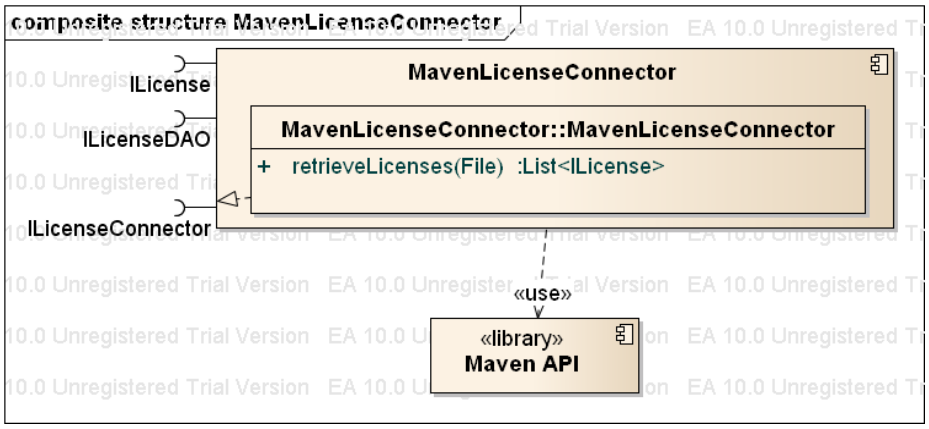

Figura 76 Diagrama de Estrutura Composta do Módulo Conector de Licenças Maven

Apêndice B – Módulos Transformadores RDF

Módulo Transformador RDF de Controle de Versão

Este módulo é responsável por converter para termos RDF do vocabulário OSLC SCM as informações recuperadas pelos diversos conectores de ferramentas de controle de versão. Além disso, fornece acesso a estas informações a outros módulos da plataforma através de diversas interfaces que representam os conceitos deste vocabulário. Abaixo apresentamos o diagrama UML de estrutura composta deste módulo (**Figura 77**), suas classes internas e as interfaces que utiliza e requer de outros módulos.

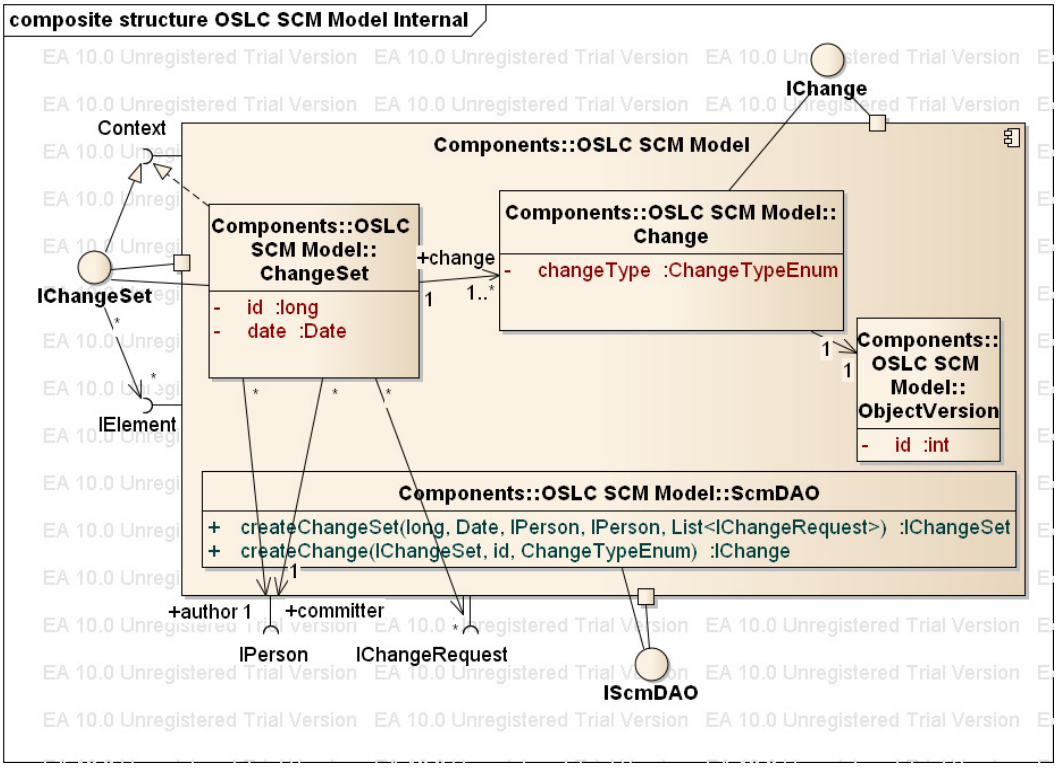

Figura 77 Diagrama de Estrutura Composta do Módulo Transformador RDF de Controle de Versão

Classes Internas:

ChangeSet: classe Java que representa um conjunto de alterações e tem as informações presentes na classe **ChangeSet** do vocabulário OSLC SCM. Podem estar associada a um ou mais objetos do tipo **Change**. É a realização da interface **IChangeSet**.

Change: classe Java que representa a modificação, inclusão ou remoção de um elemento do tipo **ObjectVersion**. Está associada a apenas um elemento **ChangeSet**.

ObjectVersion: classe Java que representa uma versão de um arquivo sob controle de versão.

ChangeTypeEnum: enumeração Java que representa os tipo de uma mudança. Exemplos destes tipos podem ser ADD, REMOVE e MODIFY.

ScmDAO: classe Java que implementa mecanismos para a recuperação e persistência dos dados RDF que estão sob controle deste módulo.

Interfaces que provê:

IChangeSet: interface Java que prove acesso externo as informações de um ChangeSet.

IChange: interface Java que prove acesso externo as informações de um Change.

IScmDAO: interface Java que prove acesso externo ao mecanismo de recuperação de informações armazenadas por este módulo.

Interfaces que requer:

IPerson: interface Java que representa tanto o autor como o *committer* de um conjunto de alterações (ChangeSet).

IChangeRequest: interface Java que é utilizada para representar as tarefas envolvidas em um modificação no controle de versão.

IElement: interface Java que representa um elemento de código-fonte conforme definido pelo módulo Transformador RDF de Código Fonte. Um conjunto de alterações pode conter um conjunto de elementos que podem ter sofrido alterações em relação a sua versão anterior.

Context: interface Java que representa um contexto conforme definido pelo módulo Transformador RDF de Impactos. Esta interface é implementada pela classe ChangeSet e permite que seja registrada a diferença estrutural de uma entidade de código-fonte modificada em dois conjuntos de alterações.

Módulo Transformador RDF de Licença

Este módulo é responsável por converter para termos RDF do vocabulário CC REL as informações recuperadas pelos diversos conectores de licença. Além disso, fornece acesso a estas informações a outros módulos da plataforma através de diversas interfaces que representam os conceitos deste vocabulário. Abaixo apresentamos o diagrama UML de estrutura composta deste módulo (**Figura 78**), suas classes internas e as interfaces que utiliza e requer de outros módulos.

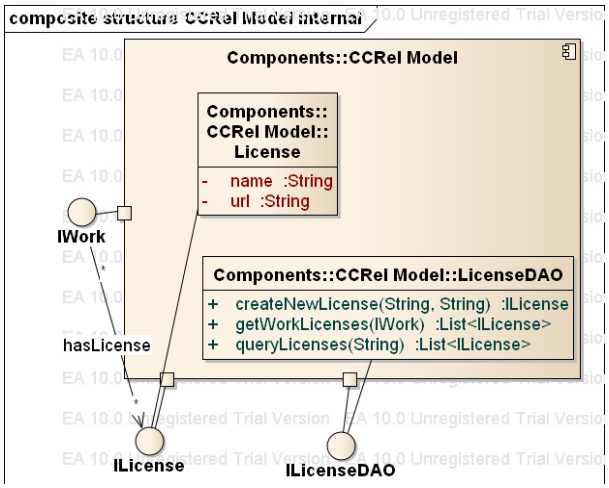

Figura 78 Diagrama de Estrutura Composta do Módulo Transformador RDF de Licença.

Classes Internas:

License: classe Java que representa uma licença de utilização. Define atributos como o nome da licença e seu endereço.

LicenseDAO: classe Java que fornece mecanismos para a recuperação e persistência de elementos definidos por este módulo.

Interfaces que provê:

ILicense: interface provida para que outros módulos possam acessar as informações de uma licença.

ILicenseDAO: interface provida para que outros módulos possam executar métodos relacionados a recuperação, criação e modificação de elementos definidos neste módulo.

IWork: interface que representa algo licenciável e é provida por este módulo para que outros possam implementá-la a fim de se associarem a uma ou mais licenças.

Módulo Transformador RDF de Artefatos

Este módulo é responsável por converter para termos RDF do vocabulário OSLC Asset Management as informações recuperadas pelos diversos conectores de artefatos. Além disso, fornece acesso a estas informações a outros módulos da plataforma através de diversas interfaces que representam os conceitos deste vocabulário. Abaixo apresentamos o diagrama UML de estrutura composta deste módulo (**Figura 79**), suas classes internas e as interfaces que utiliza e requer de outros módulos.

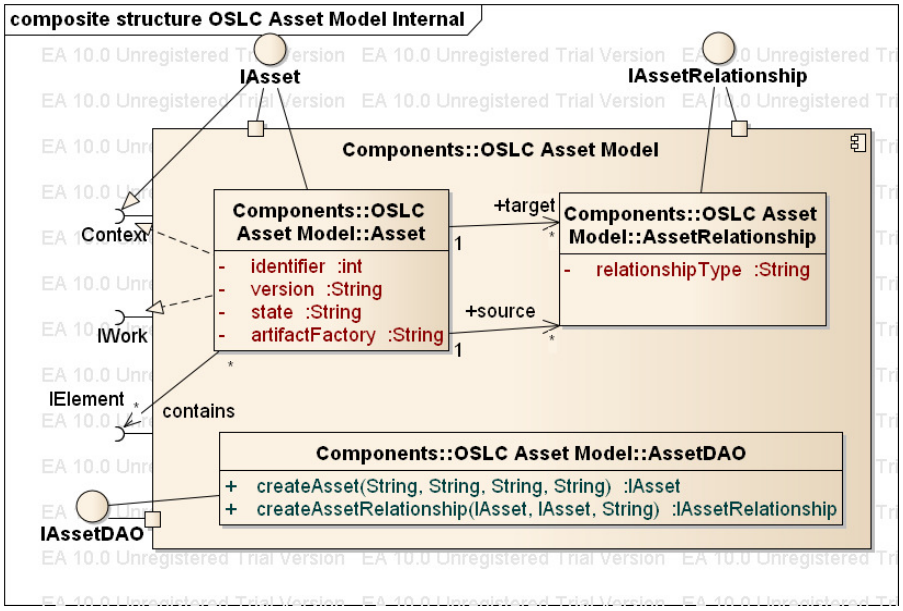

Figura 79 Diagrama de Estrutura Composta do Módulo Transformador RDF de Artefatos

Classes Internas:

Asset: classe Java que representa um artefato de uma biblioteca. Define atributos como versão, estado e identificador.

AssetRelationship: classe que representa um relacionamento entre dois artefatos.

AssetDAO: classe que fornece mecanismo para a recuperação e persistência de elementos definidos por este módulo.

Interfaces que provê:

IAsset: interface que fornece acesso externo as informações de um artefato.

IAssetRelationship: interface que fornece acesso externo as informações de um relacionamento entre dois artefatos.

IAssetDAO: interface que prove acesso externo ao mecanismo de persistência definido neste módulo.

Interfaces que requer:

Context: interface que representa um contexto conforme definido pelo módulo Transformador RDF de Impactos. Esta interface é implementada por um Asset e permite que seja registrada a diferença entre duas versões de uma biblioteca.

IWork: interface que representa trabalho licenciável conforme definido pelo módulo de modelo CCRel. Esta interface é implementada por um Asset e permite que sejam registradas licenças associadas a um Asset.

IElement: interface que representa um elemento de código-fonte conforme definido pelo módulo Transformador RDF de Código-Fonte. Um artefato pode conter um conjunto de elementos.

Módulo Transformador RDF de Código-Fonte

Este módulo é responsável por converter para termos RDF da Ontologia EPR as informações recuperadas pelos diversos conectores de linguagem de programação. Além disso, fornece acesso a estas informações a outros módulos da plataforma através de diversas interfaces que representam os conceitos deste vocabulário. Abaixo apresentamos o diagrama UML de estrutura composta deste módulo (**Figura 80**), suas classes internas e as interfaces que utiliza e requer de outros módulos.

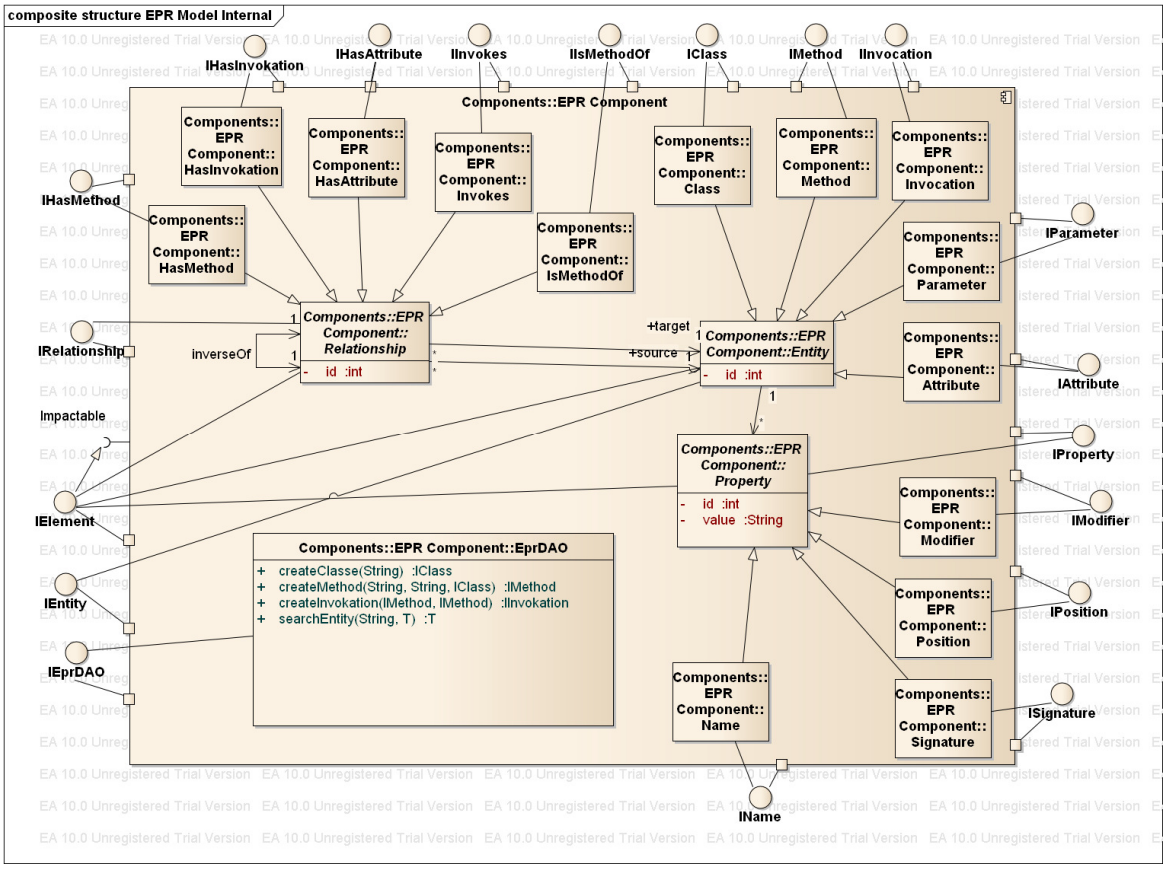

Figura 80 Diagrama de Estrutura Composta do Módulo Transformador RDF de Código-Fonte

Principais Classes Internas:

Entity: classe Java abstrata que é a generalização de todos os tipos de entidades de código-fonte representadas na plataforma, como Método, Classe, Atributo, Parâmetro e Invocação. Define um único atributo que é o identificado da entidade.

Property: classe Java abstrata que é a generalização das propriedades de entidades representadas na plataforma, como Signature, Name, Modifier e etc. Define o identificador da propriedade e o valor da propriedade, como o valor do nome de uma classe.

Relationship: classe Java abstrata que é a generalização de todos os tipos de relacionamento entre entidades de código-fonte representadas na plataforma,

como HasMethod, HasReturnType, IsAttributeOf e etc. Define o identificador do relacionamento e qual é o relacionamento inverso do relacionamento corrente, conforme definido na ontologia.

EprDAO: classe Java que fornece mecanismo para a recuperação e persistência de elementos definidos por este módulo

Principais Interfaces que provê:

IElement: interface que fornece acesso externo a elementos do tipo Entity, Relationship e Property. É utilizada nos módulos Transformador RDF de Artefatos e de Controle de Versão para representar os elementos de código-fonte que fazem parte de uma versão de biblioteca ou de um conjunto de alterações.

IEntity: interface que fornece acesso externo as entidade de código-fonte como Classes e Métodos.

IRelationship: interface que fornece acesso externo as entidade do tipo Relationship.

IProperty: interface que fornece acesso externo as entidade do tipo Property.

IEprDAO: interface provida para que outros módulos possam executar métodos relacionados a recuperação, criação e modificação de elementos definidos neste módulo.

Interfaces que requer:

Impactable: interface que representa algo que pode ter sido afeto por uma mudança. No contexto da plataforma, pode representar uma entidade, propriedade ou relacionamento que sofreu mudanças de um conjunto de alterações para outro ou de uma versão de biblioteca para outra. Esta interface é fornecida pelo módulo Transformador RDF de Impactos.

Módulo Transformador RDF de Impacto

Este módulo é responsável converter as informações computadas das diferenças entre duas versões de elementos de código-fonte representados com a Ontologia EPR. A conversão é feita utilizando os termos RDF da Ontologia de Impactos. Além disso, fornece acesso a estas informações a outros módulos da plataforma através de diversas interfaces que representam os conceitos deste vocabulário. Abaixo apresentamos o diagrama UML de estrutura composta deste módulo (**Figura 81**), suas classes internas e as interfaces que utiliza e requer de outros módulos.

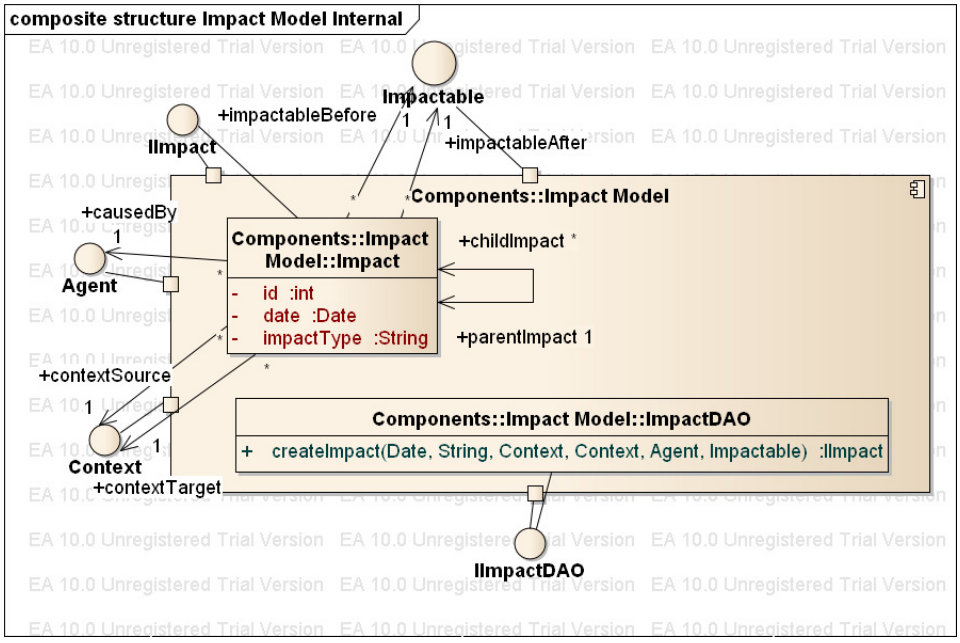

Figura 81 Diagrama de Estrutura Composta do Módulo Transformador RDF de Impactos

Classes Internas:

Impact: classe Java que representa um Impacto conforme definido na ontologia de Impactos. Esta classe tem o objetivo de representar uma diferença ocorrida em um impactado em um determinando contexto e causada por um determinado Agente. Define atributos como o identificador do Impacto, a data da ocorrência e

o tipo de Impacto: ADD, MODIFY, DELETE. Além disso, é possível estruturar uma hierarquia de impactos conforme dito na discussão sobre esta ontologia.

ImpactDAO: classe Java que fornece mecanismo para a recuperação e persistência de impactos.

Interfaces que provê:

IImpacto: interface Java que fornece acesso externo a um impacto. Fornece acesso a seus principais atributos.

Impactable: interface Java que representa algo que pode ser impactado, ou seja, passível de sofrer alterações em sua estrutura, como classes, métodos, atributos, seus relacionamentos e propriedades.

Context: interface Java que representa o contexto de origem e de destino de um impacto, ou seja, em que circunstâncias um determinando impacto foi computado. Atualmente objetos deste tipo podem ser ChangeSets ou Assets, representando os conjuntos de alterações ou versões de bibliotecas que modificaram alguma entidade de código-fonte.

Agent: interface Java que representa o originador de um impacto, ou seja, a entidade responsável por impactar algo entre dois contextos. Atualmente somente a classe Person implementa tal interface.

IImpactDAO: interface Java provida para que outros módulos possam executar métodos relacionados a recuperação, criação e modificação de impactos.

Módulo Transformador RDF de Integração Contínua

Este módulo é responsável por converter para termos RDF do vocabulário OSLC Automation as informações das ferramentas de integração contínua recuperadas pelos diversos conectores da plataforma. Além disso, fornece acesso a estas informações a outros módulos da plataforma através de diversas interfaces

que representam os conceitos deste vocabulário. Abaixo apresentamos o diagrama UML de estrutura composta deste módulo (**Figura 82**), suas classes internas e as interfaces que utiliza e requer de outros módulos.

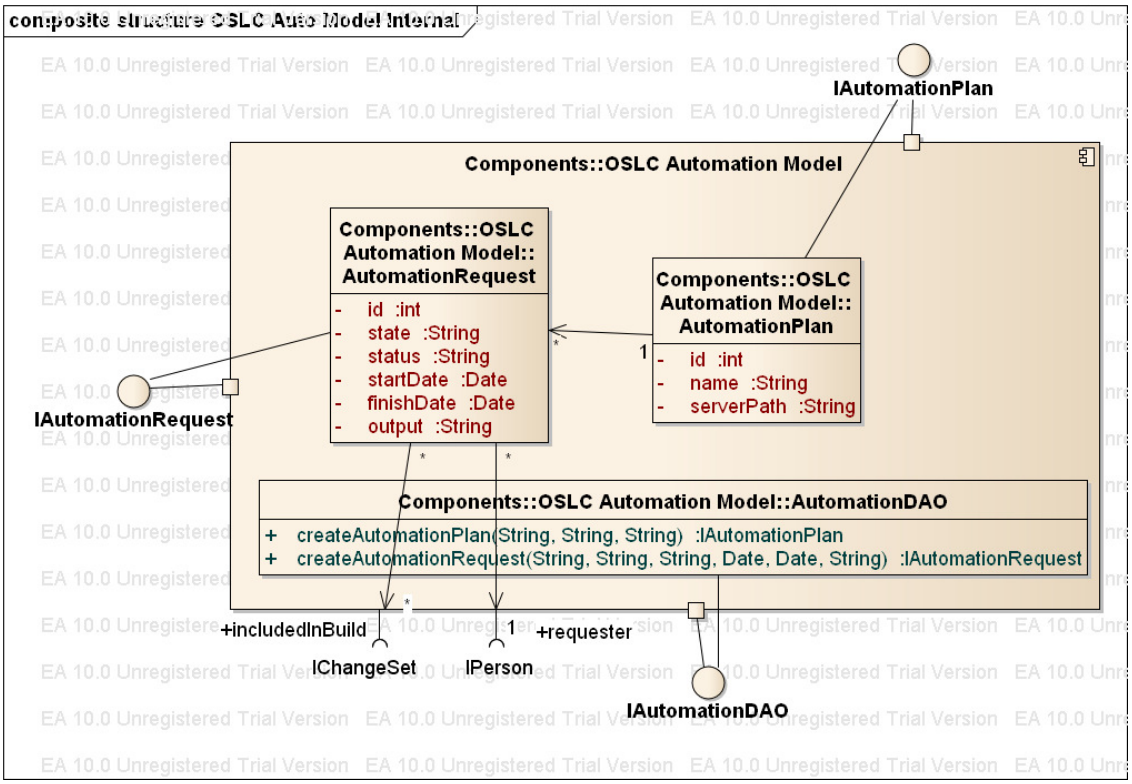

Figura 82 Diagrama de Estrutura Composta do Módulo Transformador RDF de Integração Contínua

Classes Internas:

AutomationPlan: classe Java que representa uma definição de planto de automação, que é conhecido nas ferramentas de integração contínua como Job ou Build.

AutomationRequest: classe Java que representa um pedido de execução de um plano de automação. Define atributos, como estado, data de início da requisição, data de fim da requisição e saída da execução. Pode estar associado a um ou mais conjuntos de alterações e a uma pessoa que representa o solicitante.

AutomationDAO: classe Java que fornece mecanismo para a recuperação e persistência de elementos definidos por este módulo.

Interfaces que provê:

IAutomationRequest: interface Java provida para que outros módulos possam acessar as informações de um AutomationRequest.

IAutomationPlan: interface provida para que outros módulos possam acessar as informações de um AutomationPlan.

IAutomationDAO: interface Java que prove acesso ao mecanismo de persistência definido neste módulo.

Interfaces que requer:

IChangeSet: interface Java que representa um determinado conjunto de alterações que foi incluído no AutomationRequest.

IPerson: interface que é utilizada para representar o solicitante.

Módulo Transformador RDF de Demandas/Defeitos

Este módulo é responsável por converter para termos RDF do vocabulário OSLC Change Management as informações das ferramentas de gerenciamento de defeitos e demandas recuperadas pelos diversos conectores da plataforma. Além disso, fornece acesso a estas informações a outros módulos da plataforma através de diversas interfaces que representam os conceitos deste vocabulário. Abaixo apresentamos o diagrama UML de estrutura composta deste módulo (**Figura 83**), suas classes internas e as interfaces que utiliza e requer de outros módulos.

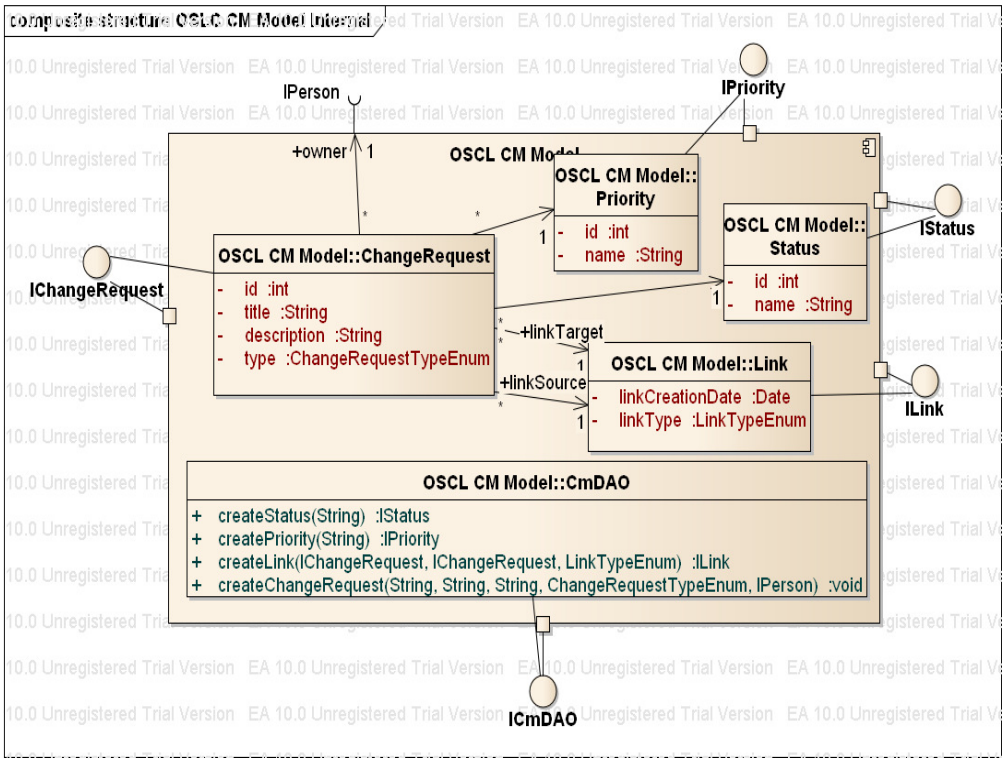

Figura 83 Diagrama de Estrutura Composta do Módulo Transformador RDF de Demandas/Defeitos

Classes Internas:

ChangeRequest: classe Java que representa um defeito ou demanda conforme definido no vocabulário OSLC CM. É a realização da interface IChangeRequest.

Status: classe que representa os possíveis estados que um(a) defeito/demanda pode estar.

Priority: classe que representar os valores de prioridade que um(a) defeito/demanda pode ter.

ChangeRequestTypeEnum: enumeração que representa os tipo um(a) defeito/demanda existentes. Exemplos destes tipos podem ser Defeito, Tarefa ou Atividade.

Link: classe Java que representa a associação entre dois ChangeRequests. Possui uma data de criação e um tipo.

LinkTypeEnum: enumeração Java que representa o tipo de um relacionamento. Alguns valores válidos são Pai-Filho, Bloqueia e Relacionado.

CmDAO: classe Java que fornece mecanismo para a recuperação e persistência de elementos definidos por este módulo

Interfaces que provê:

IChangeRequest: interface Java que prove acesso externo as informações de um ChangeRequest.

IPriority: interface Java que prove acesso externo as informações de um tipo de prioridade.

IStatus: interface Java que prove acesso externo as informações de um tipo de status.

ICmDAO: interface Java que fornece acesso ao mecanismo de persistência definido neste módulo.

Interfaces que requer:

IPerson: interface que representa o dono de um ChangeRequest.

Módulo Transformador RDF de Testes

Este módulo é responsável por converter para termos RDF do vocabulário OSLC Quality Management as informações das ferramentas que executam testes, como as ferramentas de integração contínua. Além disso, fornece acesso a estas informações a outros módulos da plataforma através de diversas interfaces que representam os conceitos deste vocabulário. Abaixo apresentamos o diagrama

UML de estrutura composta deste módulo (**Figura 84**), suas classes internas e as interfaces que utiliza e requer de outros módulos.

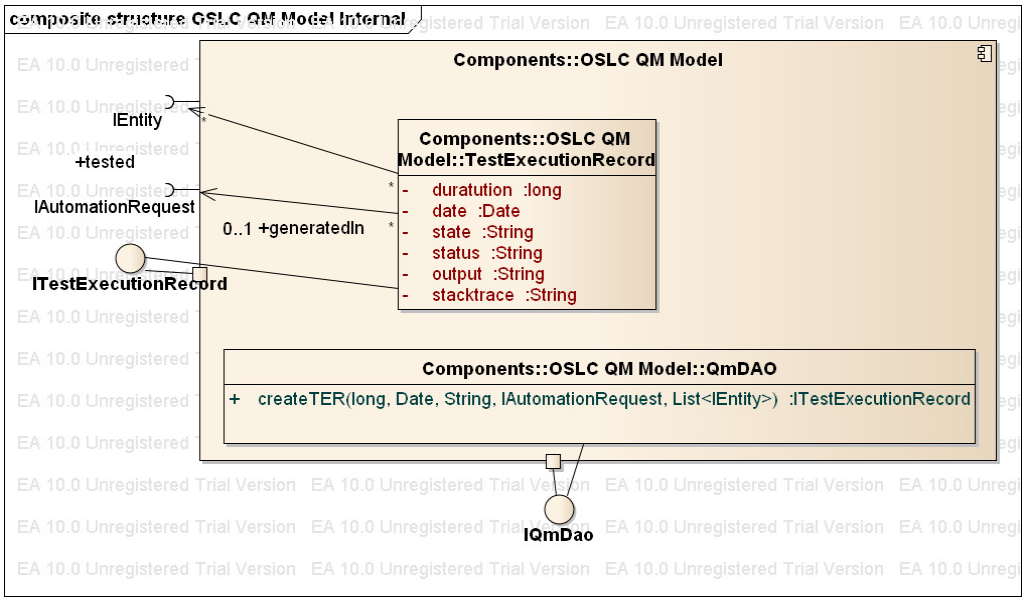

Figura 84 Diagrama de Estrutura Composta do Módulo Transformador RDF de Testes.

Classes Internas:

TestExecutionRecord: classe Java que representa uma execução de testes. Define atributos como a duração do teste, data de execução, status, saída e etc.

QmDAO: classe Java que fornece mecanismo para a recuperação e persistência de elementos definidos por este módulo.

Interfaces que provê:

ITestExecutionRecord: interface Java provida para que outros módulos possam acessar as informações de um TestExecutionRecord.

IQmDAO: interface Java que prove acesso ao mecanismo de persistência definido por este módulo.

Interfaces que requer:

IEntity: interface Java que representa a entidade de código-fonte executada durante o teste, normalmente é uma instância de IMethod.

IAutomationRequest: interface Java que é utilizada para representar o AutomationRequest onde o resultado do teste foi gerado.

Módulo Transformador RDF FOAF

Este módulo representa como classes os conceitos definidos no vocabulário FOAF. Abaixo apresentamos em detalhes a estrutura deste componente:

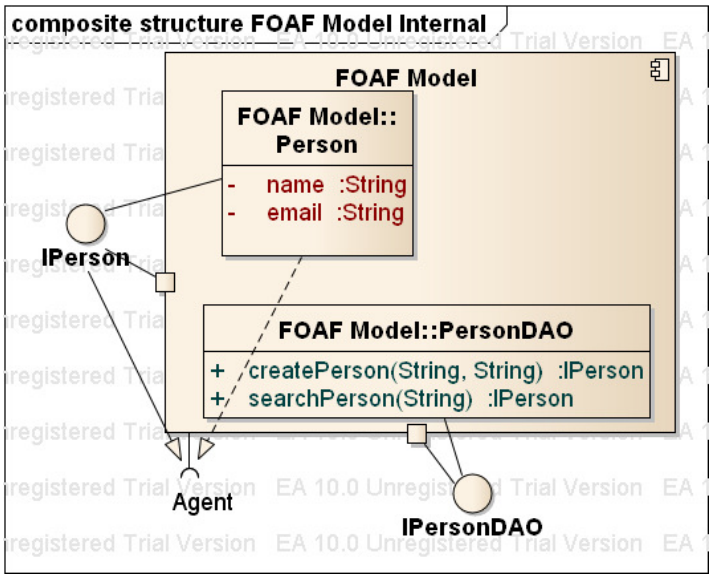

Figura 85 Diagrama de Estrutura Composta do Módulo FOAF

Classes Internas:

Person: classe Java que representa um indivíduo, como um desenvolvedor ou gerente. Define atributos como nome e e-mail.

PersonDAO: classe Java que fornece mecanismo para a recuperação e persistência de pessoas.

Interfaces que provê:

IPerson: interface Java que fornece acesso externo a elementos do tipo Person.

IPersonDAO: interface Java provida para que outros módulos possam executar métodos relacionados a recuperação, criação e modificação de pessoas.

Interfaces que requer:

Agent: interface que representa o responsável por causar um determinado impacto em uma entidade de código-fonte.

Módulo Transformador RDF DOAP

Este módulo representa conceitos como Projeto, Desenvolvedor e Versão de Projeto conforme definidos no vocabulário DOAP. Abaixo apresentamos em detalhes a estrutura deste componente nesta versão:

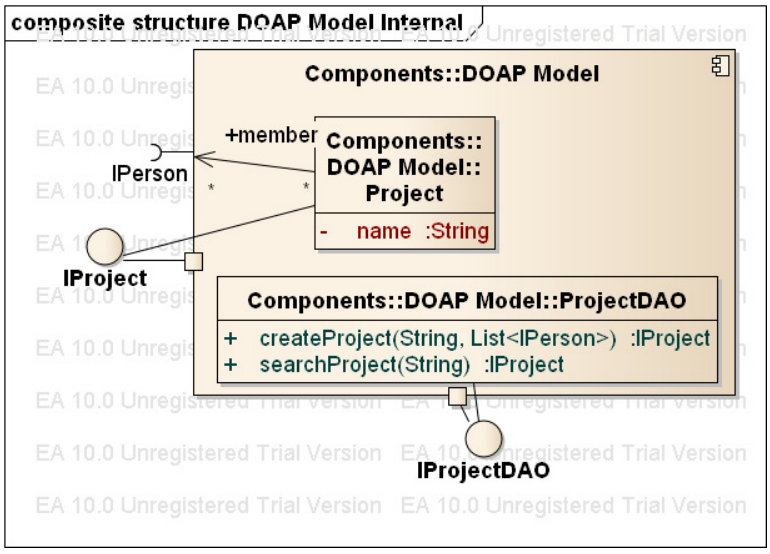

Figura 86 Diagrama de Estrutura Composta do Módulo DOAP

Classes Internas:

Project: classe Java que representa projeto de software conforme o vocabulário DOAP.

ProjectDAO: classe que fornece mecanismo para a recuperação e persistência de projetos.

Interfaces que provê:

IProject: interface Java que fornece acesso externo a elementos do tipo Project.

IProjectDAO: interface Java provida para que outros módulos possam executar métodos relacionados a recuperação, criação e modificação de projetos.

Interfaces que requer:

IPerson: interface que representa um pessoa e é utilizada para representar os membros de um projeto.

Apêndice C – Módulos Auxiliares

Módulo de Verificação do Tipo de Arquivo

Este módulo recebe um conjunto de arquivos como entrada e tem a responsabilidade de determinar qual é o tipo de arquivo em questão. Por exemplo, caso um determinando arquivo seja identificado como arquivo de código-fonte Java, o módulo coordenador de processamento de artefatos ou de controle de versão pode buscar o conector Java AST para extrair as informações dos elementos de código-fonte do arquivo.

Por conta da necessidade de tornar a plataforma extensível e flexível, este módulo conta com um mecanismo dinâmico de registro de tipos de arquivo de acordo com sua extensão. Abaixo apresentamos o diagrama de estrutura interna deste módulo:

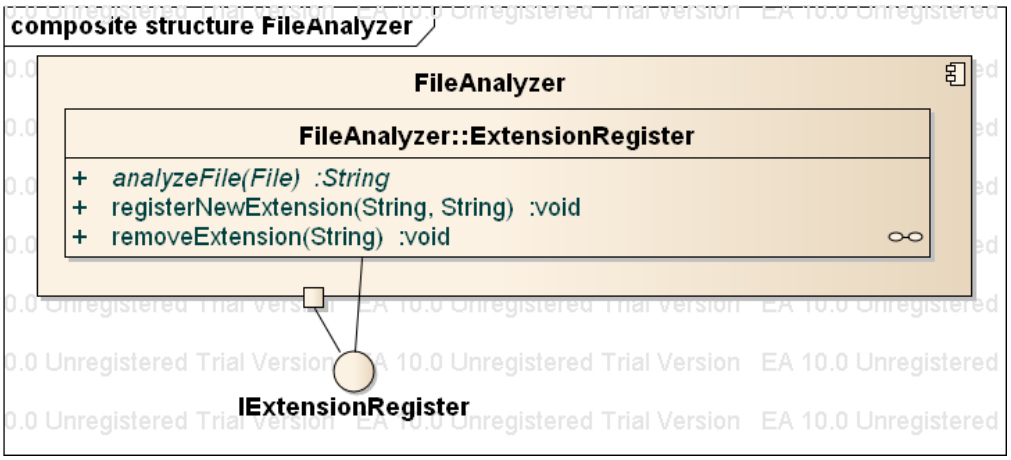

Figura 87 Diagrama de Estrutura Composta do Módulo de Verificação do Tipo de Arquivo

.

Módulos de Análise de Impactos

Este módulo é responsável por receber as entidades, atributos e relacionamento de uma versão de uma AST e comparar com as entidades,

atributos e relacionamento da versão anterior. Esta comparação objetiva determinar mudanças estruturais ou de características que possam ter ocorrido entre as duas versões. Estas duas versões podem ser originadas de dois elementos contextos, como dois commits ou duas versões de uma biblioteca. A representação das informações computadas é feita utilizando o módulo Transformador RDF de Impactos. Outra necessidade deste módulo é identificar quando duas entidades representam a mesma entidade lógica, algo necessário para a detecção de renomeamentos nas entidades. Para isto, utilizamos a biblioteca Change Distiller[92] que utiliza um algoritmo para extrair mudanças granulares de código-fonte entre versões subsequentes de classes Java. O algoritmo calcula um script de edição que contém operações básicas sobre arvores que transforma a AST antiga na nova. Apesar de esta ferramenta ter a capacidade de calcular grande parte das diferenças estruturais de um código, não foi possível utilizá-la em detrimento ao módulo de análise de impactos pelo fato dela não considerar as dependências do projeto durante a identificação das diferenças. Isto não permitiria distinguir, por exemplo, qual método é invocado quando existem dois métodos de mesmo nome e mesmo número de parâmetros durante a análise das mudanças sofridas no corpo de um método.

Com forma de comparar duas versões de uma AST utilizamos uma abordagem baseada no algoritmo da teoria de grafos conhecido como busca em profundidade[93], ou seja, inicia-se a verificação de impactos por elementos de baixo nível (ex.: chamada de método) até elementos de alto nível (ex.: classe). Sendo que um impacto em um elemento de nível hierárquico mais baixo implica na existência de impacto nos elementos de alto nível subsequentes, mesmo que estes não tenham sofrido alterações diretas em suas propriedades.

Outra característica relevante deste módulo é que ele utiliza os metadados definidos na Ontologia de Entidades, Propriedades e Relacionamentos para definir como as propriedades e relacionamentos devem ser navegados. Como por exemplo, o caso dos relacionamentos do tipo container, onde a exclusão de um relacionamento container implica na criação de um impacto do tipo exclusão para as entidades associadas a ele. Abaixo apresentamos o diagrama de estrutura composta deste módulo:

[92] https://bitbucket.org/sealuzh/tools-changedistiller/wiki/Home
[93] http://en.wikipedia.org/wiki/Depth-first_search

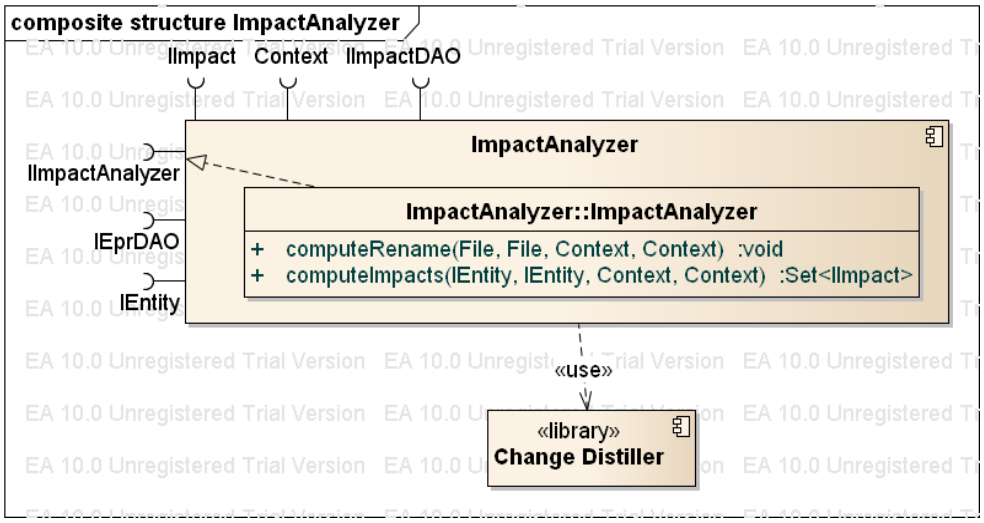

Figura 88 Diagrama de Estrutura Composta do Módulo de Análise de Impactos

Apêndice D – Módulos Coordenadores

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 se pode observar o diagrama UML de componentes dos módulos envolvidos (**Figura 89**) e o diagrama UML de atividades (**Figura 90**) coordenadas por este módulo e o detalhamento de cada atividade.

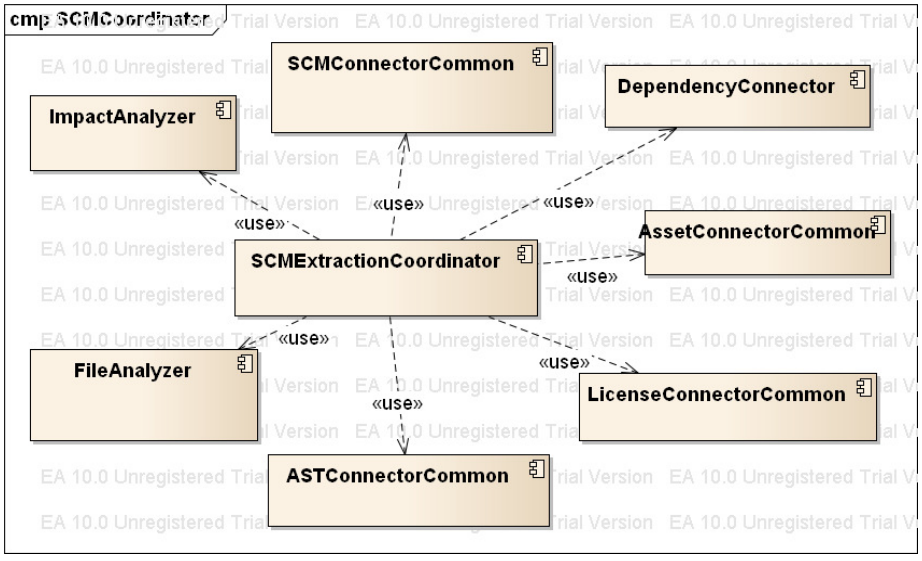

Figura 89 Diagrama de Componentes dos Módulos utilizados pelo Módulo de Coordenação de Extração de Informações de Ferramentas de Controle de Versão

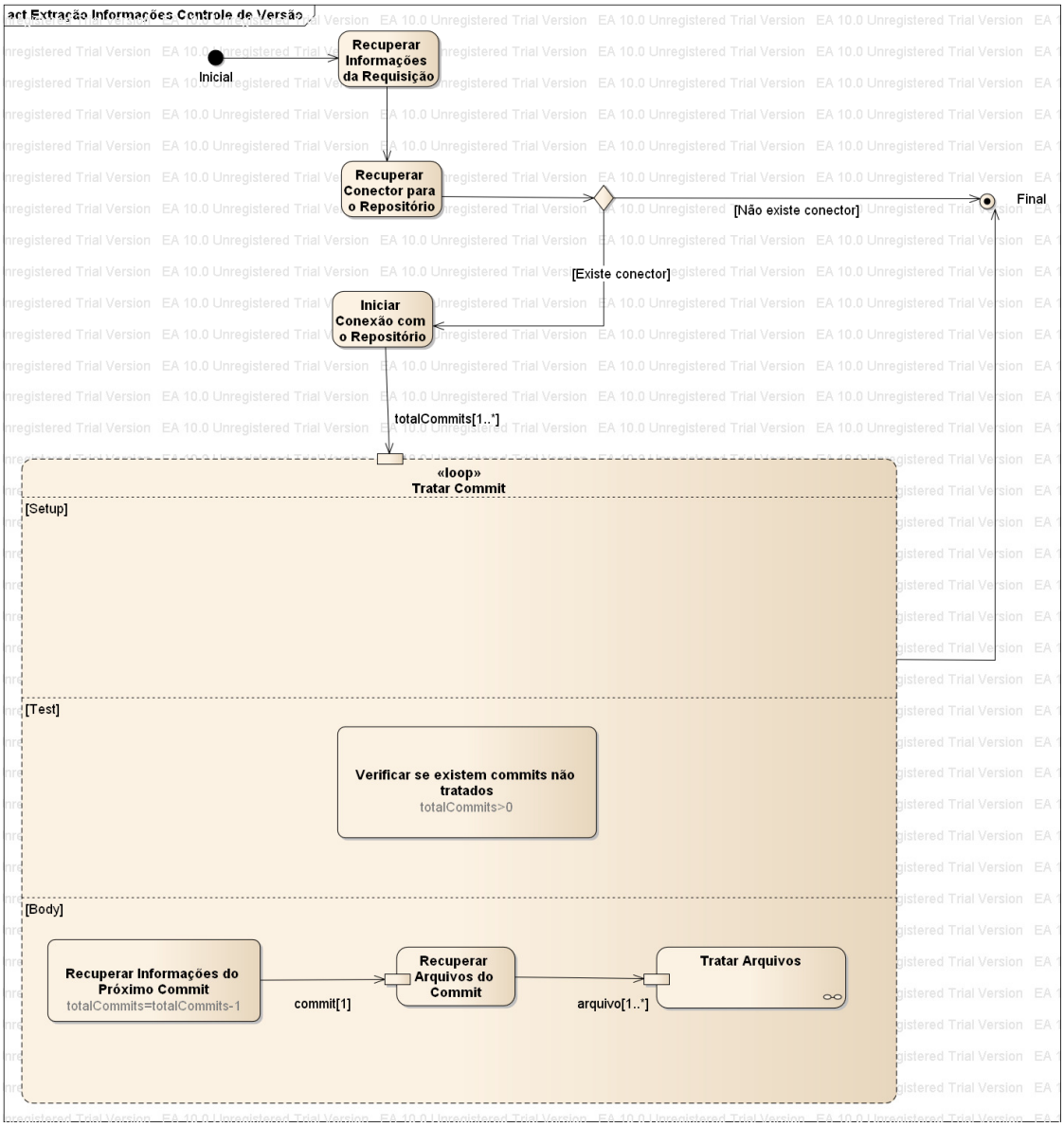

Figura 90 Diagrama de Atividades do Processo de Extração de Informações de Ferramentas de Controle de Versão

Atividades do Módulo Coordenador:

1. **Recuperar Informações da Requisição:** recupera as informações da requisição que são necessárias para iniciar o processo de extração, como: a URL do repositório, o tipo de repositório, o diretório local onde armazenar os arquivos de cada conjunto de alteração e a data de início e fim para a filtragem.

2. **Recuperar Conector para o repositório:** a partir do tipo de repositório informado na requisição, é feita uma pesquisa na plataforma para encontrar um conector que forneça acesso às informações do repositório, como seus conjuntos de alterações. Caso não exista conector, o processo é finalizado.

3. **Iniciar conexão com o repositório:** neste momento iniciasse a conexão com o repositório a partir do conector encontrado, da url e da data de início e fim que restringem os conjuntos de alterações que serão tratados. O resultado desta atividade é o conjunto de alterações que serão tratadas.

4. **Tratar Commit:** esta atividade é um loop que executa enquanto houver conjuntos de alterações a serem tratados. Para cada conjunto de alteração, recuperar suas informações como seu identificador, o autor e a data de criação. Após isso, recupera os arquivos modificados ou incluídos para serem tratados pela Atividade **Tratar Arquivos.**

5. **Tratar Arquivos:** esta é uma atividade que utiliza o Módulo de Verificação de Tipo de Arquivo para determinar o tipo do arquivo. Caso ele descreva uma licença, a atividade **Tratar Arquivo de Licença** é chamada, caso seja um arquivo de dependência, a atividade **Tratar Arquivo de Dependência** é chamada. E por último, caso seja um arquivo de código-fonte, a atividade **Trata Arquivo de Código-Fonte** é chamada e logo após, a atividade **Tratar Impactos na AST do Arquivo**. Abaixo o diagrama interno de atividades da atividade **Tratar Arquivos** (**Figura 91**) e suas respectivas atividades:

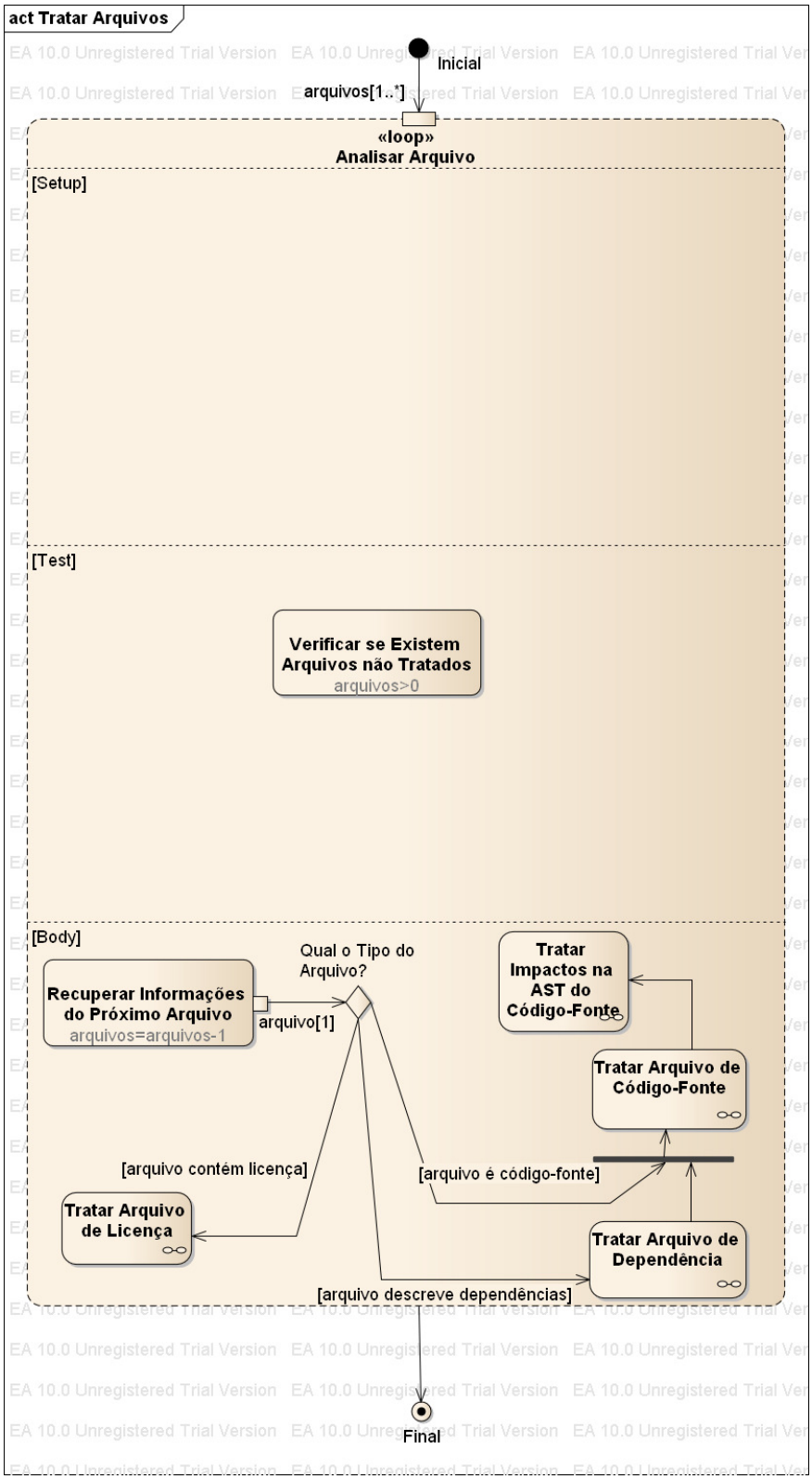

Figura 91 Diagrama de Atividades Tratar Arquivos

Atividades da Atividade Tratar Arquivos:

5.1 **Tratar Arquivo de Licença:** esta atividade verifica qual a licença está associada ao arquivo e armazena está informação. Abaixo o diagrama de atividades que representa esta atividade:

Figura 92 Diagrama de Atividades Tratar Arquivo de Licença

5.2 **Tratar Arquivo de Dependência:** esta atividade verifica as dependências indicadas no arquivo e realiza seu download. Abaixo o diagrama que representa esta atividade:

Figura 93 Diagrama de Atividades Tratar Arquivo de Dependências

5.3 Tratar Arquivo de Código-Fonte: esta atividade recupera o conector da linguagem de programação utilizada no arquivo e recupera as dependências

correntes do projeto que foram encontradas pela atividade anterior. A partir destas informações, constrói a AST do código-fonte e armazena suas informações. Abaixo detalhamos esta atividade:

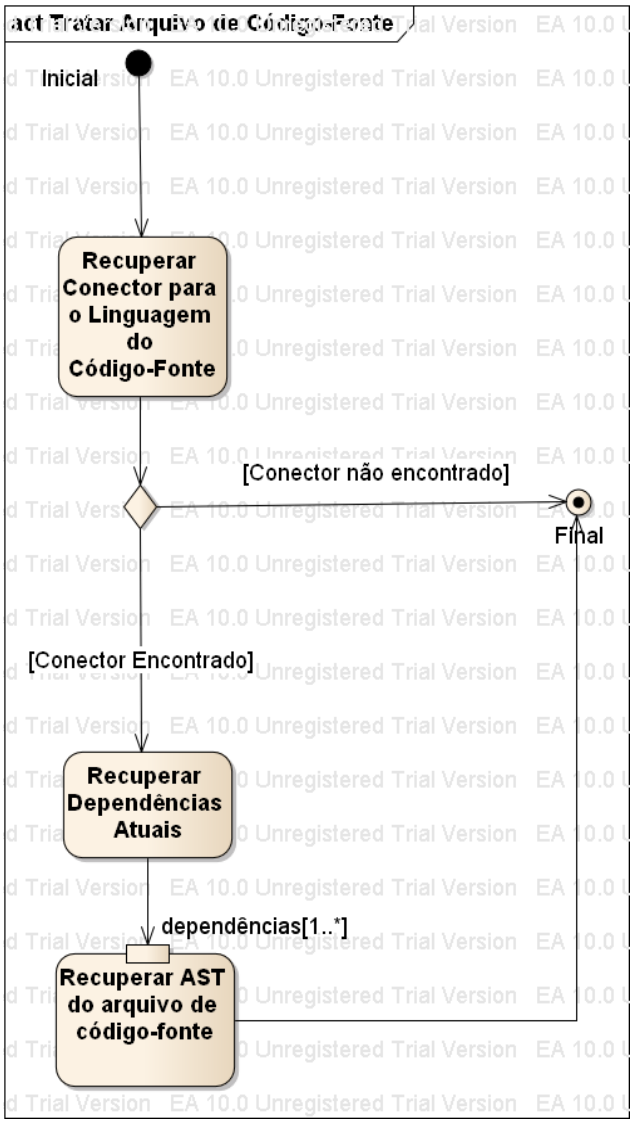

Figura 94 Diagrama de Atividade Tratar Arquivo de Código-Fonte.

5.4 **Tratar Impactos na AST do Código-Fonte:** esta atividade recupera os elementos da AST atual do arquivo e compara com os elementos de sua versão anterior utilizando o módulo de Análise de Impactos.

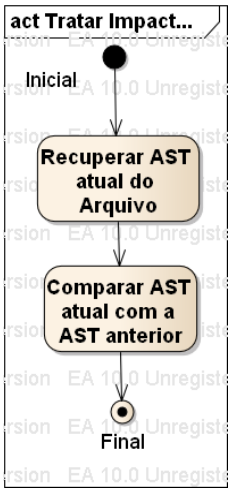

Figura 95 Diagrama de Atividade Tratar Impactos na AST do Arquivo

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 se pode observar o diagrama UML de componentes dos módulos envolvidos (**Figura 96**) e o diagrama UML de atividades (**Figura 97**) coordenadas por este módulo e o detalhamento de cada atividade.

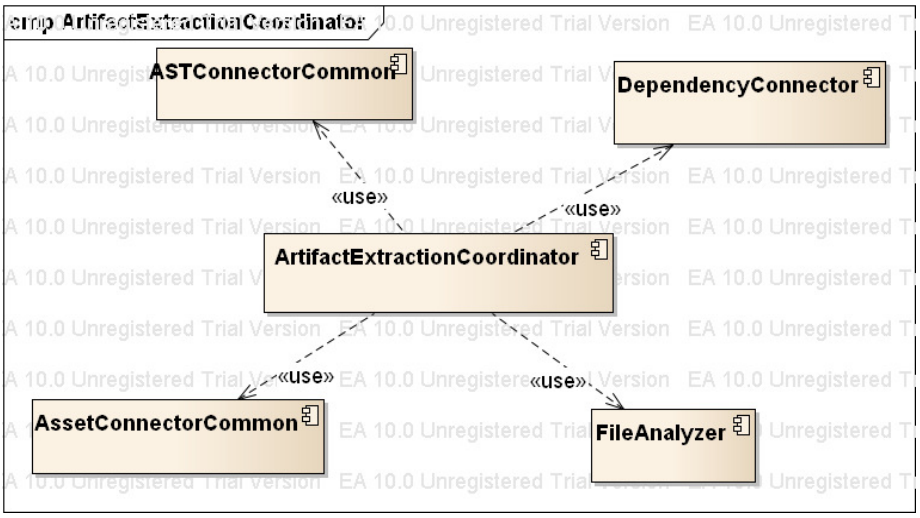

Figura 96 Diagrama de Componentes dos Módulos utilizados pelo Módulo Coordenador do Processamento da Extração de Informações de uma Versão de uma Biblioteca.

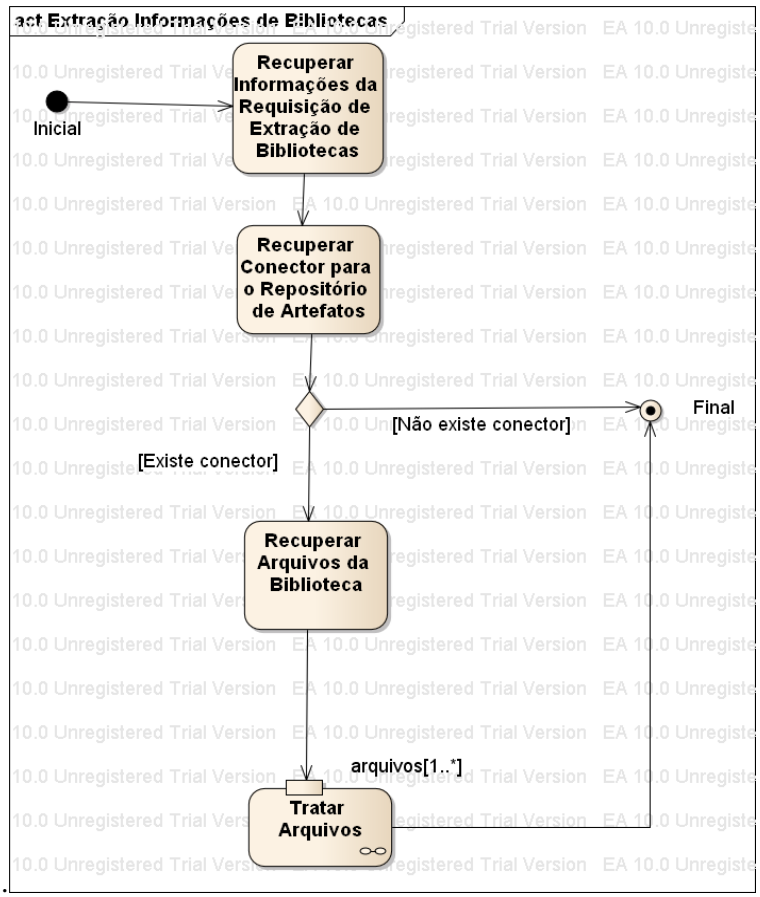

Figura 97 Diagrama de Atividades do Processo de Extração de Informações de uma Versão de uma Biblioteca.

Atividades:

1. **Recuperar Informações da Requisição de Extração de Bibliotecas:** recupera as informações da requisição que são necessárias para iniciar o processo de extração, como: a url do repositório onde está localizada a biblioteca, o tipo de repositório e o identificador da versão da biblioteca que será analisada.

2. **Recuperar Conector para o Repositório de Artefatos:** a partir do tipo de repositório informado na requisição, é feita uma pesquisa na plataforma para encontrar um conector que forneça acesso às informações do repositório. Caso não exista conector, o processo é finalizado.

3. **Recuperar Arquivos da Biblioteca:** neste momento é feito o download da biblioteca e os arquivos presentes nela serão recuperados. Estes arquivos serão passados para a Atividade **Tratar Arquivos** da mesma forma que ocorre no processo de extração de informações de ferramentas de controle de versão.

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 se pode observar o diagrama UML de componentes dos módulos envolvidos (**Figura 98**) e o diagrama UML de atividades (**Figura 99**) coordenadas por este módulo e o detalhamento de cada atividade.

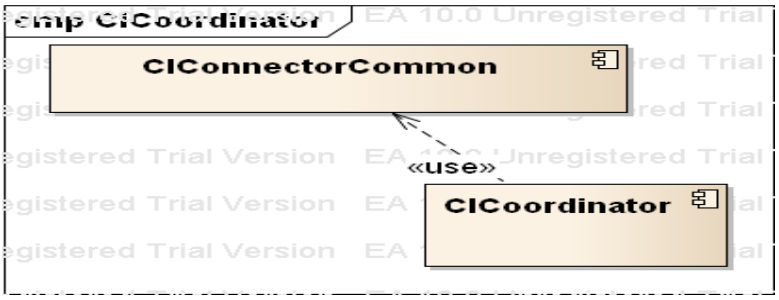

Figura 98 Diagrama de Componentes dos Módulos utilizados pelo Módulo Coordenador de Extração de Informações de Ferramentas de Integração Contínua.

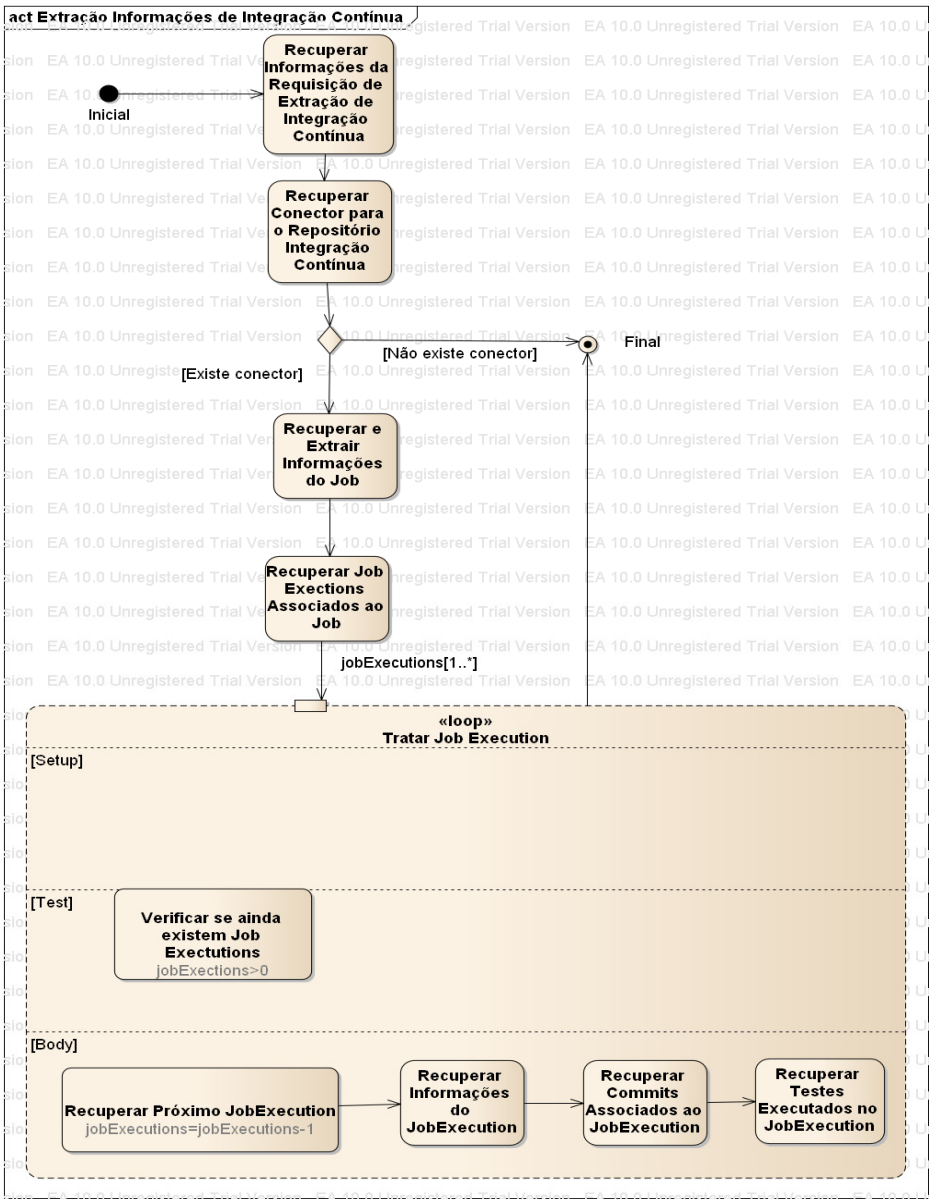

Figura 99 Diagrama de Atividades Extrair Informações de Integração Contínua.

Atividades:

1. **Recuperar Informações da Requisição de Extração de Integração Contínua:** recupera as informações da requisição que são necessárias para iniciar o processo de extração, como: a url do repositório de integração contínua, o tipo de repositório e o plano de automação que será extraído.

2. **Recuperar Conector para o Repositório de Integração Contínua:** a partir do tipo de repositório informado na requisição, é feita uma pesquisa na plataforma para encontrar um conector que forneça acesso às informações do repositório. Caso não exista conector, o processo é finalizado.

3. **Recuperar e Extrair Informações do Job:** esta atividade recupera o plano de automação requisitado e extrai suas informações, como nome e url.

4. **Recuperar Job Executions associados ao Job:** esta atividade recupera a listagem de todas as execuções associados ao plano de automação em questão.

5. **Tratar Job Execution:** esta atividade é um loop que é executado enquanto houver execuções não tratadas. Para cada execução, é feita a recuperação das informações associadas, como o usuário quem faz a requisição, qual foi sua duração, quais os commits estão associados e quais os testes foram executados e qual seu resultado.

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 na AST dos elementos de código-fonte presentes em duas versões de uma biblioteca. Abaixo se pode observar o diagrama UML de componentes dos

módulos envolvidos (**Figura 100**) e o diagrama UML de atividades (**Figura 101**) coordenadas por este módulo e o detalhamento de cada atividade.

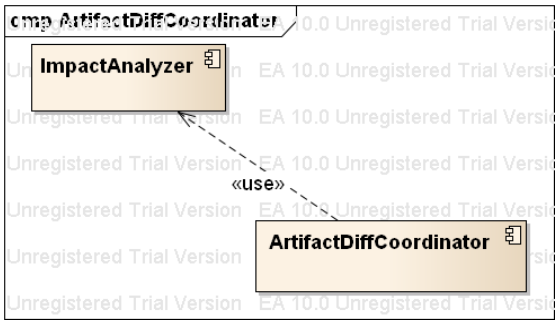

Figura 100 Diagrama de Componentes dos Módulos utilizados pelo Módulo de Coordenação de Extração de Diferenças entre duas Versões de uma Biblioteca.

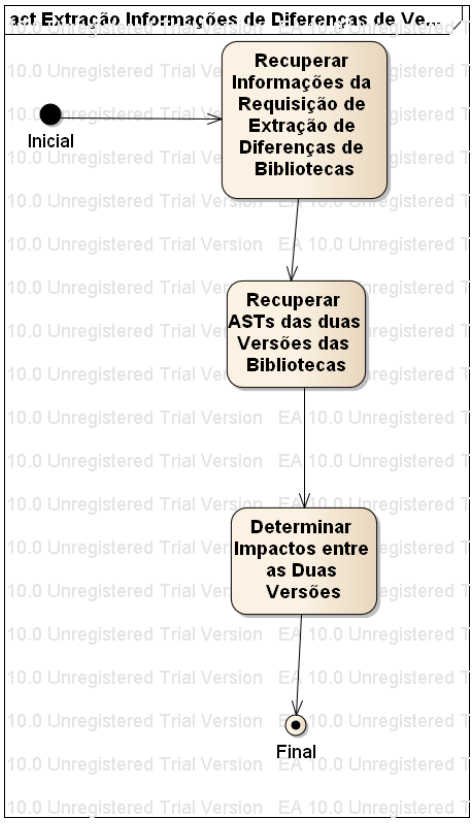

Figura 101 Diagrama de Atividades do Processo de Extração das diferenças entre duas versões de uma biblioteca.

Atividades:

1. **Recuperar Informações da Requisição de Extração de Diferenças entre duas versões de uma biblioteca:** recupera as informações da requisição que são necessárias para iniciar o processo de extração, como: a url do repositório onde está localizada a biblioteca, o tipo de repositório e os identificadores das versões da biblioteca que serão analisadas.

2. **Recuperar ASTs das Duas Versões da Biblioteca:** esta atividade recuperar todos os elementos de código-fonte gerados durante a extração de informações das duas versões de uma biblioteca.

3. **Determinar Impactos entre as Duas Versões de uma Biblioteca:** esta atividade compara as versões de cada elemento de código-fonte para determinar suas diferenças e para gerar a lista de impactos utilizando o Módulo de Análise de Impactos.

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 se pode observar o diagrama UML de componentes dos módulos envolvidos (**Figura 102**) e o diagrama UML de atividades (**Figura 103**) coordenadas por este módulo e o detalhamento de cada atividade.

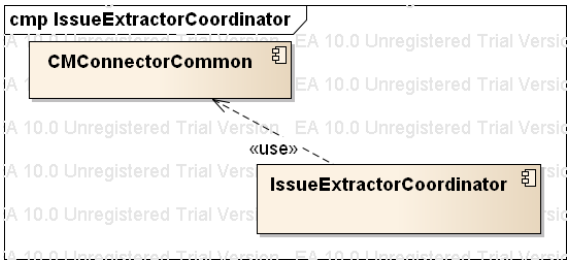

Figura 102 Diagrama de Componentes dos Módulos utilizados pelo Módulo Coordenador de Extração de Informações de Ferramentas de Gerenciamento de Demandas/Defeitos.

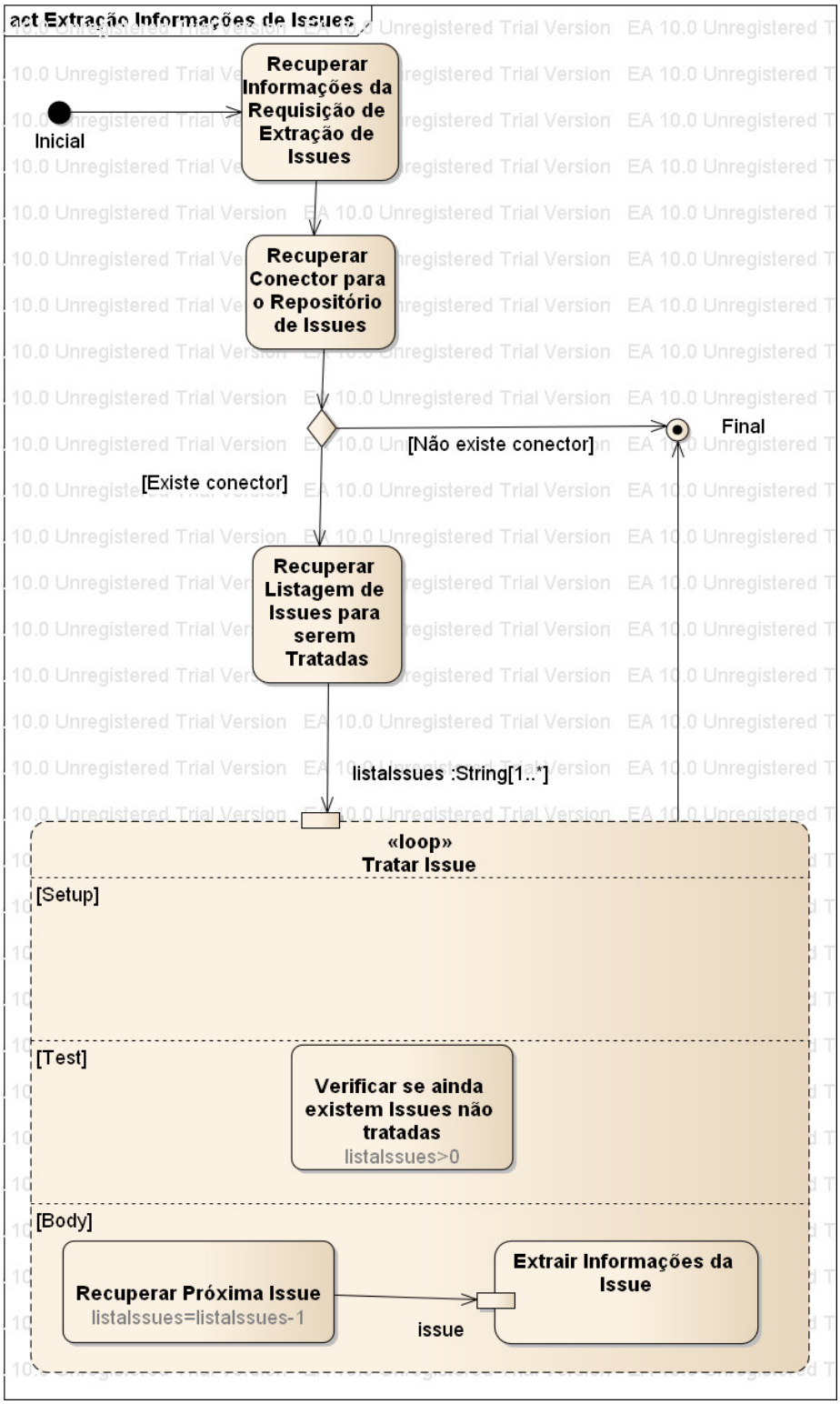

Figura 103 Diagrama de Atividades Extrair Informações de Demandas/Defeitos.

Atividades:

1. **Recuperar Informações da Requisição de Extração de Demandas/Defeitos:** recupera as informações da requisição que são necessárias para iniciar o processo de extração, como: a url do repositório onde estão localizados os demandas/defeitos informados, o tipo de repositório e os identificadores das demandas/defeitos.

2. **Recuperar Conector para o Repositório de Demandas/Defeitos:** a partir do tipo de repositório informado na requisição, é feita uma pesquisa na plataforma para encontrar um conector que forneça acesso às informações do repositório. Caso não exista conector, o processo é finalizado.

3. **Tratar Demanda:** esta atividade é um loop que é executado enquanto houver identificadores de demandas/defeitos. Para cada identificador, é feita a recuperação das informações associadas à issue, como o seu título, tipo, criador e descrição.

Apêndice E – Consultas SPARQL “Perguntas Frequentes dos Desenvolvedores”

1. Quem alterou este código, categorizado por pessoa?

```
PREFIX sm:<http://semancti/>

SELECT ?developer ?element  (COUNT(?impact) AS ?count)
WHERE {
?developer a sm:Person.
?developer ^sm:commitAuthor  ?changeSet .
?changeSet a sm:ChangeSet.
?changeSet  ^sm:contextTo ?impact.
?impact sm:impactOfAfter ?element.
FILTER NOT EXISTS { ?impact sm:impactType 'NOTHING' }
FILTER ( ?element IN ( $elements )  )}
GROUP BY ?developer ?element
```

2. Para quem atribuir uma revisão de código? Quem tem conhecimento para realizar a revisão de código?

```
PREFIX sm:<http://semancti/>

SELECT ?developer ?element  (COUNT(?impact) AS ?count)
WHERE {
?developer a sm:Person.
?developer ^sm:commitAuthor  ?changeSet .
?changeSet a sm:ChangeSet.
?changeSet  sm:commitDate ?impactDate.
?changeSet  ^sm:contextTo ?impact.
?impact sm:impactOfAfter ?element.
FILTER NOT EXISTS { ?impact sm:impactType 'NOTHING' }
FILTER ( ?element IN (  $element )  )}
GROUP BY ?developer ?element
ORDER BY DESC(?count) DESC(?impactDate)
```

3. Quem alterou classes que eu modifico?

```
PREFIX sm:<http://semancti/>

SELECT ?otherDeveloper  (COUNT(?element) AS ?count)
WHERE {
$currentDeveloper ^sm:commitAuthor  ?changeSet .
?changeSet  ^sm:contextTo ?impact.
?impact sm:impactOfAfter ?element.
?element a sm:Class.
?element ^sm:impactOfAfter ?impactOther.
?impactOther sm:contextTo ?changeSetOther.
?changeSetOther sm:commitAuthor ?otherDeveloper.
?otherDeveloper a sm:Person.
FILTER    (!sameTerm($currentDeveloper,    ?otherDeveloper)    &&
!sameTerm(?changeSet,?changeSetOther)                          &&
!sameTerm(?impact,?impactOther)  )
FILTER NOT EXISTS { ?impact sm:impactType 'NOTHING' }
FILTER NOT EXISTS { ?impactOther sm:impactType 'NOTHING' } }
GROUP BY ?otherDeveloper ORDER BY DESC(?count)
```

4. Quem está utilizando esta API [que eu vou modificar]?

```
PREFIX sm:<http://semancti/>

SELECT ?element ?elementOther
WHERE {
?changeSet a sm:ChangeSet.
?changeSet  sm:elements ?element .
?changeSetOther a sm:ChangeSet.
?changeSetOther    sm:elements  ?elementOther.  ?elementOther  a
sm:Method.
?elementOther ^sm:isRelationshipOf ?hasInvokation. ?hasInvokation
a sm:HasInvokation.
?hasInvokation rdf:value ?invokation.
?invokation a sm:Invokation.
?invokation  ^sm:isRelationshipOf  ?relationship.  ?relationship  a
sm:Invokes.
?relationship rdf:value ?element.
```

```
FILTER (!sameTerm(?changeSet, ?changeSetOther))            FILTER
(?element IN ($element))}
GROUP BY ?element  ?elementOther
```

5. Quem são os criadores desta API [que eu vou modificar]?

```
PREFIX sm:<http://semancti/>

SELECT ?developer ?element  (COUNT(?impact) AS ?count)
WHERE {
?developer a sm:Person.
?developer ^sm:commitAuthor  ?changeSet .
?changeSet a sm:ChangeSet.
?changeSet  sm:commitDate ?impactDate.
?changeSet  ^sm:contextTo ?impact.
?impact sm:impactOfAfter ?element.
FILTER NOT EXISTS { ?impact sm:impactType 'NOTHING' }
FILTER EXISTS { ?impact sm:impactType 'ADD' }
FILTER ( ?element IN (  $elementAPI )  )}
GROUP BY ?developer ?element
ORDER BY DESC(?count) DESC(?impactDate)
```

6. Quem é o dono deste pedaço de código? Quem modificou por último?

```
PREFIX sm:<http://semancti/>

SELECT ?element ?developer  ?maxdate
WHERE {
?changeSet sm:commitCommiter ?developer.
?changeSet  ^sm:contextTo ?impact.
?impact sm:impactOfAfter ?element.
?changeSet sm:commitDate ?maxdate.
{
SELECT ?element (MAX(?date) AS ?maxdate)
WHERE {
?changeSet sm:commitCommiter ?developer.
?changeSet  ^sm:contextTo ?impact.
?impact sm:impactOfAfter ?element.
?changeSet sm:commitDate ?date.
FILTER NOT EXISTS { ?impact sm:impactType 'NOTHING' }
```

```
FILTER (?element IN ($element)  )
}
GROUP BY  ?element }}
```

7. Com quem falar quando se tem que trabalhar com pacotes que nunca se trabalhou?

```
PREFIX sm:<http://semancti/>

SELECT  ?developer (COUNT (?impact) AS ?total)
WHERE {
?changeSet a sm:ChangeSet.
?changeSet sm:commitCommiter ?developer.
?changeSet  ^sm:contextTo ?impact.
?impact sm:impactOfAfter  ?relationship.
?relationship rdf:value ?entity.
?entity ^sm:elements $libraryVersion.
FILTER NOT EXISTS { ?impact sm:impactType 'NOTHING' } }
GROUP BY ?developer
ORDER BY DESC(?total)
```

8. Qual é a classe mais popular? [Qual classes tem sido a mais modificada]?

```
PREFIX sm:<http://semancti/>

SELECT  ?classe (COUNT (?impact) AS ?total)
WHERE {
?impact sm:impactOfAfter  ?classe.
?classe a sm:Class.
FILTER NOT EXISTS {
?impact sm:impactType 'NOTHING' } }
GROUP BY ?classe
ORDER BY DESC(?total)
LIMIT 1
```

9. Qual outro código que eu trabalhei utiliza esta função utilitária?

```
PREFIX sm:<http://semancti/>

SELECT  ?method
WHERE {
$developer  ^sm:commitCommiter ?changeSet.
?changeSet  ^sm:contextTo ?impact.
?impact sm:impactOfBefore  ?relationship.
?relationship a sm:Invokes.
?relationship sm:isRelationshipOf ?invokation.
?invokation ^rdf:value ?hasInvokation.
?hasInvokation sm:isRelationshipOf ?method.
?method a sm:Method.
?relationship rdf:value ?element.
FILTER ( ?element IN ($function)  ) }
```

10. Qual código recentemente modificado que está relacionado a mim?

```
PREFIX sm:<http://semancti/>

SELECT  ?otherDeveloper  ?element  (COUNT(?impactOther)  AS  ?count)
WHERE
{
$currentDeveloper ^sm:commitAuthor  ?changeSet .
?changeSet  ^sm:contextTo ?impact.
?impact sm:impactOfAfter ?element.
?element a sm:Class.
?element ^sm:impactOfAfter ?impactOther.
?impactOther sm:contextTo ?changeSetOther.
{
      SELECT ?changeSetOther
      WHERE{
      ?changeSetOther sm:commitDate ?date.
      FILTER (!sameTerm(?changeSet,?changeSetOther))
}
ORDER BY DESC(?date)
LIMIT 10
}
?changeSetOther sm:commitAuthor ?otherDeveloper.
```

```
?otherDeveloper a sm:Person.
FILTER      (!sameTerm($currentDeveloper,?otherDeveloper)      &&
!sameTerm(?impact,?impactOther))
FILTER NOT EXISTS { ?impact sm:impactType 'NOTHING' }
FILTER NOT EXISTS { ?impactOther sm:impactType 'NOTHING' } }
GROUP BY ?otherDeveloper ?element
ORDER BY DESC(?count)
```

11. Quem causou a quebra da construção? (Quem é responsável pela quebra dos testes?)

```
PREFIX sm:<http://semancti/>

SELECT ?developer
WHERE {
?ter a sm:TestExecutionRecord.
?ter sm:testStatus 'FAILED'.
?ter sm:testExecutedIn ?moduleBuild.
?moduleBuild sm:builtIn ?jobExecution.
?jobExecution         sm:changeSetsInBuild    ?changeSetsInBuild.
?changeSetsInBuild sm:commitCommiter ?developer.
?ter sm:executedTestMethod ?method.
?method ^sm:isRelationshipOf ?hasInvokation.
?hasInvokation a sm:HasInvokation.
?hasInvokation rdf:value ?invokation.
?invokation a sm:Invokation.
?invokation ^sm:isRelationshipOf ?relationship.
?relationship a sm:Invokes.
?relationship rdf:value ?element.
?element ^sm:impactOfAfter  ?impact .
?impact sm:contextTo ?changeSetsInBuild.
FILTER NOT EXISTS { ?impact sm:impactType 'NOTHING' } }
```

12. Quem é dono deste caso de teste? (Quem resolveu a última tarefa que corrige o caso de teste?)

```
PREFIX sm:<http://semancti/>

SELECT ?developer
WHERE {
?developer ^sm:owner ?issue.
?issue sm:status ?status.
?status sm:statusName ‘Closed’.
?issue ^sm:commitIssue ?changeSet.
?changeSet ^sm:impactOfAfter ?method.
?method ^sm:executedTestMethod ?ter.
?ter a sm:TestExecutionRecord.
?ter sm:testStatus 'SUCCESS'.
?ter sm:testExecutedIn ?moduleBuild.
?moduleBuild sm:builtIn ?jobExecution.
?jobExecution  sm:changeSetsInBuild ?changeSet.
?changeSet ^sm:contextTo ?impact.
FILTER ( ?ter IN ($ter)  )
FILTER NOT EXISTS { ?impact sm:impactType 'NOTHING' } }
```

13. O que mais foi modificado quando este código foi modificado ou inserido?

```
PREFIX sm:<http://semancti/>

SELECT ?element ?otherElement (COUNT(?changeSet) as ?total)
WHERE {
?changeSet  a sm:ChangeSet.
?changeSet  ^sm:contextTo ?impact.
?impact sm:impactOfAfter ?element.
?changeSet  ^sm:contexto ?impact2.
?impact2 sm:impactOfAfter ?otherElement.
FILTER NOT EXISTS { ?impact sm:impactType 'NOTHING' }
FILTER (?element IN ($element)  )
FILTER (!sameTerm(?impact, ?impact2))}
GROUP BY  ?element
```

14. Como este código interage com bibliotecas?

```
PREFIX sm:<http://semancti/>

SELECT DISTINCT ?element ?relationship ?entity
WHERE {
?changeSet a sm:ChangeSet.
?changeSet   sm:elements ?element.
?element a ?classe.
?element ^sm:isRelationshipOf ?relationship.
?relationship rdf:value ?entity.
?entity a ?otherClasse.
?entity ^sm:elements $library.      }
```