UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

UM PROCESSO PARA O USO DE LINGUAGENS DE CONSULTA EM CÓDIGO FONTE

DISSERTAÇÃO DE MESTRADO

Gustavo Stangherlin Cantarelli

Santa Maria, RS, Brasil
2012

UM PROCESSO PARA O USO DE LINGUAGENS DE CONSULTA EM CÓDIGO FONTE

Gustavo Stangherlin Cantarelli

Dissertação apresentada ao Curso de Mestrado do Programa de Pós-Graduação em Informática, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Ciência da Computação**

Orientador: Prof. Dr. Eduardo Kessler Piveta

Santa Maria, RS, Brasil

2012

Ficha catalográfica elaborada através do Programa de Geração Automática da Biblioteca Central da UFSM, com os dados fornecidos pelo(a) autor(a).

Stangherlin Cantarelli, Gustavo
Um processo para o uso de linguagens de consulta em código fonte / Gustavo Stangherlin Cantarelli.-2012.
145 p.; 30cm

Orientador: Eduardo Kessler Piveta
Dissertação (mestrado) - Universidade Federal de Santa Maria, Centro de Tecnologia, Programa de Pós-Graduação em Informática, RS, 2012

1. Processos de *Software* 2. Linguagens de Consulta 3. Orientação a Objetos 4. Orientação a Aspectos I. Kessler Piveta, Eduardo II. Título.

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Informática

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

UM PROCESSO PARA O USO DE LINGUAGENS DE CONSULTA EM CÓDIGO FONTE

elaborada por
Gustavo Stangherlin Cantarelli

como requisito parcial para obtenção do grau de
Mestre em Ciência da Computação

COMISSÃO ORGANIZADORA

Eduardo Kessler Piveta, Dr.
(Presidente/Orientador)

Lisandra Manzoni Fontoura, Dra. (UFSM)

Marcelo Soares Pimenta, Dr. (UFRGS)

Santa Maria, 08 de novembro de 2012.

Dedico este trabalho à memória de meus pais, Gelso Antonio Cantarelli e Maria Stangherlin Cantarelli, e de minha avó Angela Segala Stangherlin, que sempre me apoiaram e contribuíram para minha formação pessoal e profissional com seus valores, conselhos e exemplos de vida.

“Não existem pessoas de sucesso e pessoas fracassadas.

O que existem são pessoas que lutam pelos seus sonhos ou desistem deles.”

(AUGUSTO CURY)

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

UM PROCESSO PARA O USO DE LINGUAGENS DE CONSULTA EM CÓDIGO FONTE

AUTOR: GUSTAVO STANGHERLIN CANTARELLI
ORIENTADOR: DR. EDUARDO KESSLER PIVETA
Data e Local da Defesa: Santa Maria, 08 de novembro de 2012.

As buscas em código-fonte estão ganhando cada vez mais espaço devido ao atual aumento da complexidade dos sistemas de *software* e também à necessidade de melhorias em código-fonte. Embora os paradigmas de programação orientada a objetos e de programação orientada a aspectos possuam diversos recursos para melhorar o reuso e a clareza de código, quando é necessária manutenção de trechos de código, os programadores tendem a reduzir sua produtividade em função de problemas em localizar os trechos a serem corrigidos ou melhorados. Visando as atividades de manutenção, este trabalho apresenta um processo de consulta que pode ser aplicado em repositórios de código-fonte. Tal processo é exemplificado através de três instanciações: buscas em código orientado a objetos usando SQL e SPARQL, e em código orientado a aspectos usando XML, evidenciando que seu uso poderá ser de grande valia para qualquer combinação entre ferramentas e linguagens de consulta.

Palavras-chave: Processos de *Software*. Linguagens de Consulta. Orientação a Objetos. Orientação a Aspectos.

ABSTRACT

Master Course Dissertation
Professional Graduation Program in Informatics
Universidade Federal de Santa Maria

USE OF SEARCH LANGUAGES IN SOUCE CODE REPOSITORY

AUTHOR: GUSTAVO STANGHERLIN CANTARELLI
ADVISER: EDUARDO KESSLER PIVETA
Defense Place and Date: Santa Maria, November 08nd, 2012.

The search in the source code is gaining more and more space because of the increasing complexity of current *software* systems and also the need for improvements in source code. Although the paradigms of object-oriented programming and aspect-oriented programming have several features to improve code reuse and clarity when maintenance of code is required, developers tend to reduce productivity because of problems on locating the parts to be corrected or improved. Aiming maintenance activities, this paper presents a search code process that can be applied to source code repositories. This process is exemplified through three instantiations: searches in object-oriented code using SPARQL and SQL, and aspect-oriented code using XML, suggesting that its use may be of great value for any combination of tools and query languages.

Keywords: Software Process. Search Languages. Object-Oriented. Aspect-Oriented.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AO	*Aspect-Oriented*
AOP	*Aspect Oriented Programming*
API	*Application Programming Interface*
APT	*Annotation Processing Tool*
AST	*Abstract Sintax Tree*
CACV	*Computer-Aided Constraint Verification*
DOM	*Document Object Model*
DSL	*Domain-Specific Languages*
EPF	*Eclipse Process Framework*
JDT	*Java Develpment Tools*
JTL	Java *Tools Language*
MOP	*Meta-Object Protocol*
OA	Orientação a Aspectos
ODE	*Ontology-based software Development Environment*
OMG	*Object Management Group*
OO	*Object-Oriented* (Orientação a Objetos)
OOP	*Object-Oriented Programming*
OWL	*Web Ontology Language*
POA	Programação Oriantada a Aspectos
POO	Programação Orientada a Objetos
RAD	*Rapid Application Development*
RUP	*Rational Unified Process*
SCML	*Source Code Modeling Language*
SGBD	Sistema Gerenciador de Banco de Dados
SPEM	*Software Process Engineering Metamodel*
SQL	*Structured Query Language*
UFSM	Universidade Federal de Santa Maria
UML	*Unified Modeling Language*
UI	*User Interface*
UP	*Unified Process*
XML	*eXtensible Markup Language*
XP	*eXtreme Programming*

LISTA DE ANEXOS

LISTA DE APÊNDICES

SUMÁRIO

1 INTRODUÇÃO

Novos paradigmas e linguagens de programação estão sendo constantemente propostos. Embora tais paradigmas e linguagens tenham sido criados para facilitar o desenvolvimento e a manutenção de sistemas, ainda existem diversas oportunidades de melhoria e limitações existentes em código-fonte, tanto novo quanto legado.

Visando aumentar a qualidade do produto final, e considerando a necessidade de armazenamento de informações, as linguagens de consultas começaram a ter importância na obtenção de resultados de buscas em repositórios (KORTH, SILBERSCHATZ; SUDARSHAN, 2006). Dentre as linguagens de consulta mais utilizadas está *Structured Query Language* SQL (ELMASRI; NAVTHE, 2011), a qual possibilita buscas em repositórios de dados. Para a consulta em código fonte, há poucas opções (tais como JQuery[1] e JTL[2]), porém diversas necessidades de aplicação, dentre as quais podem ser destacadas o reaproveitamento de código, a correções de defeitos e a busca por oportunidades de refatoração.

Este trabalho propõe um processo de busca em código fonte, que possibilita tanto o uso de linguagens de consulta existentes quanto servir de apoio para a criação de novas linguagens. Para avaliação do uso do processo proposto, foram feitas três instanciações como prova de conceito, sendo duas delas para o paradigma de Programação Orientada a Objetos (POO). A primeira instanciação mostra como o processo pode ser usado para buscar construções em código orientado a objetos usando a linguagem de consulta SQL com meta-dados armazenados em um Sistema Gerenciador de Banco de Dados (SGBD) relacional. A segunda instanciação mostra o processo sendo usado para buscar informações, também para código orientado a objetos, usando a linguagem de consulta SPARQL em uma ontologia. Por sua vez, a terceira instanciação mostra o processo usado para código orientado a aspectos, utilizando a linguagem de consulta XQuery em dados armazenados em arquivos XML. O objetivo desta escolha dos modelos de representação (relacional, ontologias e XML) é mostrar diferentes configurações do processo.

Para as instanciações OO, foi desenvolvida uma ferramenta - implementada em Java (na plataforma Eclipse), chamada OopExtract, que realiza percursos em Árvores Sintáticas Abstratas (ASTs) para a extração de metadados em código-fonte Java. Para popular a

[1] JQuery: a query-based code browser: http://jquery.cs.ubc.ca/
[2] JTL - The Java Tools Language: http://openjtl.sourceforge.net/

ontologia utilizada na segunda instanciação, foi utilizada a *Application Programming Interface* (API) Jena (ASF, 2012). Este trabalho está inserido no contexto de um projeto de pesquisa mais amplo que busca fornecer meios de buscar por oportunidades de refatoração através de linguagens de consulta (sejam linguagens de propósito geral ou de propósito específico) (FAPERGS, 2010).

A instanciação Orientação a Aspectos (OA) usou os resultados da ferramenta *Aspect Oriented Programming* (AOP) Jungle, a qual está sendo desenvolvida no contexto da dissertação de mestrado de Cristiano de Faveri, no Grupo de Linguagens de Programação e Bancos de Dados da Universidade Federal de Santa Maria (UFSM).

Este trabalho está organizado como segue: o Capítulo 2 mostra uma revisão de literatura dos assuntos relacionados ao tema desta dissertação, incluindo: processos de *software*, linguagem de programação e linguagens de consulta (trabalhos relacionados). O Capítulo 3 mostra a especificação e a modelagem do processo proposto. O Capítulo 4 mostra e discute as instanciações, bem como os resultados obtidos. Por fim, o Capítulo 5 relata as considerações finais, destaca as contribuições deste trabalho e descreve trabalhos futuros relacionados ao modelo do processo.

2 REVISÃO DE LITERATURA

Este capítulo aborda os principais conceitos estudados para o desenvolvimento deste trabalho. Ele está dividido em três seções. Inicialmente, a Seção 2.1 descreve conceitos sobre modelos processos de *software*. A seção 2.2 mostra alguns modelos prescritivos de *software*. Alguns conceitos importantes sobre modelagem de processos são mostrados na seção 2.3 e, os principais conceitos do *framework* usado neste trabalho, o *Open Unified Process* (OpenUp) referenciados na seção 2.4. Por último, na seção 2.5, algumas linguagens de busca para código relacionadas a este trabalho são mostradas.

Inicialmente foram estudados modelos de processos de *software* e linguagens de programação, foi escolhido como *framework* o Processo Unificado Aberto (OpenUp) para a especificação do processo. A escolha se embasou pelo fato de o OpenUp ser customizável, permitindo assim, sua utilização de acordo com as necessidades do processo proposto.

2.1 Modelos de Processos de *Software*

Processos de *software* são conjuntos de etapas, parcialmente ordenadas, com a finalidade de organizar, sistematizar o desenvolvimento de *software*, visando sim a obtenção de *software* de qualidade. Os modelos de processos de *software* adotam um ciclo de vida de sistemas, padronizam atividades, usam ferramentas e tratam da organização do trabalho que está sendo desenvolvido. Os modelos de processos de *software* também formalizam a distribuição e execução das tarefas entre os envolvidos fornecendo uma documentação completa, que auxiliará tanto no decorrer do desenvolvimento de *software* quanto em futuras manutenções. A maioria dos modelos de processos de *software* possui quatro atividades fundamentais: especificação, projeto e implementação, validação e evolução.

Os **modelos de processos prescritivos** tentam organizar, estruturar e documentar cada etapa da construção de um sistema de *software* (PRESSMAN, 2010). Em outras palavras, os modelos prescritivos prescrevem um conjunto de elementos e regras de como esses elementos se inter-relacionam e devem ser adaptados à equipe, ao domínio e ao projeto. Cada modelo sugere um fluxo de execução, porém evidencia genericamente as etapas de: comunicação, planejamento, modelagem e implementação.

A seguir serão descritos alguns modelos de processos de *software*, tais como: o modelo cascata, o modelo incremental, o modelo *Rapid Application Development* (RAD) e os modelos evolucionários (prototipação e modelo espiral).

O **modelo em cascata** possui seu foco no planejamento e na conclusão de cada etapa do processo sequencialmente (SOMMERVILLE, 2011). Cada estágio deve ser concluído e sua documentação aprovada antes de passar ao estágio seguinte. Seu uso é sugerido em situações nas quais os requisitos foram bem compreendidos e não venham a sofrer alterações significativas no decorrer do projeto.

A figura 1 ilustra o fluxo das atividades do processo, descritas a seguir:

- **Definição de requisitos**: consiste na especificação dos objetivos, das regras e das restrições definidas em conjunto com os usuários que servirão de base para o detalhamento das funcionalidades do sistema.
- **Projeto de sistema e de *software***: consiste na especificação de requisitos de *software* e de hardware (arquitetura geral do sistema).
- **Implementação e testes unitários**: são caracterizados pela codificação de unidades do projeto e, após, testadas de modo que atendam as especificações do projeto.
- **Integração e testes de sistema**: as unidades individuais de *software* são integradas (caracterizando um *software* completo) e testes são efetuados a fim de verificar se os objetivos do projeto foram alcançados. Com a aprovação, um sistema de *software* é entregue ao cliente.
- **Operação e manutenção**: caracteriza-se pelo uso de *software* pelo cliente. Consequentemente envolve a correção de erros e/ou a descoberta de novos requisitos.

O **modelo incremental** consiste em uma implementação inicial e em demonstração aos usuários. Através de *feedbacks*, novas versões são criadas e mostradas aos usuários até que uma versão que atenda às necessidades dos envolvidos no projeto seja produzida (SOMMERVILLE, 2011). A figura 2 mostra as etapas envolvidas na execução do processo: desenvolvimento e validação intercalados, criando assim várias versões e obtendo um *feedback* rápido.

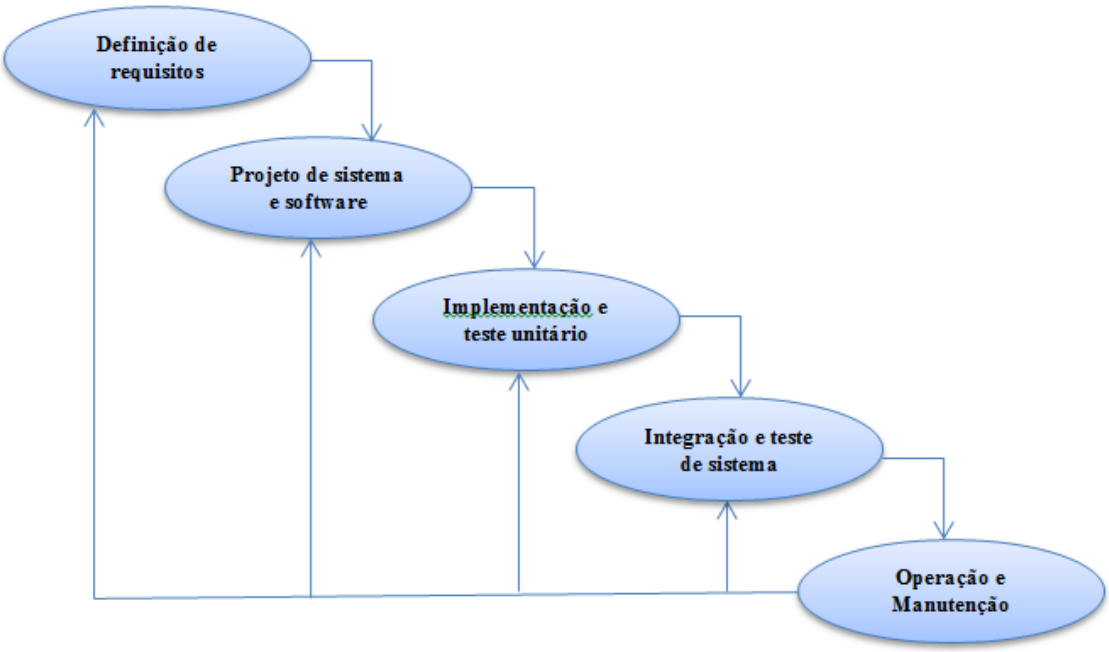


Figura 1 – Modelo em cascata.

Fonte: Sommerville (2011).

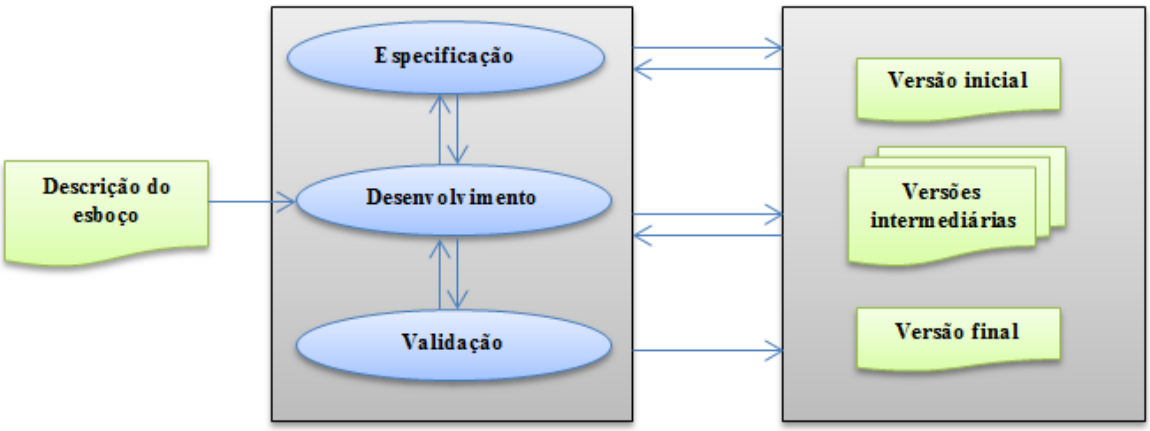


Figura 2 – Modelo Incremental.

Fonte: Sommerville (2011).

O **modelo RAD** (*Rapid Application Development*) é uma adaptação do modelo cascata, sendo que o desenvolvimento rápido é baseado na construção de componentes. Tal modelo segue as seguintes etapas (Figura 3):

- **Comunicação**: tenta entender as regras de negócio e suas características;
- **Planejamento**: trabalha em paralelo com várias equipes em diferentes funções de um sistema;

- **Modelagem**: sugere boas práticas para a modelagem de negócio, dados e processo;
- **Construção**: enfatiza o reuso de componentes preexistentes de *software*.
- **Implantação**: Estabelece a base para iterações necessárias.

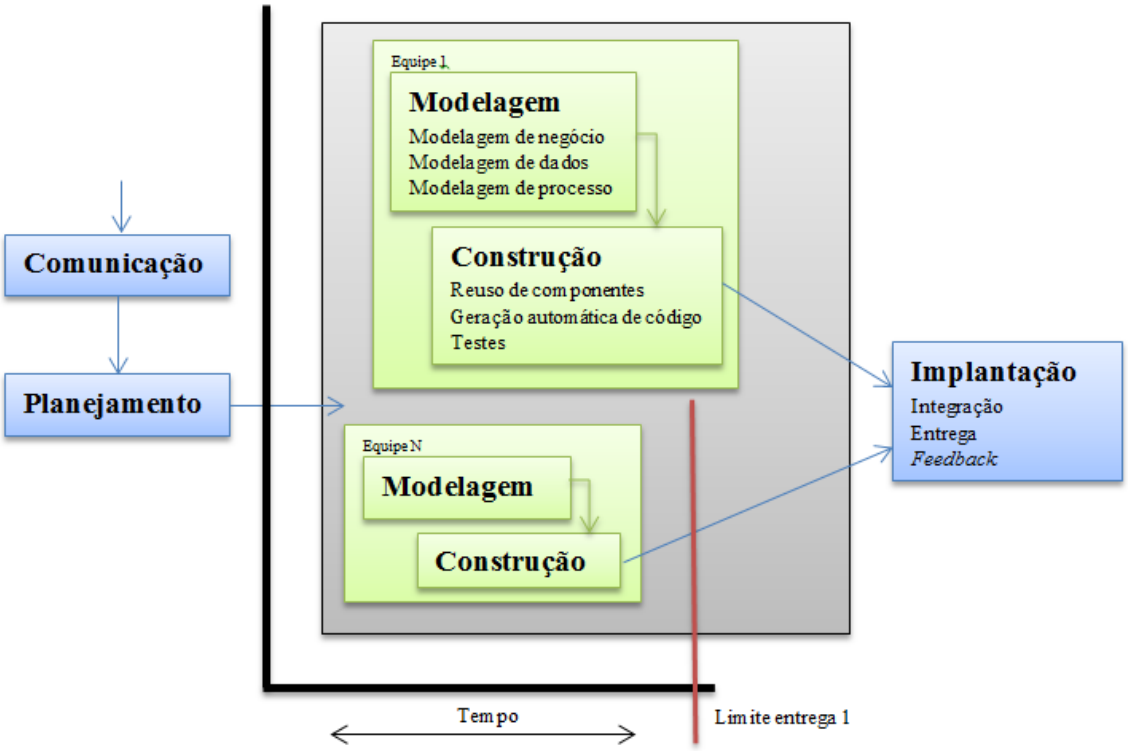


Figura 3 – Modelo RAD.

Fonte: Pressman (2010).

O **modelo de desenvolvimento evolucionário** é usado quando os requisitos de *software* mudam frequentemente e o tempo para finalização do projeto é curto, fazendo assim com que um planejamento adequado não possa ser seguido. Uma implementação inicial de *software* é feita tomando por base especificações mais abstratas e melhorado até que satisfaça as necessidades dos usuários. Cada melhoria é tratada como um incremento de *software*, fazendo com que os usuários passem a compreender melhor suas necessidades, contribuindo assim para uma nova versão de *software* (PRESSMAN, 2010).

Dentre os modelos evolucionários, mostrados a seguir, estão: a prototipação e o modelo espiral.

A **prototipação** não trata um protótipo de *software* como algo que possa ser desconsiderado após entendimento dos requisitos iniciais. O protótipo é usado como *software*

inicial que, através de incrementos, evoluirá até o sistema final. Tal modelo possui embasamento em requisitos já compreendidos, e não há uma documentação definida. Há uma interação constante com o usuário e o foco é fornecer entregas frequentes (incrementais) de *software* (PRESSMAN, 2010).

A figura 4 mostra a iteração do processo, que inicia com a comunicação com os envolvidos no projeto com a finalidade de definir o objetivo e a identificação dos principais requisitos de *software* e a modelagem de um projeto é feita. Na sequência, o protótipo é implementado e submetido à avaliação dos usuários, os quais fornecerão um *feedback* para melhoria do protótipo. As iterações ocorrerão até que o protótipo possa ser considerado como produto final ou, muitas vezes, considerado como *software* inicial.

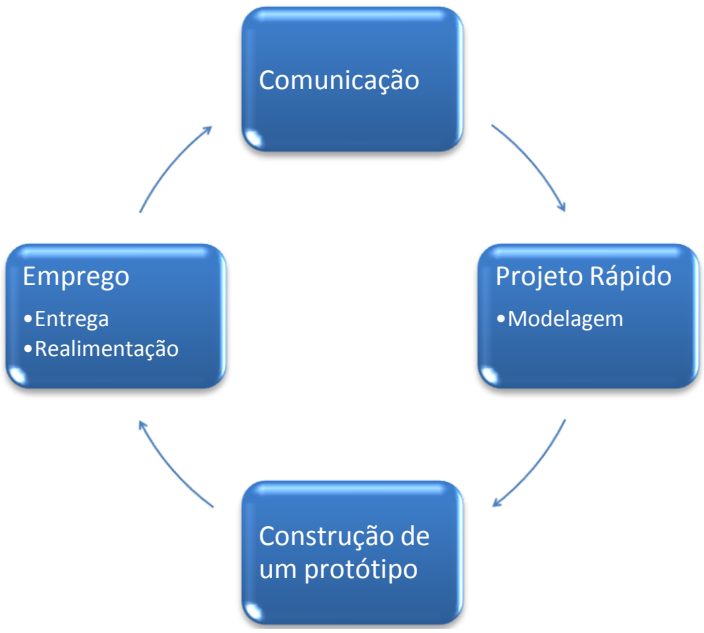


Figura 4 – Prototipação.

Fonte: Pressman (2010).

O **modelo espiral** engloba os conceitos da prototipação com controles do modelo em cascata, provendo diretrizes para a implementação rápida e completa de versões. Inicialmente pode ser um protótipo ou um modelo e, a cada iteração, se tornar mais completo.

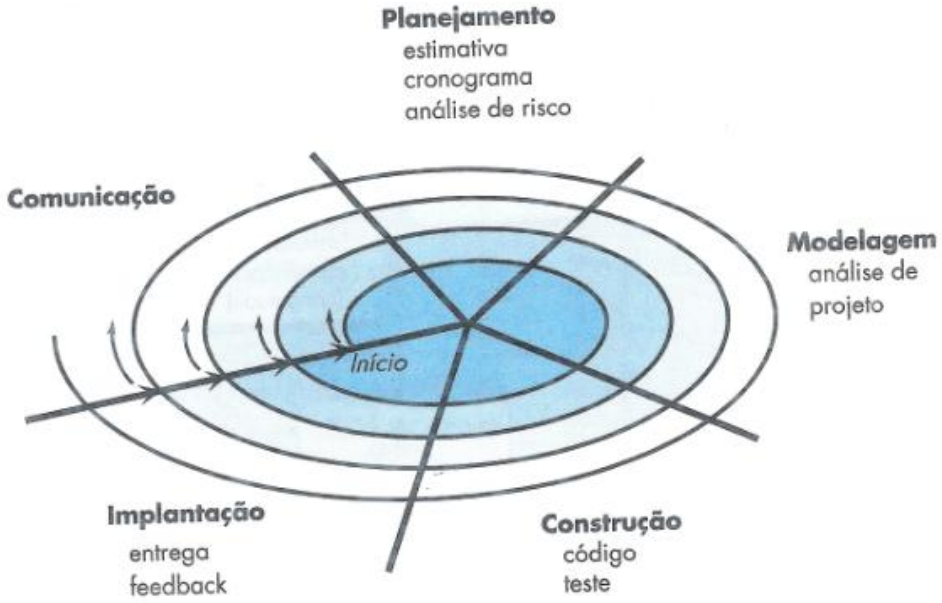


Figura 5 – Modelo Espiral.

Fonte: Pressman (2010).

A execução do processo (Figura **5**) tem início (primeira iteração) com a comunicação entre os envolvidos no projeto, podendo resultar na especificação de *software* e um protótipo. A cada iteração (passagem pela espiral) resultará em novas versões do protótipo e ajustes no planejamento do projeto de acordo com os *feedbacks* fornecidos pelos usuários.

2.2 Modelagem de Processos

A modelagem de processos visa auxiliar na identificação das áreas envolvidas no negócio, de todos os passos necessários para a execução de um processo, assim como a documentação utilizada. Com o conhecimento e documentação adquiridos, é possível detectar e propor mudanças nos processos atuais, atendendo às novas necessidades (ACUÑA et al., 2000).

Para que a modelagem seja precisa, é recomendado o uso de uma metodologia e de técnicas apropriadas. A metodologia deve permitir, principalmente, análise de requisitos, construção de modelos, análise do processo, documentação e melhorias contínuas.

A modelagem de processos pode ser associada ao ciclo *Plan-Do-Check-Action* (PDCA), como mostra a figura 6: entendimento do domínio de aplicação (Aprendizado), entendimento do processo (Entendimento), modelagem de processos (Documentação) e, finalmente, verificação de melhorias e manutenção (Melhoria) (JUBILEU, 2008).

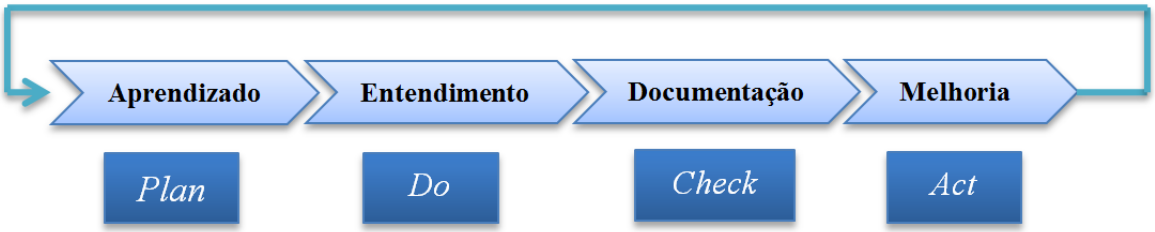


Figura 6 – Ciclo PDCA associado a processos de *software*.

Fonte: Adaptado de Valle e Barbará (2009).

2.2.1 SPEM

O *Software Process Engineering Metamodel* (SPEM) é um meta-modelo, proposto pela *Object Management Group* (OMG), para a especificação de processos e regras de desenvolvimento de *software* com um propósito de unificação das diferentes propostas de modelagem de processos (NUNES, 2008). O SPEM usa a UML como notação para modelar um conjunto de informações relacionadas a processos (GENVIGIR; SANT'ANNA; FILHO, 2003). O SPEM define os conceitos necessários para modelar, apresentar e gerenciar métodos e processos de *software*. Está estruturado em um meta-modelo por sete principais pacotes relacionados, conforme mostra a figura 7.

A seguir é mostrada uma breve descrição dos pacotes que compõem a estrutura do SPEM (NUNES, 2008):

- ***Core***: contém as classes e abstrações comuns para as classes de todos os demais pacotes do SPEM.
- ***Process Structure***: representa processos como partes estáticas e relações entre atividades.
- ***Process Behavior***: permite estender estruturas do meta-modelo do processo como estruturas comportamentais fornecendo ligações entre elas, porém não as controla.
- ***Managed Content***: os processos de *software* são, normalmente, representados na forma de modelos e diagramas, os quais devem ser traduzidos para linguagem natural (documentação) a fim de proporcionar o entendimento dos envolvidos.
- ***Method Content***: são bibliotecas de conteúdos reusáveis para a construção de bases de conhecimento, especificações de processos e desenvolvimento de projetos.

- ***Process With Methods***: define novas (ou redefine) estruturas para a integração dos conceitos de processos definidos pelos meta-modelos do pacote *Process Structure* com instâncias do pacote *Method Content*. É importante, pois define diferentes partes do processo, dentre as quais: ciclo de vida, papéis, tarefas, entre outras.
- ***Method Plugin***: introduz conceitos de projeto, gerenciamento, configuração, reusabilidade e repositórios.

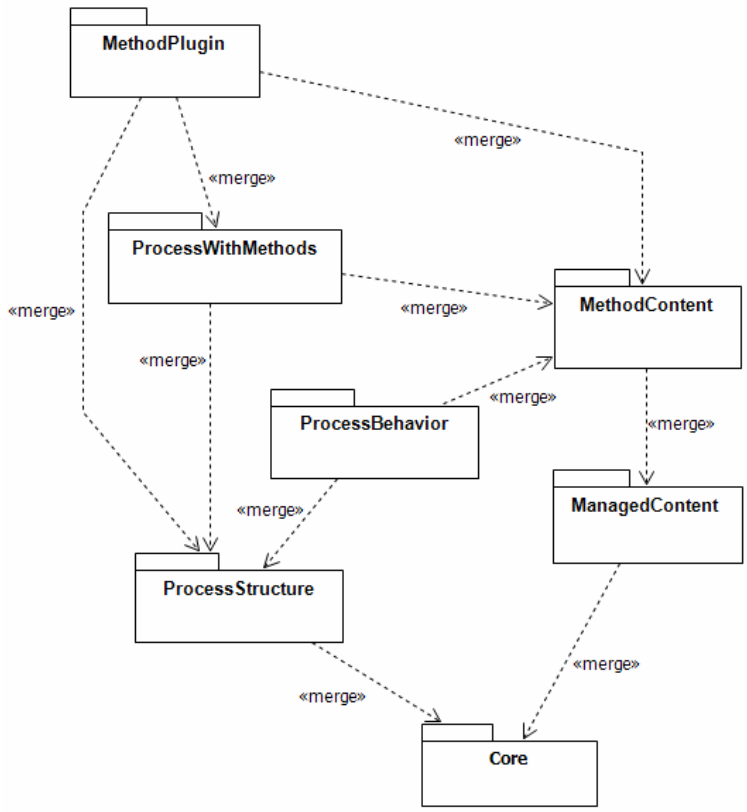


Figura 7 – Estrutura do SPEM.

Fonte: NUNES (2008).

A OMG divide a modelagem de processos em uma arquitetura de quatro camadas, conforme a figura 8 (NUNES, 2008):

- M0: É o processo como representado em um projeto (em execução).
- M1: Definição dos processos tanto genéricos quanto adaptados.
- M2: Neste nível é definida a linguagem para especificar modelos.
- M3: Instanciação do meta-modelo (nível mais alto), utilizando MOF[3].

[3] Linguagem de especificação de metamodelos.

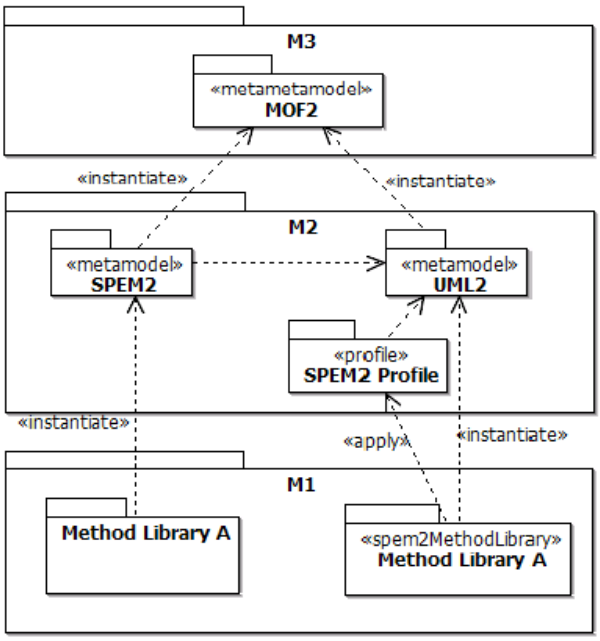


Figura 8 – Níveis de modelagem do SPEM.

Fonte: NUNES (2008).

Os processos são definidos através da configuração de elementos (Tabela 1) que descrevem ou restringem comportamentos e auxiliam no planejamento, execução e monitoramento. Basicamente o SPEM define que um processo de *software* é uma colaboração entre os papéis (entidades ativas), os quais realizam atividades (operações) em função de entidades concretas e tangíveis (produtos de trabalho).

Tabela 1 – Elementos do SPEM 2.0

Estereótipo	**Representação**	**Descrição**
Tarefa		Unidade de trabalho. Pode ser formada por diversos passos de execução.
Atividade		Corresponde ao agrupamento de tarefas.
Papel		Responsável por um produto de trabalho que é usado e produzido por uma atividade que é executada por um papel.
Produto de Trabalho		Corresponde a artefatos consumidos, produzidos ou transformados durante a execução de uma atividade.
Diretriz		Fornece informações adicionais descritas (ou anexadas), tais como relatórios, cronogramas, diagramas, entre outros.
Papéis Compostos		Descrevem habilidades, competências ou responsabilidades para um grupo de pessoas (equipe).
Passo		Especifica as etapas de execução de cada atividade.
Guia de Uso de Ferramenta		Tutorial de auxílio na aprendizagem (treinamento) do uso de ferramentas.
Pacote de Processos		Contém elementos de processo. Pode ser instanciado integral ou parcialmente para uso em outros processos.
Equipe		Agrupa papéis simples e/ou equipes.
Biblioteca de Métodos e Configurações		Container para métodos e configurações
Plugin		Representa um container físico de conteúdos e pacote de processos.
Capacidade do Processo		Pacote específico de processos que representa encapsulamento.
Fase		Representa um período significante no projeto, de maior gerenciamento de marcos e andamento do processo.
Processo de Entrega		Descreve parte do processo próxima de sua finalização ou conclusão do mesmo
Processo de Iteração		Descreve um conjunto de atividades reusáveis no decorrer do processo.
Problemas		Descrevem produtos cujos objetivos não foram alcançados.
Entregável		Representa o processo concluído.
Marco		Especifica o tempo final (esperado) para produção de algum artefato ou execução de alguma atividade.

Fonte: Adaptado de NUNES (2008).

A escolha do SPEM para este trabalho se deve ao fato da possibilidade de customização e seu uso simplificado através da ferramenta EPF Composer.

2.2.2 EPF

O *Eclipse Process Framework* (EPF), *framework* desenvolvido pela *Eclipse Foundation*, tem como principal finalidade estabelecer padrões para manter uma base de conhecimento de conteúdo sobre os processos definidos, a fim de facilitar sua especificação, gerenciamento e visualização (ECLIPSE FOUNDATION, 2009). O EPF tem como base a plataforma Eclipse e o metamodelo SPEM (NUNES, 2008) e contém seus elementos originais. A vantagem no uso do EPF está na flexibilidade, pois é possível utilizar elementos de outros *frameworks*/metodologias, tais como OpenUp (Anexo A), SCRUM e XP (ECLIPSE FOUNDATION, 2010).

A figura 9 ilustra a arquitetura do EPF. Neste trabalho foram foco de estudo apenas os itens 1 (Eclipse), 2 (Metamodelo baseado em SPEM), 3 (Ferramenta EPF Composer) e 4 (OpenUp).

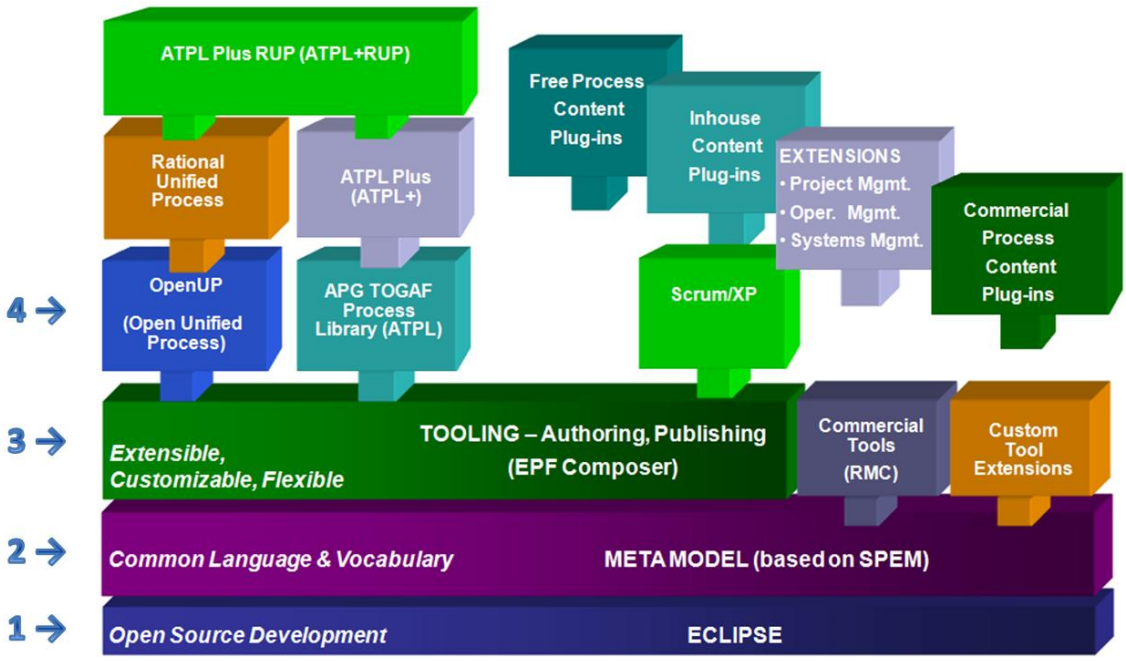


Figura 9 – Arquitetura do EPF.

Fonte: ECLIPSE FOUNDATION (2010).

Da mesma forma que outros *frameworks*, o EPF possui alguns elementos principais, visualizados na figura 10, a serem especificados, divididos em dois grandes grupos: conteúdo (*Method Content*) e processo.

Para a especificação de processos em EPF, existe uma ferramenta, o EPF Composer, a qual é indicada para pequenos e médios projetos, possibilitando análise da gestão do processo de *software* de maneira iterativa, ágil e incremental (ECLIPSE FOUNDATION, 2010).

A figura 10 mostra que o conteúdo do processo (lado esquerdo) se baseia na especificação de artefatos, papéis e tarefas. No lado direito, são mostrados os elementos do processo, onde o principal elemento é a atividade que pode ser aninhada de forma a reproduzir o fluxo de trabalho. Ao centro (cruzamento entre Conteúdo e Processo) encontram-se as diretrizes (podem ser roteiros, diagramas, exemplos, entre outros) contendo informações necessárias aos dois lados.

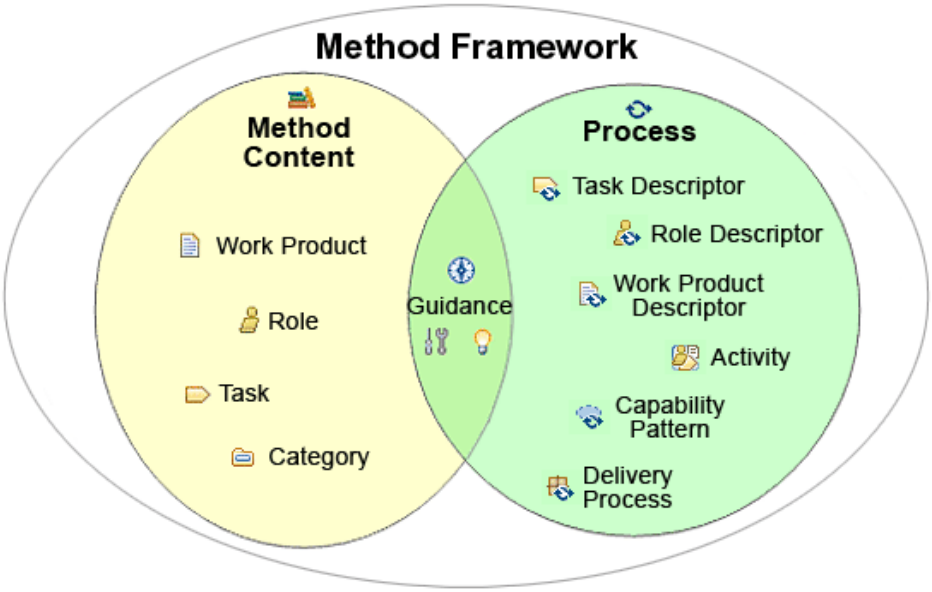


Figura 10 – Visão geral dos elementos do SPEM.

Fonte: ECLIPSE FOUNDATION (2010).

Através do EPF Composer, conteúdos referentes aos processos podem ser adicionados, sendo eles internos (criados através da própria ferramenta) ou externos (documentação, requisitos de *software*, diagramas, entre outros).

Utilizando o EPF Composer, podem-se criar processos de desenvolvimento de *software* padronizados, através de esquemas anteriormente especificados, sendo estes, uma evolução do *Software Process Engineering Metamodel* (SPEM) (NUNES, 2008).

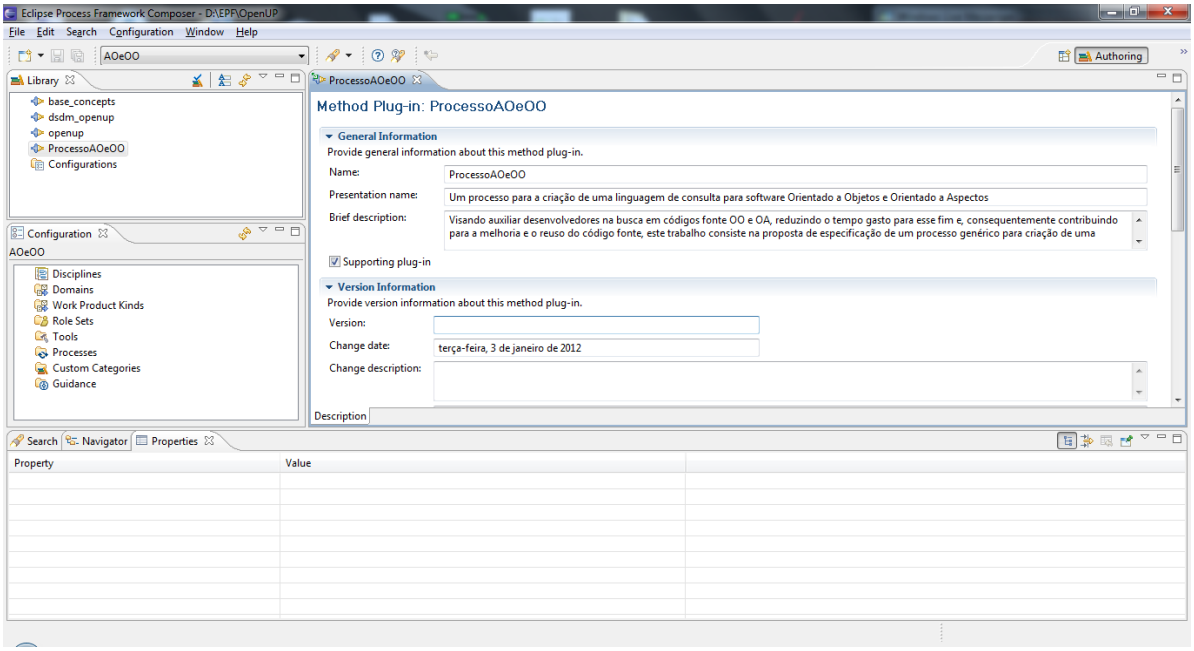


Figura 11 – Interface do EPF Composer.

2.3 JDT

A plataforma Eclipse[4] é uma das principais IDEs para o desenvolvimento em Java e uma das suas principais característica é o desenvolvimento baseado em *plugins* (ERICKSON, 2001). Dentre as vantagens do uso da plataforma Eclipse estão a reutilização de código e a flexibilidade de implementação, ou seja, podem ser criados e adicionados *plugins* para auxiliar no desenvolvimento de acordo com a necessidade.

O projeto *Java Develpment Tools* (JDT) consiste em um biblioteca de *plugins* que fornece à IDE do Eclipse a perspectiva de um projeto Java, juntamente com suas mais diversas ferramentas. Fazem parte da JDT quatro componentes principais (Figura 12):

- ***Core***: contém a infra-estrutura Java. Inclui compilador Java, modelo para navegação em árvore sintática de elementos Java (classes, métodos, interfaces, entre outros) e formatação de código.
- ***Annotation Processing Tool* (APT)**: trata-se de uma ferramenta para processamento de anotações
- ***Debug***: Implementa o suporte à depuração de programas Java.
- ***User Interface* (UI)**: implementa visões de usuário as quais contribuem para a manipulação do código Java.

[4] Eclipse Project: http://eclipse.org/

O componente JUnit não faz parte da JDT, porém é considerada a ligação do pacote DOM com a linguagem Java (ECLIPSE FOUNDATION, 2012).

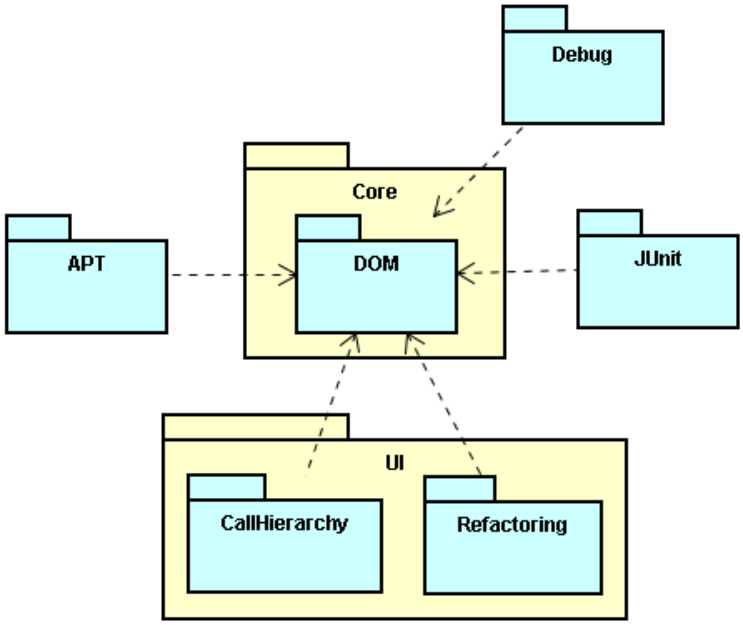


Figura 12 – Dependências internas da JDT.

Fonte: ECLIPSE FOUNDATION (2012).

2.4 Ontologias e Linguagem OWL

Uma ontologia é definida como uma especificação formal e explícita de uma conceituação compartilhada (GRUBER, 1993). As ontologias definem as regras que inferem os termos e as relações. As relações entre os termos são especificadas por especialistas e os usuários elaboram consultas utilizando os conceitos e termos especificados (GUARINO, 1998).

O vocabulário formado por predicados lógicos forma a rede conceitual que confere o caráter intencional às ontologias (GRUBER, 1993). Com isso, para formalizar a citada rede conceitual, cinco tipos de componentes são utilizados:

- **Classes**: conceitos do domínio ou tarefas, geralmente organizados de forma taxonômica.
- **Relações**: representam as conexões, vínculos entre conceitos pertencentes a um domínio.

- **Funções**: caso especial de relações onde o enésimo elemento é único para os vários elementos precedentes.
- **Axiomas**: são regras que declaram as relações que os elementos de uma ontologia devem cumprir. Inferem conhecimento que não estão indicados nas taxonomias de uma ontologia.
- **Instâncias**: utilizadas para representar objetos por meio de um conceito em um determinado ambiente ou domínio.

A *Web Ontology Language* (OWL) é uma linguagem de ontologias especialmente desenvolvida para descrever naturalmente classes e seus relacionamentos em aplicações web. A construção de uma ontologia OWL deve conter a descrição das classes, propriedades e suas instâncias. Para relacionar as classes com suas propriedades, ou atribuir outras informações, são utilizados axiomas, os quais contêm restrições e conjuntos de valores necessários e/ou permitidos (LIMA; CARVALHO, 2005). Um aspecto positivo da OWL é a possibilidade de utilizar *plugins* para raciocinar sobre determinada ontologia e seu domínio (*reasoners*), facilitando a construção das ontologias (SMITH; WELTY; MCGUINNESS, 2004).

2.5 Linguagens para Busca em Código

Linguagem para busca em código é uma linguagem de computador usada para realizar consultas em arquivo de código fonte, escrito em uma linguagem de programação definida, representada através de uma estrutura apropriada (FOWLER; PARSONS, 2010).

Existem diversas linguagens de consulta para códigos, dentre as quais destacam-se a JQuery, JTL, Lost, e outras que serão mostradas nas seções seguintes.

2.5.1 JQuery

JQuery é um plugin da plataforma Eclipse para a manipulação de código OO, usando um banco de dados a partir do código-fonte, fornecendo meios para consultar os dados constantes em tal banco (MCCORMICK; DE VOLDER, 2004). Após a execução da consulta, é apresentada uma hierarquia em formato árvore com nós e sub nós que representam as

classes e seus métodos. Porém, um problema de visão poluída (muitas informações dispostas de forma desorganizada) da consulta poderá ocorrer em consultas mais complexas (PFEIFER; SARDOS; GURD, 2005). Essas consultas são construídas através de predicados, tendo por base uma linguagem de programação lógica chamada TyRuBa (VOLDER, 1998), implementada em Java.

A figur 13 mostra o código de uma consulta, executada em JQuery, que busca todos os pacotes, cujas classes possuem um método que comece com “d”.

```
package(?P), child(?P,?CU), child(?CU,?M),
child(?M,?T), type(?T), name(?M,?name),
re_match(/^d/,?name)
```

Figura 13 – Exemplo de consulta utilizando JQuery.

Fonte: Pfeifer, Sardos e Gurd (2005).

2.5.2 Lost

Lost é um plugin da plataforma Eclipse que permite buscas em código OO e código OA e visa obter maior precisão na consulta para código e navegação nos resultados através de árvore (PFEIFER; SARDOS; GURD, 2005). A figura 14 mostra a mesma consulta da figura utilizando Lost:

```
select Package where Package.hasClass(Class)
&& Class.hasMethod(Method)
&& Method.hasName("^d")
```

Figura 14 – Exemplo de consulta OO utilizando Lost.

Fonte: Pfeifer, Sardos e Gurd (2005).

A proposta de Lost é utilizar os benefícios da JQuery, tais como navegador universal de código e consultas iterativas ao programa OA, utilizando AspectJ. Paralelamente, Lost tenta reduzir os aspectos negativos da JQuery, simplificando o processo de construção de consultas, provendo menor tempo para aprendizado da linguagem.

A figura 15 mostra uma consulta em Lost para buscar todos os métodos cujos nomes contenham “set” ou “put” e estejam sendo afetados por adendos do aspecto “formatChecker” porém, que não sejam afetados (via adendos) pelo aspecto “policyEnforcer”.

Figura 15 – Exemplo de consulta OA utilizando Lost.

Fonte: Pfeifer, Sardos e Gurd (2005).

A navegação nos resultados das consultas é similar à proporcionada por JQuery, ou seja, em formato de árvore.

A figura 16 mostra a interface do Lost: editor de código de consultas (1), resultado da consulta (apenas metadados) (2), propriedades, modificadores e referências dos elementos consultados (3) e apresentação de mensagens de erros de codificação (4).

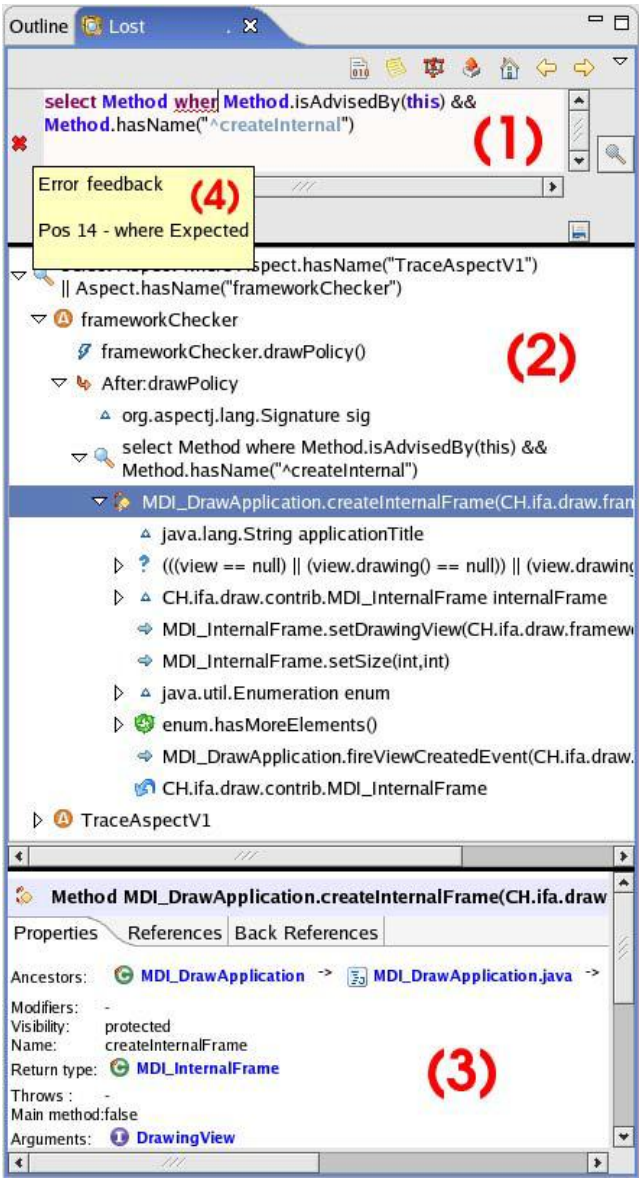


Figura 16 – Visualização do Lost.

Fonte: Pfeifer, Sardos e Gurd (2005).

Com os resultados são possíveis novas consultas referenciando nós e/ou sub nós na área de construção de consultas. Também são mostrados níveis refinados de consultas, tais como condicionais, laços de repetição e adendos.

2.5.3 JTL

A *Java Tools Language* (JTL) é uma linguagem cuja finalidade é efetuar buscas em trechos de código para Java e baseando-se em um banco de dados relacional como

representação (ao invés de uma AST). Ela pode ser utilizada tanto para aplicações OO quanto para aplicações OA (COHEN; YOSSI, 2006). Em particular, os autores citam o futuro uso da JTL em buscas por pontos de junção (POA), podendo se tornar uma alternativa a AspectJ, e outros elementos genéricos utilizados em códigos derivados da linguagem Java.

A figura 17 mostra um exemplo de consulta que verifica se uma classe é considerada "clássica", ou seja, é concreta, possui pelo menos um campo e alguns métodos de leitura e métodos de escrita distintos.

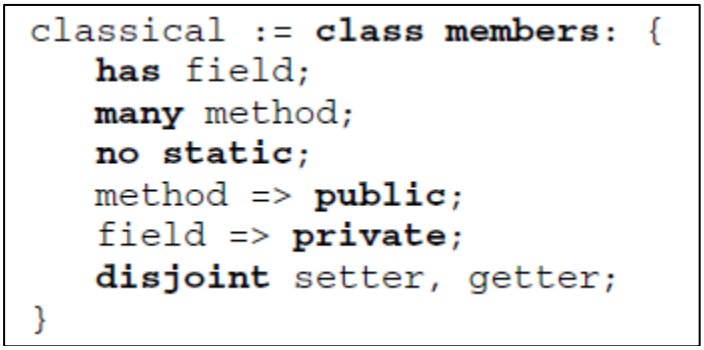


Figura 17 – Exemplo de consulta em classes utilizando JTL.

Fonte: Cohen e Yossi (2006).

Os autores compararam o uso da JQuery com a JTL em três consultas (Tabela **2**), a fim de mostrar a simplicidade do código da JTL.

Tabela 2 – Comparativo JQuery e JTL

Consulta	**JQuery**	**JTL**
Encontrar a classe BoardManager	class(?C,name,BoardManager)	**class** BoardManager
Encontrar todos os métodos "*main*"	method(?M,name,main) method(?M,modifier,[public,static])	**public static** main(*)
Encontrar todos os métodos, cujo parâmetro possua um tipo *image*	method(?M,paramType,?PT) method(?PT,/image/)	method(*,/?*image?*/,*)

Fonte: Cohen e Yossi (2006).

2.5.4 D-Cubed

D-Cubed é uma linguagem de consulta semântica para Java que captura as semânticas de um trecho de código já analisado. O D-Cubeb pode ser utilizado em três cenários: detecção de padrões de projeto com ênfase na semântica do código, descoberta de erros e falhas de segurança, e também na melhoria e avaliação na descoberta de más práticas de programação (WEGRZYNOWICKZ; STENCEL, 2009).

O D-Cubed possui diversos elementos, dentre eles um meta-modelo de instâncias simbólicas baseadas em grafos que, através da especificação de múltiplos parâmetros de consulta, possibilita uma busca lógica de ocorrências, tais como parâmetros de código, exceções e valores de retorno. Ele possui predicados em sua linguagem (por exemplo: *hasInput*, *hasOutput*, *isAssignedTo*, entre outros) e pode ser usado de duas maneiras: com bancos de dados relacionais (PostgreSQL ou MySQL) e linguagem SQL, ou com banco de dados dedutivos (XSB[5]) e linguagem Prolog.

2.5.5 SCML

Source Code Modeling Language (SCML) é uma linguagem de modelagem que utiliza *Computer Aided Constraint Verification* (CACV) para efetuar consultas (através de ASTs), cujos elementos de código fonte são representados graficamente por nós e a relação entre eles através de setas (CIRACI; BROEK; AKSIT, 2010). Para a confiabilidade da modelagem, inicialmente é feita uma verificação das restrições do código fonte em duas camadas (Figura 18). Na primeira camada, as violações de restrições e convenção de códigos são detectadas e um relatório de erro é preparado. A segunda camada fornece meios de pesquisa para as violações detectadas. O processo usa um repositório para armazenar e gerenciar restrições e convenções.

Primeiramente o desenvolvedor verifica (modifica ou insere) as regras de modelagem das restrições no repositório e, com isso, um *template* é gerado e os nomes dos elementos do programa são parametrizados. Com isso, o código fonte é convertido para um modelo SCML (Figura 19) e representado graficamente, inicialmente usado para representar código-fonte

[5] http://xsb.sourceforge.net/

escrito nas linguagens C, C++ e Java (CIRACI; BROEK; AKSIT, 2010). A SCML possui quatro tipos principais de declarações: sentenças de declaração, comentários, sentenças simples e expressões.

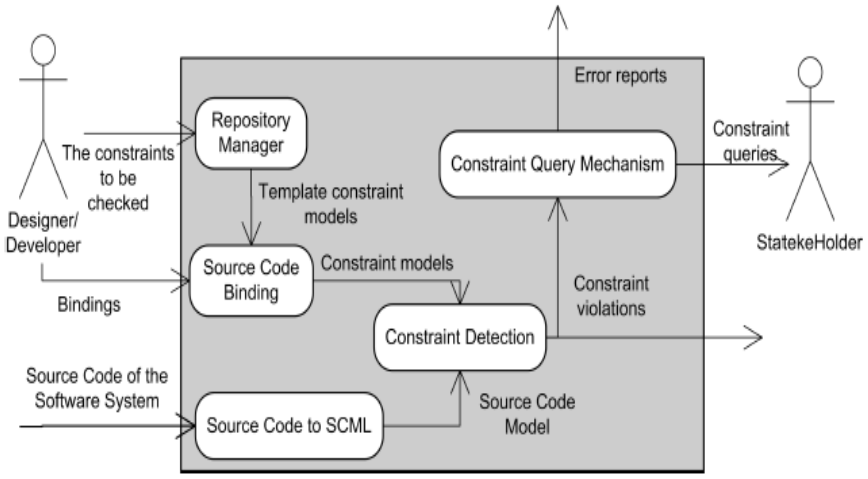


Figura 18 – Visão do CACV.

Fonte: Ciraci, Broek e Aksit (2010).

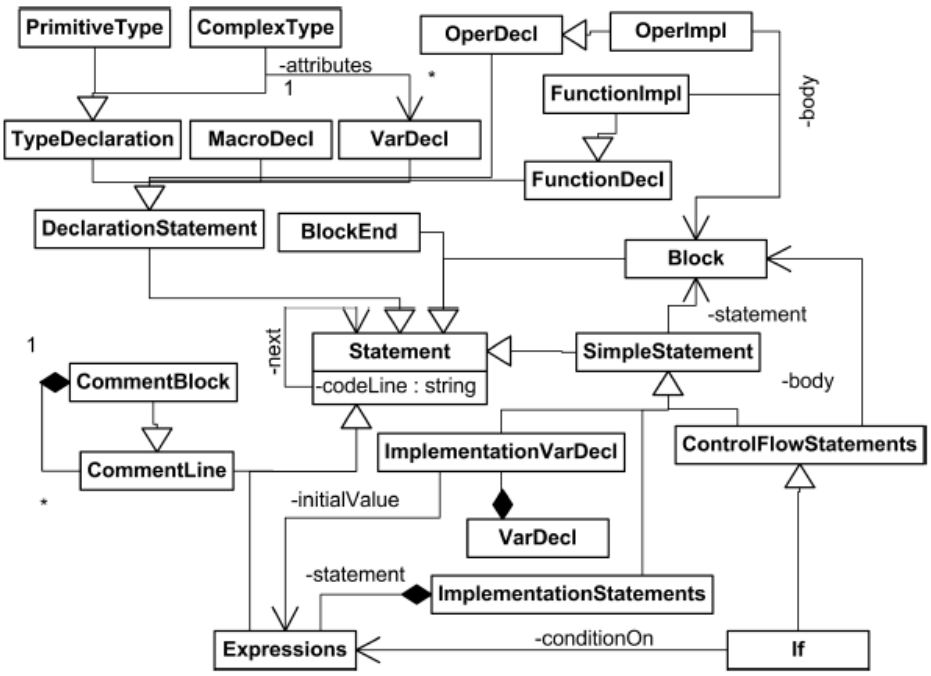


Figura 19 – Relações entre os tipos de declarações SCML.

Fonte: Ciraci, Broek e Aksit (2010).

2.6 Processos ODE

Com o objetivo de relacionar processo de *software* com ontologias, o trabalho dos autores (FALBO; BERTOLLO, 2005) mostram o mapeamento de processos, segundo a norma ISO/IEC 12207, em ontologias.

Ontology-based software Development Environment (ODE) é um ambiente de desenvolvimento de *software* baseado em ontologias e focado no domínio da engenharia de *software* e serve como padrão para um ambiente de domínio, incluindo ferramentas de definiçao de processos e de acompanhamento de projetos.

Esse trabalho mostra a decomposição de processos de *software* em bases de conhecimento e, comparando com os processos de *software* mais usados, criou níveis de abstração para a representação dos processos mais complexos (FALBO; BERTOLLO, 2005). A figura 20 mostra conceitos de processos de *software*, definidos pelos autores, aplicados juntamente com ontologias, resultando em um modelo conceitual.

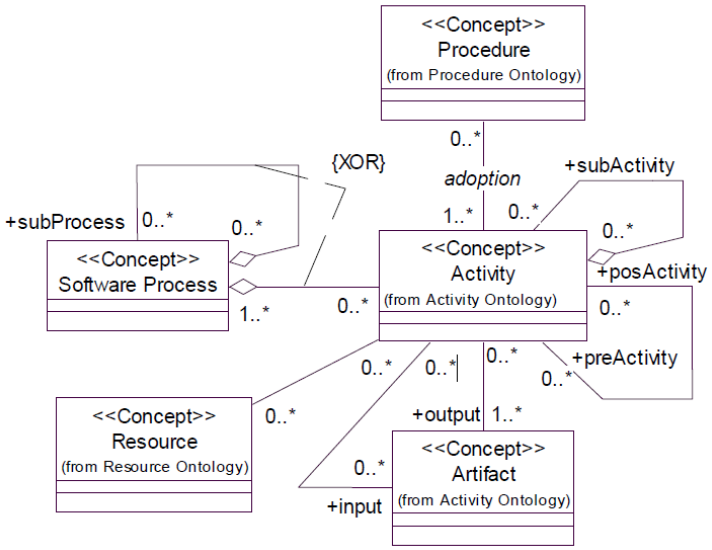


Figura 20 – Definição do Processo.

Fonte: Falbo e Bertollo (2005).

Conforme os autores (FALBO; BERTOLLO, 2005), os processos pode ser detalhados e melhorados a partir de um modelo base. A figura 21 ilustra os níveis de abstração do processo para uma organização que utiliza ISO/IEC 12207:

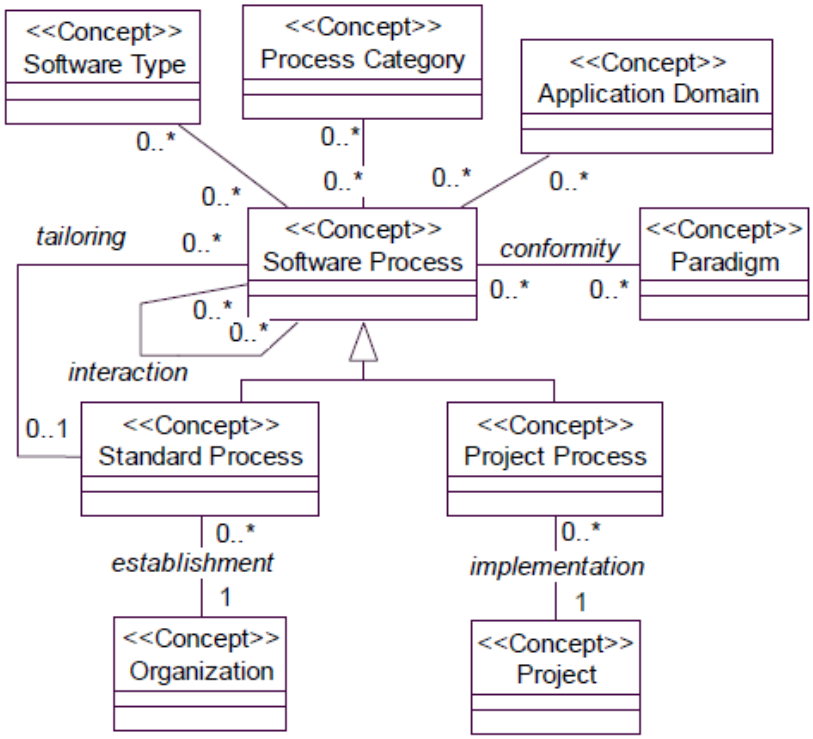


Figura 21 – Tipos de Processos e Níveis de Abstração.

Fonte: Falbo e Bertollo (2005).

O ciclo de vida do processo é tratado em um modelo em separado como mostra a figura 22.

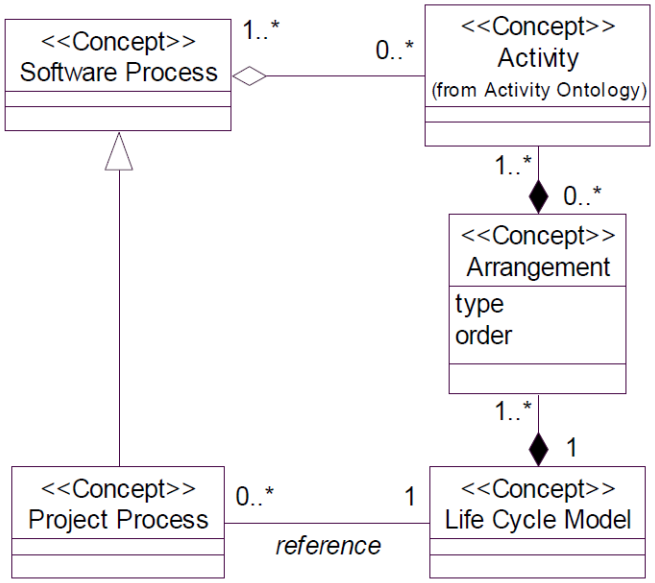


Figura 22 - Processo de Projeto e Modelos de Ciclo de Vida.

Fonte: Falbo e Bertollo (2005).

Uma das maneiras que o autor usou para a comprovação de seu modelo, pode ser visualizado na tabela 3, comparando com os elementos propostos pela ISO:

Tabela 3 – ISO x Ontologia de Processo de *Software*.

ISO	***Software* Process Ontology**
Process	*Software* Process
Standard Process	Standard Process
Tailored Process	Project Process
Work Product	Artifact
Activity / Task	Activity
Process Category	Process Category
Process *Software*	Process
Life Cycle Model	Life Cycle Model

Fonte: Falbo e Bertollo (2005).

Após a implantação experimental da ODE, diversas oportunidades de melhorias foram apontadas e, com isso, a versão inicial da ontologia foi melhorada em um trabalho posterior (BERTOLLO; SEGRINI; FALBO, 2006).

A figura 23 mostra o modelo anterior (Figura 21) com algumas melhorias em destaque, tais como: domínio de aplicação, ciclo de vida referenciado pelo projeto e cada processo associado a uma organização.

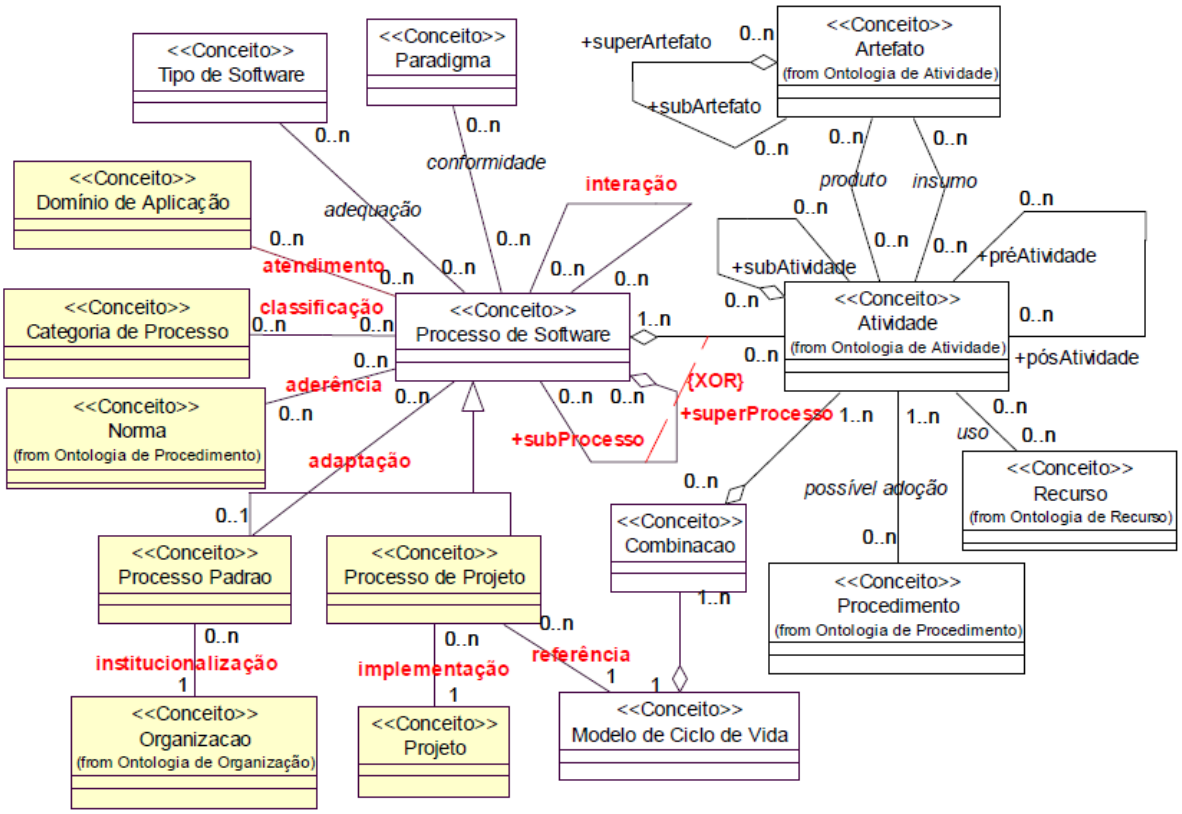


Figura 23 – Nova Versão da Ontologia de Processos de *Software*.

Fonte: Bertollo, Segrini e Falbo (2006)

A definição da nova ontologia permite que processos possam ser decompostos em sub processos ou atividades, podendo ter iterações com outros processos. Os demais conceitos se mantiveram (BERTOLLO; SEGRINI; FALBO, 2006).

3 UM PROCESSO PARA O USO DE LINGUAGENS DE CONSULTA EM CÓDIGO FONTE

Visando auxiliar o desenvolvedor na busca em códigos fonte Orientação a Objetos (OO) e Orientação a Aspectos (OA), este trabalho consiste na especificação de um processo genérico para criação ou para o uso de linguagens de consulta para a busca em código-fonte, reduzindo assim o tempo gasto para esse fim e, consequentemente contribuindo para a melhoria e o reuso de código fonte.

Com o auxílio da ferramenta EPF Composer (descrita brevemente na seção 2.3.3) foi especificado o processo e os diagramas gerados pela própria ferramenta. Basicamente foram utilizados alguns elementos do SPEM presentes no OpenUp, cujas descrições encontram-se no capítulo 2.3.1 (Tabela1). Dentre os elementos do SPEM usados estão:

- Biblioteca de métodos e configurações.
- Artefatos necessários e/ou produzidos.
- Tarefas executadas.
- Fase significativa.
- Funções desempenhadas por pessoas (papéis).
- Visão do processo.

Este capítulo descreverá o processo proposto, suas atividades, papéis e artefatos necessários e produzidos durante a execução de cada atividade.

O processo consiste de duas etapas: Definição e Instanciação. O processo inicia na etapa de **Definição**, na qual são selecionados os elementos a serem mapeados e modelos de representação, e finaliza com a etapa de **Instanciação**, na qual é criada ou usada uma ferramenta de extração de metadados e consultas (podendo utilizar uma linguagem pré-definida, tal como SQL e XQuery) para a obtenção dos resultados desejados.

Como requisitos para o processo podem ser citados:

- Possuir um repositório de código;
- Saber quais os resultados são necessários serem alcançados;
- Ter conhecimento da existência de ferramentas para o propósito (talvez seja necessária à implementação de uma ferramenta) e
- Estar em conformidade com o SPEM.

A figura 24 mostra o fluxo das atividades do processo que ocorre em cada fase.

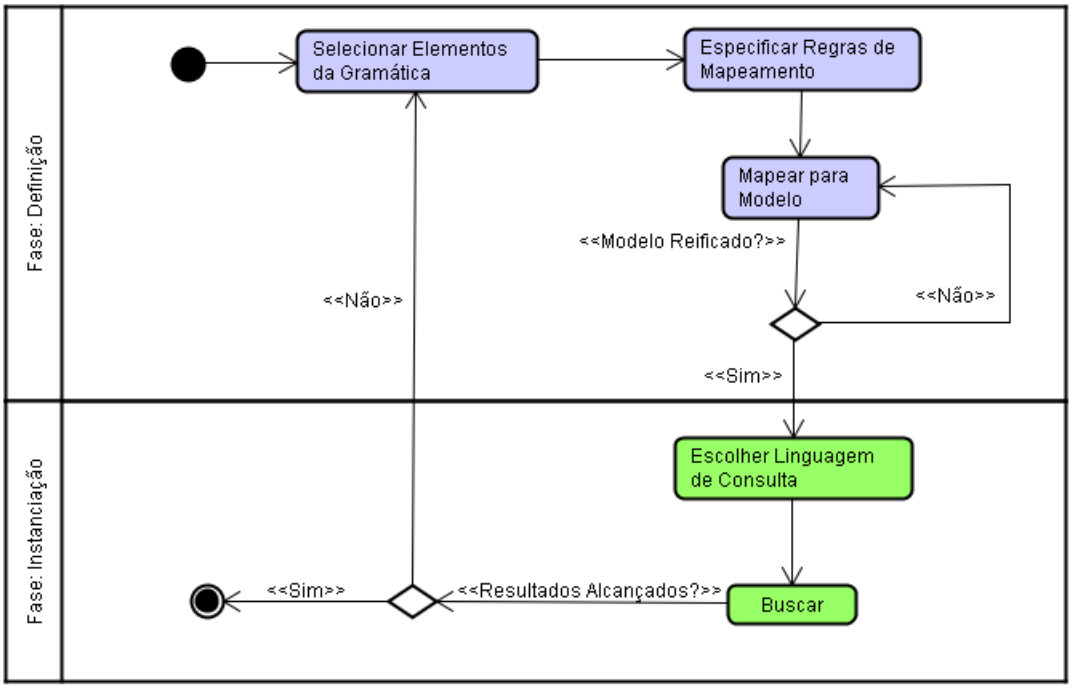


Figura 24 – Diagrama de atividades do processo.

Na etapa **Definição** são descritas as atividades: **Seleção dos Elementos da Gramática**, **Especificar Regras de Mapeamento** e **Mapear para Modelo.** E na etapa **Instanciação**, são especificadas as atividades: **Escolher Linguagem de Consulta** e **Buscar**.

A figura 25 mostra a visão geral do processo, enfatizando o executor das tarefas e os artefatos consumidos e produzidos durante e após a execução de cada tarefa.

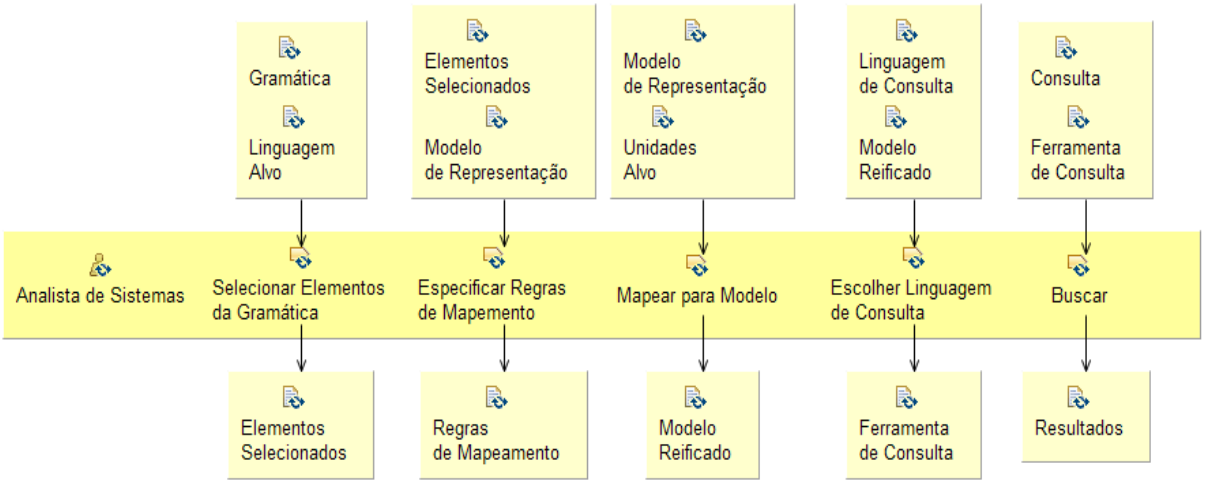


Figura 25 – Visão detalhada do processo.

3. 1Papéis

Os papéis indicam quem executará determinada tarefa e assumirá a responsabilidade pela análise dos artefatos de entrada e a pela integridade dos artefatos produzidos. Um papel pode se referir a um ou mais atores (pessoas) com as mesmas competências e habilidades. Para este trabalho, foram utilizados dois papéis: **Analista** e **Programador**.

O Analista é responsável pela análise e especificação de todo o processo de busca, tarefas e integridade dos artefatos. O Programador é o responsável pela codificação das ferramentas e pelas consultas.

3.2 Etapa de Definição

Esta primeira etapa tem como foco a escolha dos elementos gramaticais de uma linguagem de programação definida pelo analista de sistemas, até a extração dos metadados mapeando-os para o modelo reificado, o qual será codificado de acordo com as especificações contidas no Modelo de Representação.

3.2.1 Seleção dos elementos da gramática

A atividade inicial (Figura 26), possui dois artefatos de entrada: Gramática e Linguagem Alvo, na qual são selecionados elementos da gramática da linguagem de programação.

- **Linguagem Alvo**: Este artefato especifica qual linguagem de programação será utilizada, tal como: Java, C, C++, C#, dentre outras. Pode ser considerada uma especificação menos formal com a finalidade de “nortear” o processo.
- **Gramática**: As linguagens de programação são criadas a partir da definição de gramáticas. As gramáticas são contituídas por um conjunto de sentenças, as quais são formadas através de regras e símbolos terminais e não terminais (SEBESTA, 2011). A finalidade da utilização de uma gramática como artefato de entrada do processo

proposto, é que através dela, pode-se verificar qual caminho percorrer (através de árvores sintáticas) para encontrar determinados elementos em código-fonte.

Como resultado da execução desta atividade, é obtido o artefato de saída: Elementos Selecionados.

- **Elementos Selecionados**: são definidos os elementos gramaticais que proporcionarão a busca em trechos de código-fonte desta mesma linguagem. Cada elemento da gramática será selecionado de acordo com sua estrutura e necessidade de busca, servindo de referência para a extração de metadados. Por exemplo, caso o usuário queira buscar trechos de código em programas escritos em uma linguagem orientada a objetos, seria interessante que ele fosse capaz de se referir a classes, atributos ou métodos.

Esta atividade é dependente de um objetivo específico e de conhecimento da linguagem de programação a ser pesquisada.

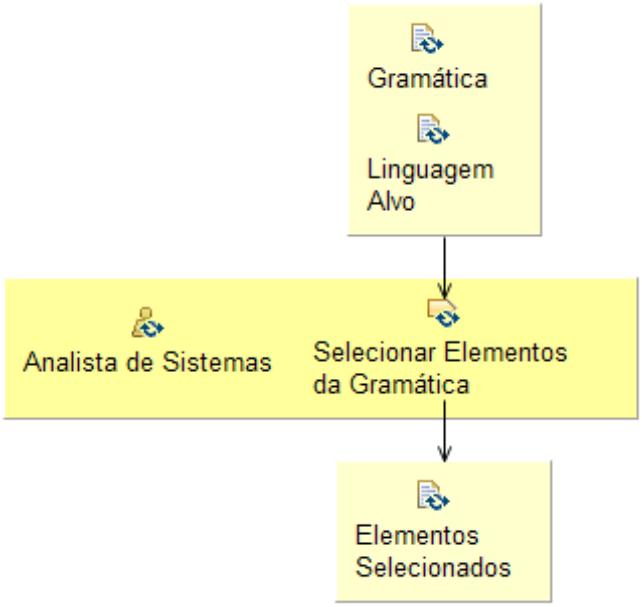


Figura 26 – Seleção de elementos da gramática da linguagem de programação escolhida.

3.2.2 Especificar as regras de mapeamento

A criação de regras de mapeamento é necessária para que seja possível definir **como** os elementos do código-fonte serão obtidos e armazenados.

Para esta atividade (Figura 27), é necessário o artefato gerado na atividade anterior, **Elementos Selecionados** e, também deter o conhecimento de como os metadados serão disponibilizados:

- **Modelo de Representação**: é definido por um **Analista de Sistemas** e será utilizado para a organização e o armazenamento dos metadados durante a extração dos mesmos. Dependendo do propósito da instanciação do processo, o modelo de representação pode ser um modelo relacional de banco de dados, um arquivo XML, uma ontologia, dentre outros.

  A execução desta atividade, resulta em regras que serão usadas para a extração de metadados e sua representação. Como artefato de saída são obtidas **Regras de Mapeamento**.

- **Regras de Mapeamento**: este artefato mostra como os elementos selecionados sao mapeados para o modelo de representacao quando encontrados em programas, tais como classes com seus modificadores, atributos, métodos dentre outros elementos. Como exemplo pode ser citado: um Aspect, na gramática, passa para o formato <aspect> em linguagem XML.

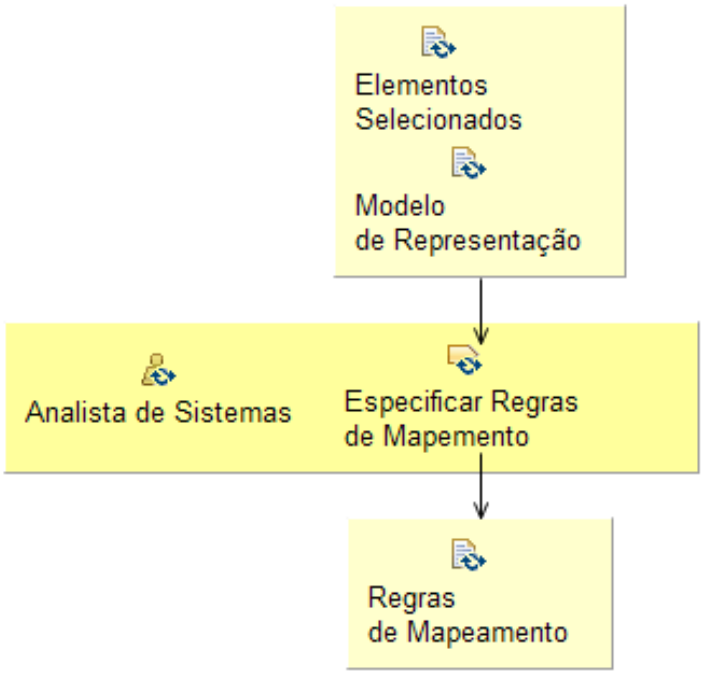


Figura 27 – Especificação das Regras de Mapeamento.

3.2.3 Mapeamento

Com a especificação de regras para mapear e extrair metadados é feita a seleção das unidades alvo que serão mapeadas (processo de reificação[6]) para o modelo reificado. Além do artefato Regras de Mapeamento, esta tarefa também utiliza o artefato Unidade Alvo. Após a execução desta tarefa, o artefato Modelo Reificado é obtido.

- **Unidades Alvo**: representa o arquivo que contém o código fonte a ser lido e/ou um repositório de vários arquivos (pasta ou espaços de trabalho) que seguem a mesma gramática. Sendo, portanto, a base da qual o processo extrairá os metadados.
- **Modelo Reificado**: caracteriza-se por tornar informações sobre a estrutura de um programa (metadados) disponíveis no modelo reificado e pronto para consultas (Figura 28).

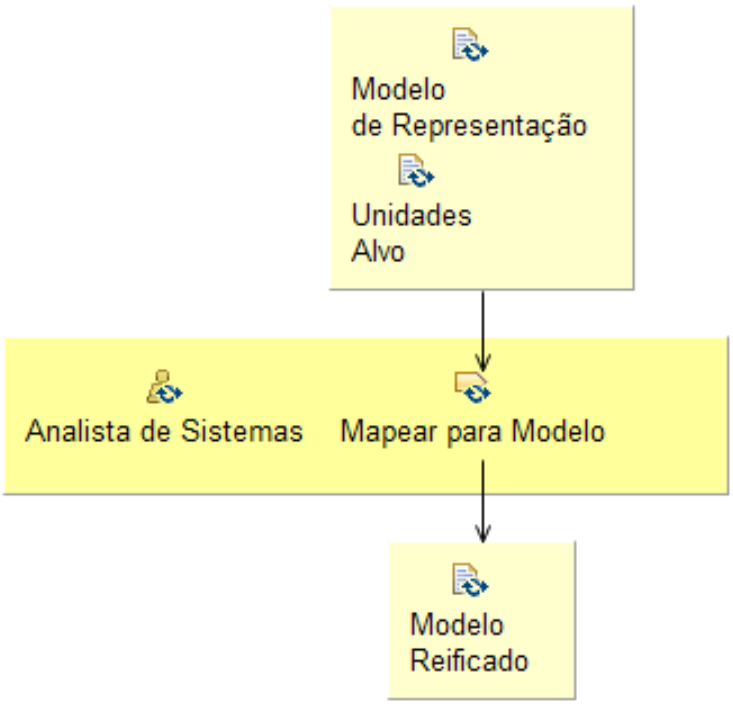


Figura 28 – Mapeamento e Reificação.

[6] A reificação caracteriza-se na transformação dos elementos do código fonte em informações estruturais, os metadados (classes, aspectos, métodos, atributos, entre outros), a serem disponibilizadas para a utilização de diversas formas.

A reificação pode ser feita com a implementação de uma ferramenta focada nos objetivos a serem alcançados, através da especificação de requisitos. A Tabela 4 mostra alguns possíveis requisitos que poderiam ser utilizados na construção de uma ferramenta de auxílio ao processo.

Tabela 4 – Especificação de requisitos de uma ferramenta para busca em código fonte.

Requisitos Funcionais		
Identificador	Nome	Descrição
RF01	Selecionar Unidades Alvo	Implementar uma classe (ou interface) para a escolha de Unidades Alvo.
RF02	Implementar Regras de Mapeamento	Implementar (ou estender) métodos que atendam às Regras de Mapeamento.
RF03	Definir Linguagem de Consulta	Escolher ou criar uma linguagem de consulta que atenda aos objetivos do processo.
RF04	Efetuar buscas	Implementar buscas na linguagem de consulta escolhida, de modo que forneçam resultados que possam ser analisados claramente.
Requisitos Não Funcionais		
Identificador	Nome	Descrição
RNF01	Escolher sistema operacional	Definir o sistema operacional cuja aplicação será executada.
RNF02	Escolher linguagem de programação	Definir a linguagem de programação para a implementação da aplicação.
RNF03	Preparar forma de armazenamento de informações	Preparar o Modelo de Representação a ser usado no armazenamento das informações.

3.3 Etapa de Instanciação

A segunda etapa é constituída pela escolha da linguagem de consulta, que pode ser uma linguagem de consulta existente, tal como SQL, XQuery, SPARQL, ou outra linguagem implementada durante o processo especificamente para a busca em uma determinada linguagem ou tipo de artefato, até mesmo, a uma ferramenta na qual são feitas consultas para a obtenção dos resultados que possam auxiliar na melhoria ou reuso do código.

3.3.1 Escolha da linguagem de consulta

Com base no Modelo Reificado, pode-se ter opções de escolha de Linguagens de Consulta existentes dependendo da especificação das regras e dos modelos. Com a escolha de

uma linguagem de consulta, podem ser utilizadas ferramentas prontas ou implementadas exclusivamente para o objetivo das consultas, obtendo o artefato Ferramenta de Consulta (Figura 29).

- **Linguagens de Consulta**: refere-se à escolha de uma linguagem de consulta já existente, tais como SQL, XQuery, SPARQL, entre outras, ou a criação de uma nova linguagem específica para a consulta, como por exemplo, uma linguagem específica de domínio (DSL) (Anexo B). Tal escolha poderá ser utilizada em uma ferramenta (implementada para o propósito) ou usada diretamente.
- **Ferramenta de Consulta**: é o mecanismo de consultas nos metadados extraídos. Pode ser a implementação de uma ferramenta apropriada com uso de linguagens de consultas existentes (ou criadas para domínios específicos).

Para auxiliar nesta atividade, podem ser usadas heurísticas as quais são definidas como técnicas que tendem a melhorar o desempenho em tarefas de solução de problemas e geralmente é uma tentativa de aproximação do conhecimento (RUSSEL; NORVIG, 2002).

A escolha da linguagem de consulta varia de acordo com os objetivos a serem alcançados e o modelo de representação escolhido. Podem ser utilizadas linguagens de consultas já existentes ou, caso nenhuma delas proporcione o resultado esperado, desenvolver uma DSL.

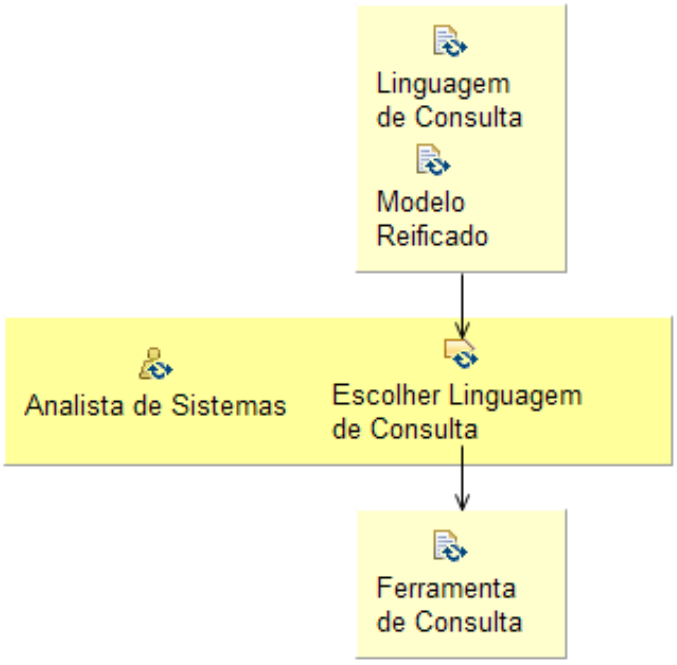


Figura 29 – Escolha da Ferramenta de Consulta.

3.3.2 Busca

Ao final do processo tem-se a atividade de busca e, como artefato de saída, os Resultados (Figura 30). Através da análise dos resultados poderão ser encontrados erros no código fonte ou oportunidades de refatoração, por exemplo.

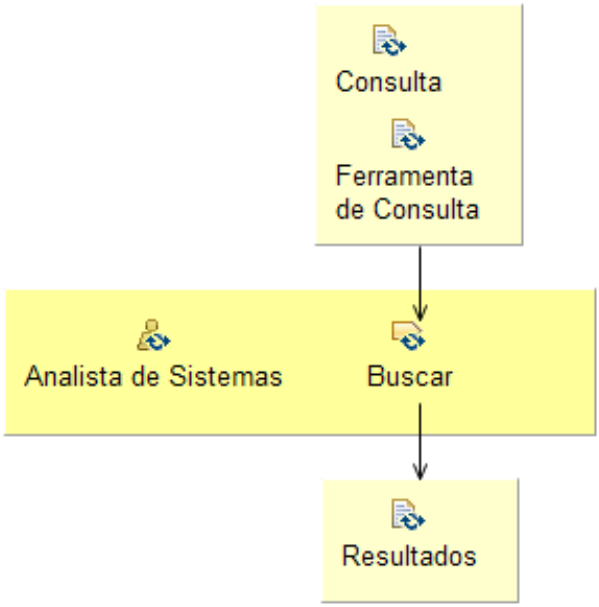


Figura 30 – Atividade de Busca nos metadados extraídos.

3.4 Metamodelo do processo

Para uma melhor visão dos artefatos envolvidos no processo, a figura 31 mostra o metamodelo do mesmo. As classes representadas no metamodelo coorespondem aos artefatos usados e/ou produzidos durante o processo. Com exceção das classes em destaque que foram adaptadas da gramática da linguagem Java.

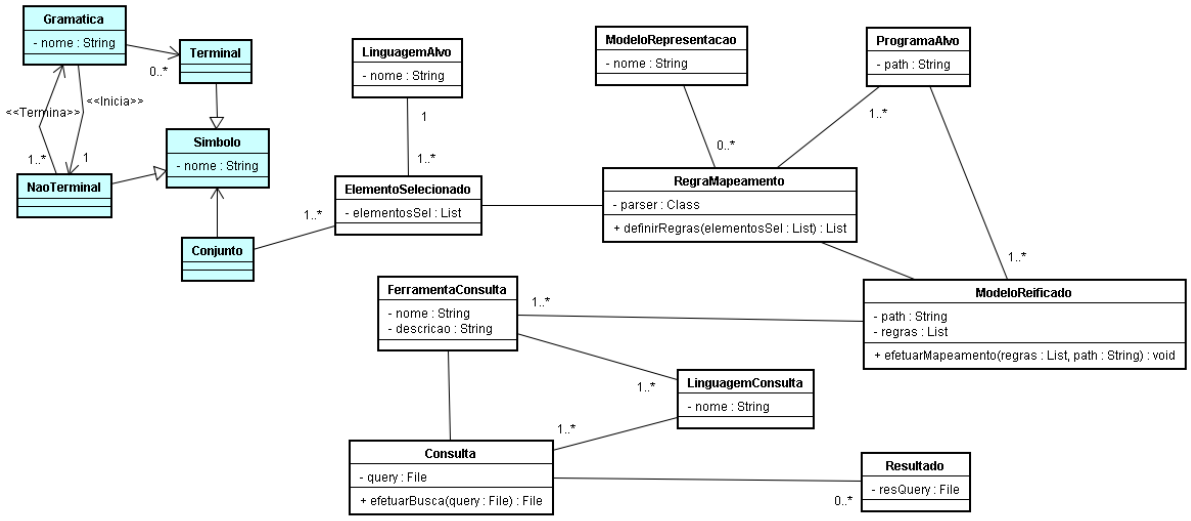


Figura 31 – Metamodelo do processo.

O objetivo deste metamodelo é prover uma visão de estrutura genérica do processo, podendo ser customizável de acordo com o tipo de busca e tecnologias envolvidas.

4 INSTANCIAÇÃO DO PROCESSO

Para avaliar a aplicabilidade do processo proposto, foram realizadas três instanciações distintas sob a ótica de dois paradigmas diferentes linguagens de programação, diferentes modelos de representação e diferentes linguagens de consulta. A primeira delas para o paradigma Orientação a Objetos (OO), utiliza o modelo relacional e linguagem de consulta *Structured Query Language* (SQL). A segunda instanciação também para o paradigma OO, usa ontologias (linguagem OWL) e linguagem de consultas SPARQL. E a terceira instanciação, para o paradigma OA, utiliza linguagem XML para a representação e a linguagem de consultas XQuery.

4.1 Instanciação OO

Para as duas instanciações OO (usando como modelos de representação respectivamente, o modelo relacional e ontologias), foram utilizados: a linguagem Java (na plataforma Eclipse), o analisador sintático contido na JDT (de forma que seu uso em unidades de compilação Java resulta em *Abstract Sintax Trees* (ASTs), sendo uma para cada unidade) e a *Application Programming Interface* (API) Jena.

Em relação ao uso de ASTs, foram usadas três classes, conforme a figura 32, para a identificação de elementos presentes no código fonte:

- a classe **TypeDeclaration** responsável pela identificação de classes e interfaces do código.
- a classe **FieldDeclaration** usada para identificar os atributos e anotações de cada classe presente no código.
- a classe **MethodDeclaration** usada para identificar os métodos de cada classe e exceções (de cada método) contidos no código.

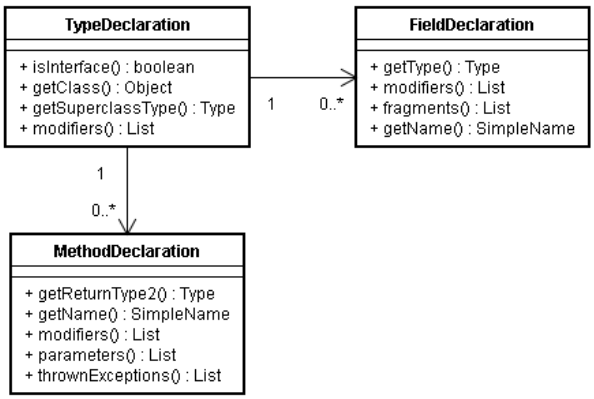


Figura 32 – Classes e Métodos da AST utilizados na instanciação OO.

Visando facilitar a extração de metadados, foi desenvolvida uma ferramenta, denominada **OOPExtract**, para efetuar a leitura de código Java contidos em uma pasta e, com a análise das ASTs geradas, inserir os metadados em instâncias nos dois **Modelos de Representação**: relacional e ontologias.

A ferramenta OOPExtract faz a análise de um conjunto de arquivos (contidos em um diretório especificado estaticamente) por execução. A cada execução, a ferramenta efetua a busca pelos elementos (Tabela 5) e insere as informações no modelo relacional (novos registros). Após a obtenção do valor do campo chave primária recém inserido, a ferramenta insere um novo indivíduo na ontologia criada.

Na ferramenta OopExtract, foram criados quatro pacotes (identificados pela funcionalidades principais) contendo suas classes funcionais. A figura 33 representa a dependência entre os pacotes implementados e as associações de suas classes internas.

O pacote principal (Figura 34) possui três classes responsáveis pelas instanciações das demais classes do programa, sendo a classe **OOPExtractExe** responsável por sua execução. A interface **OntModel**, pertencente à API Jena, é instanciada no início da execução do programa a fim de evitar que a ontologia seja sobrescrita. Já a classe **Conexao** teve sua instanciação no início do programa, como forma de evitar a criação de múltiplas conexões ao banco de dados, o que poderia impedir a execução completa da ferramenta.

Tabela 5 – Elementos selecionados para a instanciação OO

Elemento	**Descrição**	**Sintaxe Analizada**
Classe ou Interface	A AST considera uma classe ou interface como um *TypeDeclaration*. Com isso podem ser extraídos diversos elementos que, sintaticamente, a compõe.	**modificador** "**class**" ["**interface**"] nome_classe ["**extends**" nome_superclasse] ["**implements**" nome_interface] "**{**" Sendo: • **modificador**: conjunto de um ou mais modificadores (ex:. public abstract). • **extends**: Usado caso a classe seja a especialização de outra. • **implements**: Usado caso a classe implemente uma interface.
Atributo	São denominados *FieldDeclaration*s, contidos dentro de um *TypeDeclaration*. Abrangem variáveis e anotações.	**modificador** "**tipo**" nome_atributo"**;**" Sendo: • **modificador**: conjunto de um ou mais modificadores (ex:. public abstract). • **tipo**: tipo de dado da variável (ex.: int, String, tipos abstratos, entre outros).
Anotação	As anotações também são detectadas pela AST como um *FieldDeclaration*. Porém caracterizam-se pelo uso do identificador "@". Para este trabalho, foram considerados apenas os nomes das anotações.	**@**nome_anotacao Exemplo: @Produces("application/xml")).
Método	São detectados no interior de um *TypeDeclaration*, através de um *MethodDeclaration*.	**modificador** "**tipo_retorno**" nome_metodo "**(**" [**parametros**] "**)**" Sendo: • **modificador**: conjunto de um ou mais modificadores (ex:. public abstract). • **tipo_retorno**: tipo de dado da variável (ex.: int, String, tipos abstratos, entre outros). • **parâmetros**: lista de parâmetros necessários ao método. A ausência de parâmetros é caracterizada pelo uso do tipo de dado **void**.
Exceção	São identificadas dentro de um *MethodDeclaration*. Para este trabalho foram apenas consideradas as exceções declaradas juntamente com o método através do comando **throws**.	Declaração de método seguida do comando **throws** e lista de um ou mais tipos de exceções. Exemplo: **throws** ServletException, IOException.

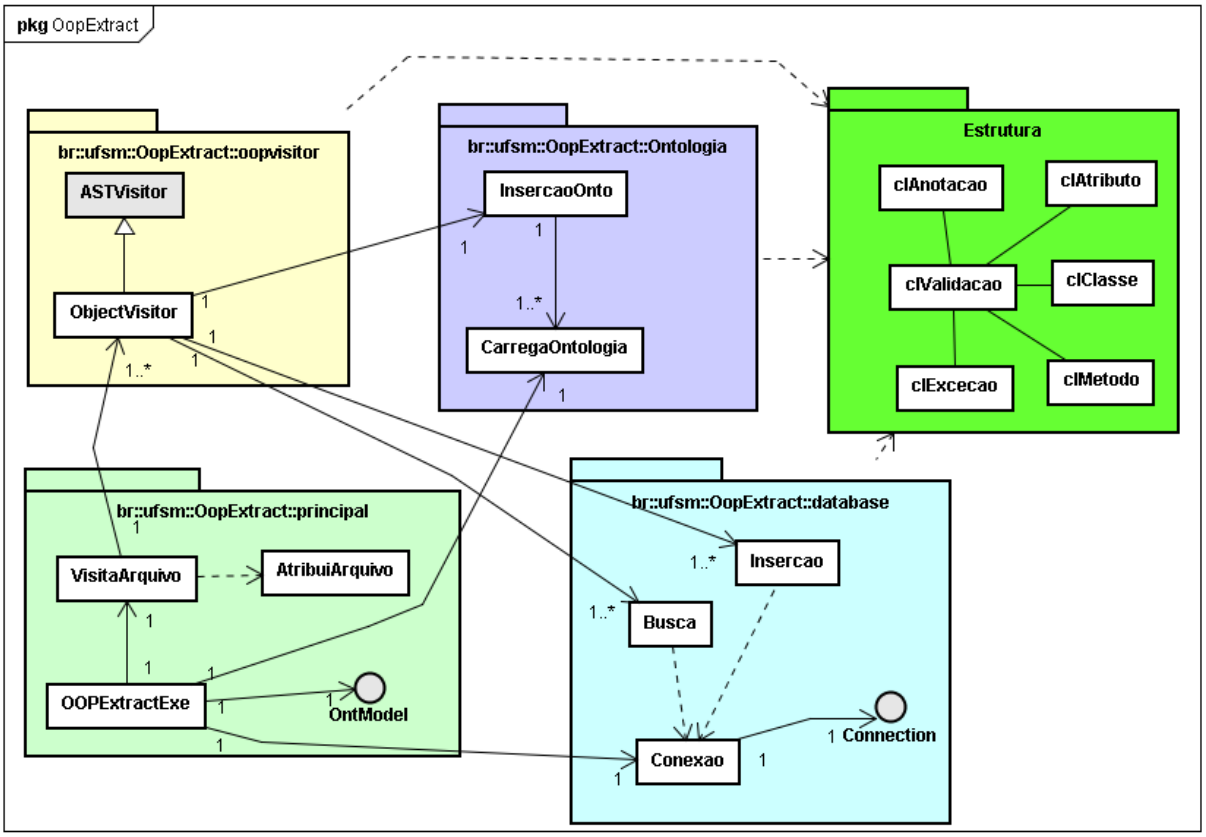


Figura 33 – Diagrama de Pacotes da ferramenta OopExtract.

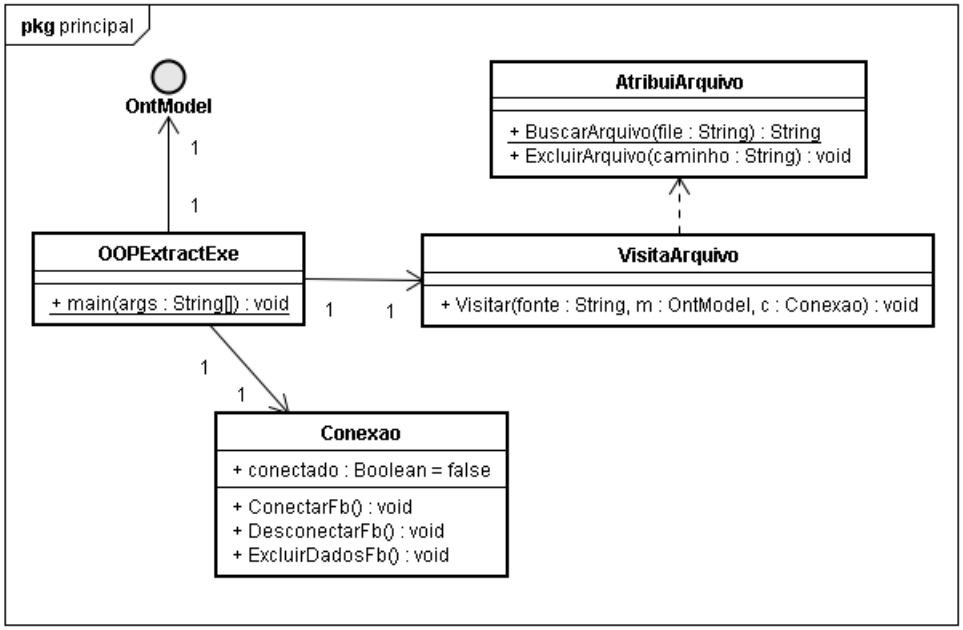


Figura 34 – Diagrama de Classes do Pacote principal.

Com o objetivo de efetuar a conexão com o banco de dados relacional, foi criado o pacote **database** (Figura 35). O pacote **database** também é responsável pelas inserções e buscas de chaves primárias em registros recentemente inseridos.

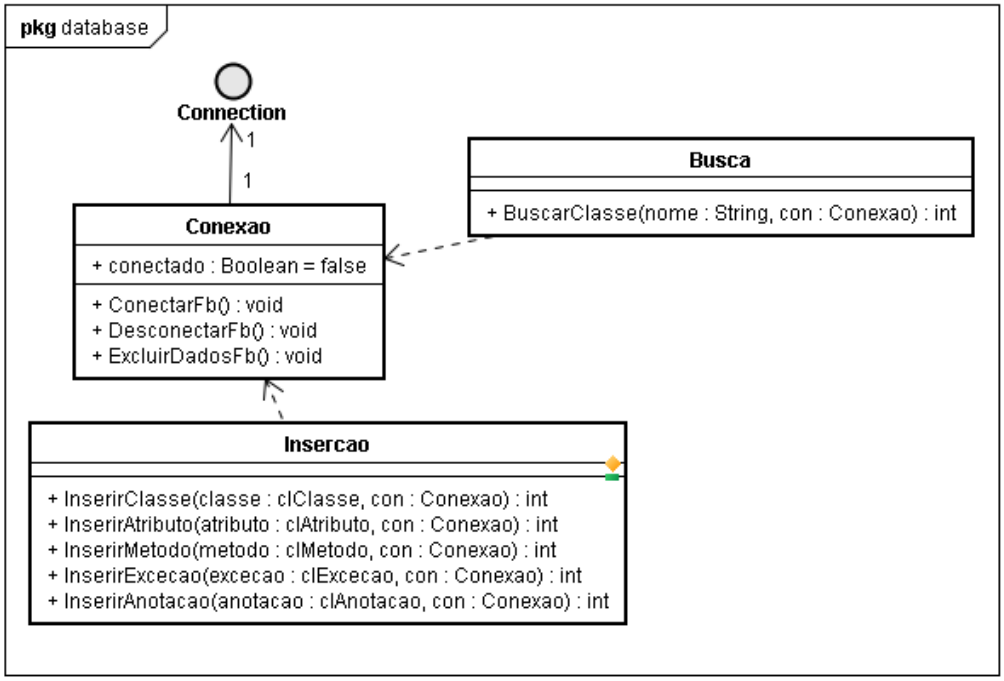


Figura 35 – Diagrama de Classes do pacote database.

Para popular instâncias da ontologia proposta, foi implementado o pacote **Ontologia** (Figura 36) que, por sua vez, possui classes para instanciar o modelo físico, efetuar inserções de indivíduos e atualizar o modelo a cada operação.

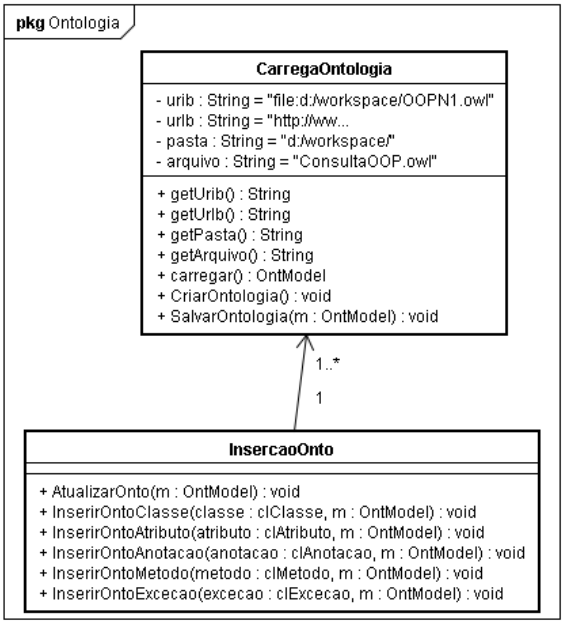


Figura 36 – Diagrama de Classes do pacote Ontologia.

O pacote **oopvisitor** (Figura 37) possui a classe **ObjectVisitor** a qual é estendida da classe **ASTVisitor** (contida na JDT). Foram implementados métodos polimórficos para atender aos objetivos das instanciações do pocesso proposto, com finalidade de percorrer arquivos contendo código escrito em linguagem Java e extrair os elementos definidos e suas características.

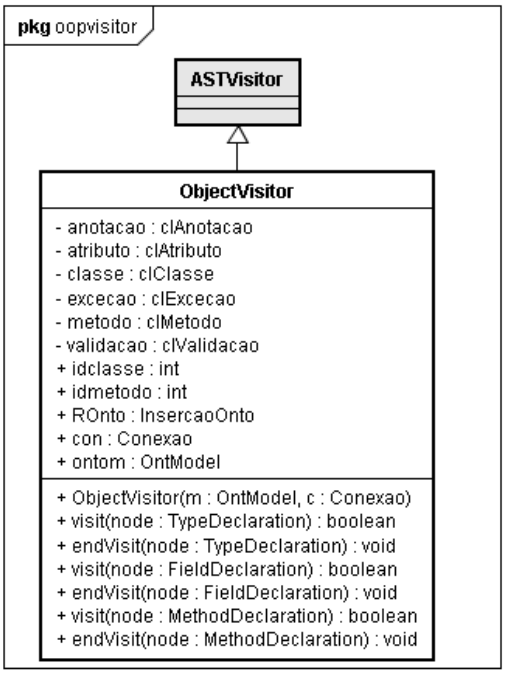


Figura 37 – Diagrama de Classes do pacote oopvisitor.

Os métodos de extração dos metadados descritos na classe **ObjectVisitor** (que estende a classe **ASTVisitor**), são sobrescritos e funcionam como um ciclo (Figura 38), visitando uma classe (ou interface), seus atributos (e anotações), logo após seus métodos (com suas exceções) e repetindo o ciclo (caso seja o final da lista de arquivos).

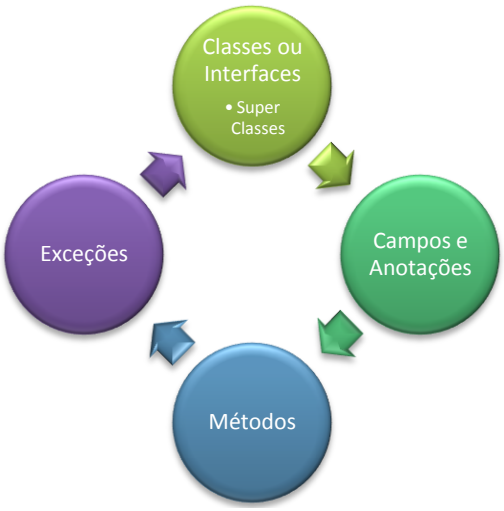


Figura 38 - Ciclo de extração dos metadados.

O fluxo da execução da ferramenta pode ser mais bem visualizado, através de um diagrama de atividades, na figura 39 (tarefas simples, tais como validações e atualizações da ontologia, não foram mostradas no diagrama).

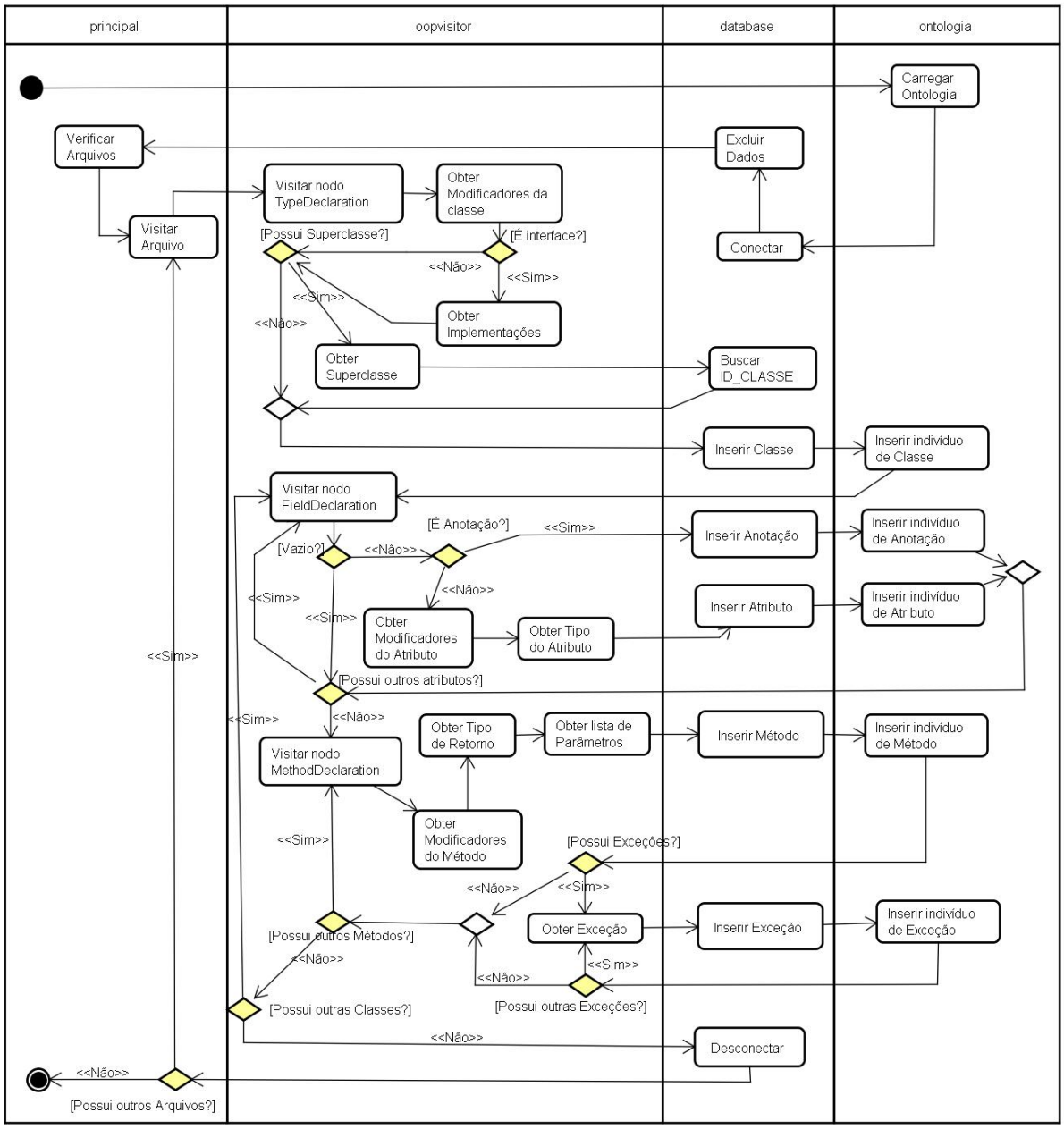


Figura 39 – Diagrama de Atividades da ferramenta OopExtract.

4.2 Instanciação OO, Modelo Relacional e SQL

Para a instanciação OO utilizando Modelo Relacional e SQL, foi utilizado o Sistema de Gerenciamento de Banco de Dados (SGBG) Firebird[7]. A Figura 40 mostra a estrutura do banco de dados implementada.

[7] Sistema de Gerenciamento de Banco de Dados *open-source*. Disponível em: http://ibphoenix.com/.

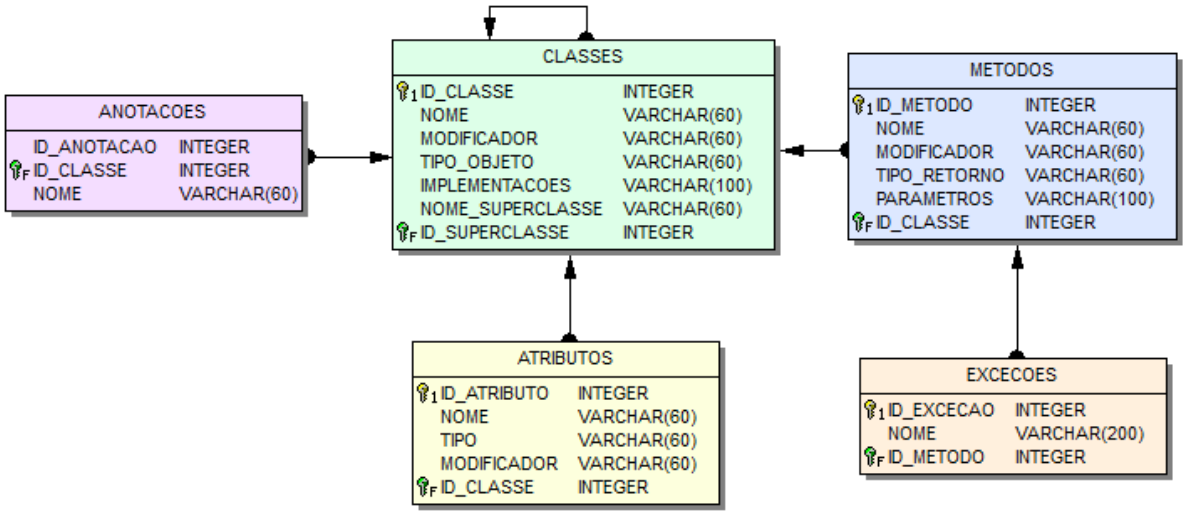


Figura 40 – Diagrama Entidade e Relacionamento da Instanciação OO.

Após popular o banco de dados (através da execução da ferramenta OopExtract em um conjunto de classes escolhidas aleatoriamente), foram elaboradas algumas consultas utilizando *Stored Procedures*[8]. Sendo estas, no caso do SGBD Firebird, usadas para executar blocos de comandos ou retornando valores (um ou mais registros mono ou multivalorados).

O código referente a consulta apresentada na figura 41 tem como resultado, uma listagem dos métodos extraídos, suas respectivas classes (através de junção) e exceções. Essas, por sua vez, são apresentadas na forma de campos, devido à verificação e à concatenação de seus nomes. O valor do parâmetro de verificação pode ser completo (uso apenas do caractere “%”), aproximado (contendo uma parte do nome precedido e/ou antecedido por “%”) ou exato (sem o uso do caractere “%”). Também pode-se utilizar o caractere “?” em substituição de um outro caractere.

[8] Procedimentos armazenados no banco de dados. Utiliza a linguagem de consultas SQL juntamente com linguagem procedural (P/SQL).

```
SET TERM ^ ;

CREATE PROCEDURE STP_CONSULTA_METODOS (
    PNOME varchar(60))
returns (
    MODIFICADOR varchar(60),
    METODO varchar(60),
    TIPO_RETORNO varchar(60),
    PARAMETROS varchar(100),
    EXECOES varchar(500),
    CLASSE varchar(60))
as
declare variable NOME_E varchar(200);
declare variable ID_METODO integer;
begin
  for
    select m.NOME,m.MODIFICADOR,m.TIPO_RETORNO,m.PARAMETROS,m.ID_METODO,c.NOME
    from METODOS m
    inner join CLASSES c on m.ID_CLASSE=c.ID_CLASSE
    where m.NOME like :PNOME order by m.NOME
    into :METODO,:MODIFICADOR,:TIPO_RETORNO,:PARAMETROS,:ID_METODO,:CLASSE
  do
  begin
       EXECOES='';
       for
        select first 10 e.NOME from EXCECOES e
        where e.ID_METODO=:ID_METODO into :NOME_E
       do
       begin
            if (:EXECOES='') then EXECOES=:NOME_E;
            else EXECOES=' '||:EXECOES||:NOME_E;
       end
       suspend;
  end
end^

SET TERM ; ^
```

Figura 41 – Código da consulta aos métodos extraídos.

A Figura 42 apresenta o resultado da execução da consulta tendo “%ar%” o valor do parâmetro.

MODIFICADOR	METODO	TIPO_RETORNO	PARAMETROS	CLASSE	EXCECOES
public	CarregarProfessores	void		Banco	
public	ListarUsuarios	ArrayList<Usuario>		UsuarioDao	
public	Selecionar	ResultSet	String tabela	Access	ClassNotFoundException
public	setComparator	void	ViewerComparator comparator	AbstractTableComposite	
public	start	void	IReportContent report	CSVReportEmitter	
public	startLabel	void	ILabelContent label	CSVReportEmitter	BirtException
public	startPage	void	IPageContent page	CSVReportEmitter	BirtException
public	startRow	void	IRowContent row	CSVReportEmitter	
public	startTable	void	ITableContent table	CSVReportEmitter	
public	startText	void	ITextContent text	CSVReportEmitter	

Figura 42 – Resultado da busca por métodos.

Para uma outra situação, foi elaborada uma consulta para as classes extraídas, tendo como retorno dados da própria tabela (CLASSES) e de suas tabelas dependentes (ATRIBUTOS, ANOTACOES, METODOS e EXCECOES) (Figura 43).

```
SET TERM ^ ;

CREATE PROCEDURE STP_CONSULTA_CLASSES (
    PNOME varchar(60),
    PTIPO varchar(20),
    PMODIF varchar(20))
returns (
    MODIFICADOR varchar(20),
    TIPO_OBJETO varchar(20),
    CLASSE varchar(60),
    SUPERCLASSE varchar(60),
    ATRIBUTO varchar(400),
    ANOTACAO varchar(200),
    METODO varchar(2000))
as
declare variable ID_CLASSE integer;
declare variable ID_METODO integer;
declare variable T_METODO varchar(300) character set WIN1252;
declare variable T_ATRIBUTO varchar(100) character set WIN1252;
declare variable T_EXCECAO varchar(60) character set WIN1252;
declare variable T_MODIFICADOR varchar(60) character set WIN1252;
declare variable T_TIPO_RETORNO varchar(60) character set WIN1252;
declare variable T_PARAMETROS varchar(100) character set WIN1252;
declare variable T_ANOTACAO varchar(60) character set WIN1252;
declare variable EXCECAO varchar(300);
begin
  for select distinct c.MODIFICADOR, c.TIPO_OBJETO, c.NOME, c.NOME_SUPERCLASSE,
             c.ID_CLASSE from CLASSES c
      where c.NOME like :PNOME and c.TIPO_OBJETO like :PTIPO
      and c.MODIFICADOR like :PMODIF order by c.NOME
      into :MODIFICADOR,:TIPO_OBJETO,:CLASSE,:SUPERCLASSE,:ID_CLASSE
  do begin
    ATRIBUTO=''; METODO=''; ANOTACAO='';
    for select first 5 a.MODIFICADOR||' '||a.TIPO||' '||a.NOME
        from ATRIBUTOS a where a.ID_CLASSE=:ID_CLASSE order by a.NOME
        into :T_ATRIBUTO
    do begin
           if (:ATRIBUTO='') then ATRIBUTO=:T_ATRIBUTO;
           else ATRIBUTO=:ATRIBUTO||', '||:T_ATRIBUTO;
    end
    for select first 10 a.NOME from ANOTACOES a
        where a.ID_CLASSE=:ID_CLASSE order by a.NOME into :T_ANOTACAO
    do begin
        if (:ANOTACAO='') then ANOTACAO=:T_ANOTACAO;
        else ANOTACAO=:ANOTACAO||', '||:T_ANOTACAO;
    end
    for select first 10 m.MODIFICADOR,m.TIPO_RETORNO,m.NOME,m.PARAMETROS,
        m.ID_METODO from METODOS m where m.ID_CLASSE=:ID_CLASSE order by m.NOME
        into :T_MODIFICADOR,:T_TIPO_RETORNO,:T_METODO,:T_PARAMETROS,:ID_METODO
    do begin
      if (:T_MODIFICADOR is null) then T_MODIFICADOR='';
      if (:T_TIPO_RETORNO is null) then T_TIPO_RETORNO='';
      if (:T_PARAMETROS is null) then T_PARAMETROS='';
      T_METODO=:T_MODIFICADOR||' '||:T_TIPO_RETORNO||' '||:T_METODO||'('||
               :T_PARAMETROS||')';
      if (:METODO='') then METODO=:T_METODO;
      else METODO=:METODO||', '||:T_METODO;
      EXCECAO='';
      for select first 10 e.NOME from EXCECOES e where e.ID_METODO=:ID_METODO
          order by e.NOME into :T_EXCECAO
      do begin
           if (:EXCECAO='') then EXCECAO=:T_EXCECAO;
           else EXCECAO=:EXCECAO||', '||:T_EXCECAO;
      end
      if (:EXCECAO='') then METODO=:T_METODO;
           else METODO=:T_METODO||' EX: '||EXCECAO;

    end
    suspend;
  end
end^

SET TERM ; ^
```

Figura 43- Código da consulta às classes extraídas.

Após a execução da consulta para os parâmetros PNOME=” %i%”, PTIPO=”%” e PMODIF=”public”, foi obtido o resultado conforme a figura 44.

MODIFICADOR	TIPO_OBJETO	CLASSE	SUPERCLASSE	ATRIBUTO	ANOTACAO	METODO
public	class	CSVReportEmitter	ContentEmitterAdapter	protected static final String OUTPUT_FORMAT_CSV, protected static final String REPORT_FILE, protected ContentEmitterVisitor contentVisitor, protected int currentColumn, protected long firstTableID		public void startLabel(ILabelContent label) EX: BirtException
public	class	DomainmodelJvmModelInferrer		private DomainmodelExtensions domainmodelExtensions, private IJvmModelAssociator jvmModelAssociator, private JvmVisibilityExtension jvmVisibilityExtension, private TypesFactory typesFactory	@Inject	public List<JvmDeclaredType> inferJvmModel(final EObject sourceObject)
public	interface	IEventListner		public static final int DOWN, public static final int LEFT, public static final int NEWGAME, public static final int PAUSE, public static final int RIGHT		public void incomingEvent(int eventType)
public	class	PdfViewerComposite	Composite	public static final UseCase USE_CASE_DEFAULT, private EnumSet<PdfViewerCompositeOption> options, private PdfCoolBar pdfCoolBar, private Control pdfCoolBarControl, private PdfSimpleNavigator pdfSimpleNavigator		public void run()
public	class	SubversiveAdapter	Adapter			private static void handleStatus(IStatus status) EX: SVNException
public	class	UsuarioDao			@GET, @Path("{name}"), @Produces("application/xml")	public static Usuario login(String nomeUsuarioTrim String senhaTrim)
public	class	light_html2xml				public static String Html2Xml(String s)

Figura 44 – Resultado da busca por classes.

4.2.1 Instanciação OO, Ontologias e SPARQL

Para essa segunda instanciação, foi usada a linguagem OWL para a construção da ontologia proposta.

A ferramenta para construção de ontologias Protégé (atualmente *open source*), teve sua primeira versão desenvolvida no ano 2000 pela Universidade de Stanford Medical Informatics com o propósito para a criação de ontologias para área médica (PROTÉGÉ, 2011).

Para a obtenção da segunda instanciação OO, foi utilizada a API Jena[9] na ferramenta OopExtract (citada no subcapítulo anterior). Enquanto a ferramenta **OOPExtract** extrai as informações e insere no SGBD Firebird, também é feita a inclusão de novos indivíduos em uma instância da ontologia. Também foram utilizadas a ferramenta Protégé, a linguagem de representação OWL e a linguagem de consultas SPARQL.

Foram criadas classes, propriedades (*Object Properties*), e tipos de dados (*Data Properties*), de modo que, estruturalmente, se assemelhasse com o modelo relacional utilizado (Tabela 6). Para os tipos de dados criados, foi utilizado apenas o tipo primitivo *String*.

Tabela 6 – Componentes da ontologia criada.

Classes	**Propriedades**	**Tipos de Dados**
ANOTACOES	AnotacaodeClasse	ID_CLASSE
ATRIBUTOS	AtributodeClasse	MODIFICADOR TIPO ID_CLASSE
CLASSES	PossuiAnotacao PossuiAtributo PossuiMetodo	MODIFICADOR TIPO_OBJETO IMPLEMENTACOES ID_SUPERCLASSE NOME_SUPERCLASSE
EXCECOES	ExcecaodeMetodo	ID_METODO
METODOS	MetododeClasse PossuiExcecao	MODIFICADOR TIPO_RETORNO PARAMETROS ID_CLASSE
Thing		ID NOME

[9] Jena é uma API open-source para linguagem Java cujo objetivo é auxiliar no desenvolvimento de aplicações e ferramentas que façam uso de Semantic Web e Linked-Data .
Foi criada por pesquisadores da HP Labs no ano de 2000 e, posteriormente no ano de 2010, a equipe integrou-se à Apache *Software* Foundation.

Os componentes e seus relacionamentos (exceto os tipos de dados) citados na tabela 8, podem ser melhor visualizados através de um grafo gerado pela ferramenta Protégé (*plugin* OntoGraph), como mostra a figura 45.

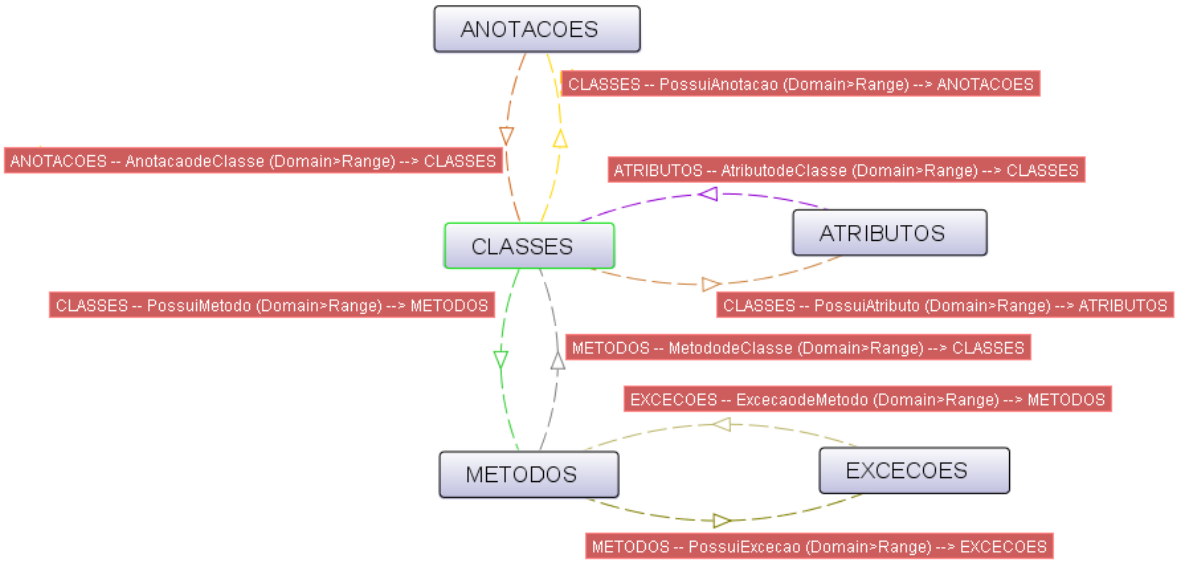


Figura 45 – Ontologia OO.

Após popular a ontologia é possível visualizar graficamente as informações. A figura 46 mostra o relacionamento de duas classes com seus respectivos atributos, anotações, métodos e exceções.

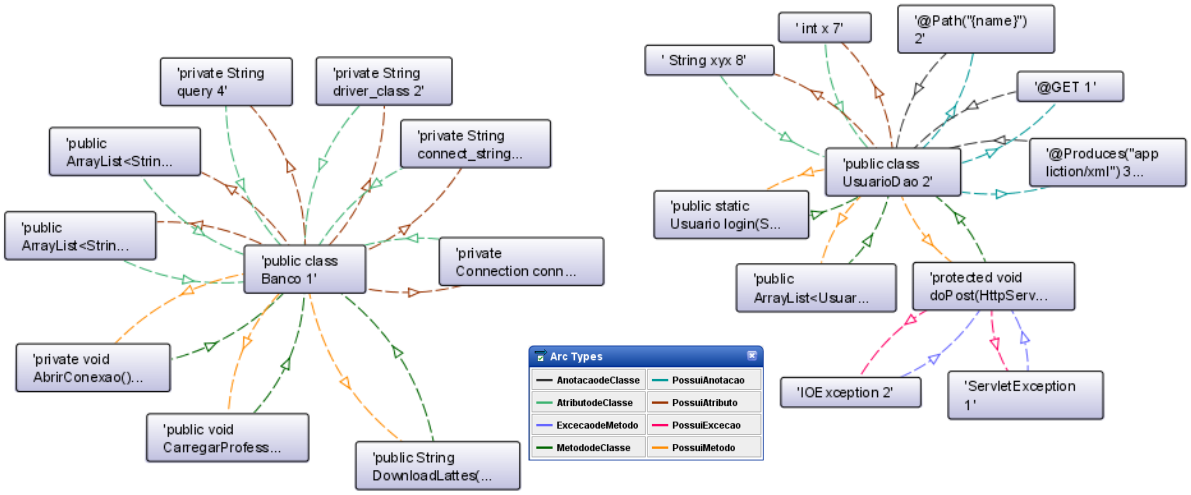


Figura 46 – Relacionamento entre indivíduos da ontologia usada.

Foram elaboradas algumas consultas usando SPARQL, através do *plugin* SPARQL Query, também presente na ferramenta Protégé.

```
1 PREFIX tb: <http://www.owl-ontologies.com/Ontology1338318212.owl#>
2 SELECT ?classe ?atributo
3 WHERE {
4     ?classe tb:PossuiAtributo ?atributo.
5 }
6 ORDER BY ?classe ?atributo
```

Figura 47 – Código da consulta por atributos.

A figura 47 mostra o código fonte de consulta, utilizando SPARQL, que lista todos os atributos das classes através da propriedade **PossuiAtributo**. Após a execução é obtido o resultado conforme a figura 48.

classe	atributo
'public class Banco 1'	'private Connection conn 1'
'public class Banco 1'	'private String driver_class 2'
'public class Banco 1'	'private String connect_string 3'
'public class Banco 1'	'private String query 4'
'public class Banco 1'	'public ArrayList<String> lstlinks 5'
'public class Banco 1'	'public ArrayList<String> lstprofessores 6'
'public class UsuarioDao 2'	' int x 7'
'public class UsuarioDao 2'	' String xyx 8'

Figura 48 – Resultado da consulta por atributos.

Uma outra consulta, utilizando três classes da ontologia, pode ser visualizada na figura 49.

```
1 PREFIX tb: <http://www.owl-ontologies.com/Ontology1338318212.owl#>
2 SELECT ?classe ?metodo ?excecao
3 WHERE {
4     ?classe tb:PossuiMetodo ?metodo.
5     ?metodo tb:PossuiExcecao ?excecao.
6 }
7 ORDER BY ?classe ?atributo ?excecao
```

Figura 49 – Código da consulta por exceções.

Como resultado, após a execução do código, foi obtido como resultado a listagem de exceções, seus respectivos métodos e classes (Figura 50).

classe	metodo	excecao
'public class UsuarioDao 2'	'protected void doPost(HttpServletRequest request,HttpServletResponse response) 4'	'ServletException 1'
'public class UsuarioDao 2'	'protected void doPost(HttpServletRequest request,HttpServletResponse response) 4'	'IOException 2'

Figura 50 – Resultado da consulta por exceções.

4.3 Instanciação OA, XML e XQuery

Para a instanciação OA (Anexo C) e XML, encontra-se em processo de desenvolvimento uma ferramenta de extração de metadados e consulta (denominada AOPJungle), a qual é o trabalho de dissertação do aluno Cristiano de Faveri, também integrante do projeto "Uma linguagem de consulta para a busca de oportunidades de refatoração em *software* orientado a aspectos e em *software* orientado a objetos" submetido ao **Edital FAPERGS 001/2010**, coordenado pelo professor Eduardo Kessler Piveta no âmbito do Laboratório de Linguagens de Programação e Bancos de Dados da UFSM.

Além da utilização do processo proposto neste trabalho, a ferramenta AOPJungle tem como objetivo atingir níveis de granularidade mais finos do código fonte POA, buscando *bad smells* para possíveis sugestões de refatoração do código fonte.

Para a exemplificação de instanciação deste trabalho, foi usado um resultado prévio fornecido pela ferramenta AOPJungle, o qual consiste de metadados extraídos de código fonte escritos em linguagem Java com AspectJ (Anexo D) e representados em um arquivo XML.

Para esta instanciação, foram selecionados alguns elementos presentes em código AO (Tabela **7**).

A figura 51 representa graficamente a relação entre os elementos utilizados nesta instanciação.

Através do arquivo XML gerado pela ferramenta AOPJungle, foram elaboradas algumas consultas utilizando XQuery (Anexo E). A figura 52 mostra o código-fonte elaborado para uma listagem dos aspectos e seus respectivos pontos de corte.

Tabela 7 – Elementos selecionados para a instanciação POA.

Elemento	**Descrição**	**Sintaxe Analizada**
Aspecto	Os aspectos possibilitam a separação de interesses. Dentro deles são declarados os pontos de corte e adendos. Também podem, além de estender outros aspectos, estender uma classe ou implementar interfaces.	**modificadores** "**aspect**" nome_aspecto ["**extends**" nome_classe_aspecto] ["**implements**" nome_interface] "**{**" Sendo: • **modificador**: conjunto de um ou mais modificadores (ex:. public abstract). • **extends**: Usado caso a classe seja a especialização de outro aspecto ou classe. • **implements**: Usado caso o aspecto implemente uma interface.
Intertipo	São declarações que alteram estaticamente atributos, métodos ou construtores ao programa.	**modificadores** "**tipo**" nome_intertipo"**;**" Sendo: • **modificadores**: conjunto de um ou mais modificadores (ex:. public abstract). • **tipo**: tipo de dado que a variável irá suportar (ex.: int, String, tipos abstratos, entre outros).

Adendo	São responsaveis pela execução das ações associadas aos pontos de corte. Podem ser dos tipos *before*, *after* ou *around*.	tipo_advice **():** nome_pointcut "**{**"
		Sendo:
		• **tipo_advice**: *after*, *before* ou *around*.
		
		Exemplo:
		after() returning : testepointcut() {
		System.out.println("Teste");
		}

Ponto de corte	É um conjunto de pontos de junção agrupados sintaticamente pelo AspectJ. Dentre eles podem ser citados: chamadas de métodos, *handlers* e adendos.	**pointcut** nome_pointcut**() :** expressao_chamada**;** Sendo: • **expressão_chamada**: onde o PointCut irá ser chamado. Ex:. call (void metodoteste()).
Warning	São mensagens de aviso (*warning*) ou erros (*errors*) contidas no interior do aspecto.	**declare error :** nome_pointcut **:** "mensagem"**;** **declare warning :** nome_pointcut **:** "mensagem"**;**

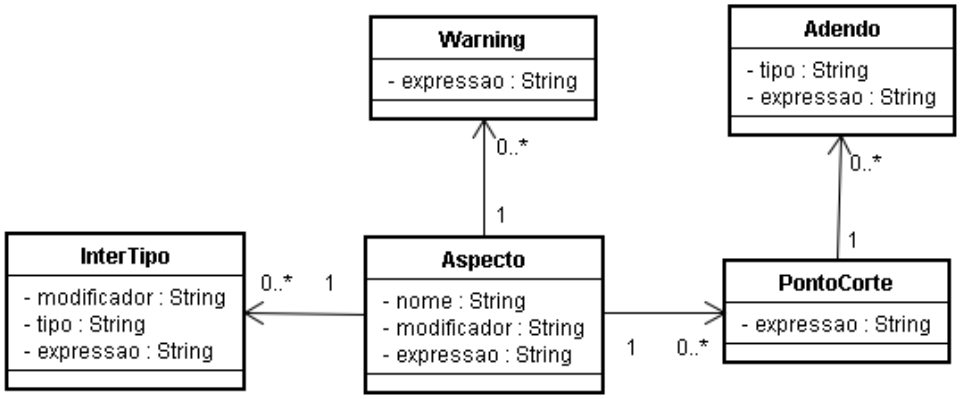


Figura 51 – Diagrama de domínio da instanciação POA.

```
<Projetos>{
let $doc := .
for $project in $doc//*[ends-with(name(), "AOJProject")]
return
    <Projeto Nome = "{$project/name}">
        <Pacotes>{
        for $package in $project//*[ends-with(name(),"AOJPackageDeclaration")]
        return
           <Pacote Nome = "{$package/name}">{
                for $aspect in $package//*[ends-with(name(),"AOJAspectDeclaration")]
                return
                    <Aspecto Nome = "{$aspect/name}">{
                        for $pointCut in $aspect//*[ends-with(name(),"AOJPointcutDeclaration")]
                        return
                            <PontoCorte Expressão = "{$pointCut/pointcutExpression/code}"></PontoCorte>
                    }</Aspecto>
           }</Pacote>
        }</Pacotes>
    </Projeto>
}</Projetos>
```

Figura 52 – Código fonte da consulta por aspectos e pontos de corte.

A figura 53 mostra como resultado obtido, apenas a estrutura dos programas lidos, seus pacotes, aspectos e pontos de corte.

```
- <Projetos>
  + <Projeto Nome="AOPTetris">
  + <Projeto Nome="Bean">
  + <Projeto Nome="Coordination">
  + <Projeto Nome="ITW">
  + <Projeto Nome="Introduction">
  + <Projeto Nome="JASTVisitor">
  - <Projeto Nome="Observer">
    - <Pacotes>
      - <Pacote Nome="org.eclipse.aspectj.examples.observer">
        - <Aspecto Nome="SubjectObserverProtocol">
            <PontoCorte Expressão="abstract pointcutstateChanges(Subject s): ;" />
          </Aspecto>
        - <Aspecto Nome="SubjectObserverProtocolImpl">
            <PontoCorte Expressão="target(s) and call(void Button.click())" />
          </Aspecto>
        </Pacote>
      </Pacotes>
    </Projeto>
  + <Projeto Nome="Spacewar">
  + <Projeto Nome="TJP">
  + <Projeto Nome="Telecom">
  + <Projeto Nome="Tracing">
  </Projetos>
```

Figura 53 – Resultado da consulta por aspectos e pontos de corte.

O código-fonte de uma consulta mostrando todos os aspectos e seus adendos pode ser visualizado na figura 54.

```
<Projetos>{
let $doc := .
for $project in $doc//*[ends-with(name(), "AOJProject")]
return
    <Projeto Nome = "{$project/name}">
        <Pacotes>{
        for $package in $project//*[ends-with(name(),"AOJPackageDeclaration")]
        return
           <Pacote Nome = "{$package/name}">{
                for $aspect in $package//*[ends-with(name(),"AOJAspectDeclaration")]
                return
                    <Aspecto Nome = "{$aspect/name}">{
                        for $advicebefore in $aspect//*[ends-with(name(),"AOJAdviceBefore")]
                        return
                            <AdviceBefore Expressão = "{$advicebefore/pointcutExpression}"> </AdviceBefore>,
                        for $adviceafter in $aspect//*[ends-with(name(),"AOJAdviceAfter")]
                        return
                            <AdviceAfter Expressão = "{$adviceafter/pointcutExpression}"> </AdviceAfter>,
                        for $advicearound in $aspect//*[ends-with(name(),"AOJAdviceAround")]
                        return
                            <AdviceAround Expressão = "{$advicearound/pointcutExpression}"> </AdviceAround>
                    }</Aspecto>
           }</Pacote>
        }</Pacotes>
    </Projeto>
}</Projetos>
```

Figura 54 – Código fonte da consulta por aspectos e adendos.

Como resultado da execução da consulta, foram obtidas as informações conforme a figura 55.

```
- <Projetos>
  + <Projeto Nome="AOPTetris">
  - <Projeto Nome="Bean">
    - <Pacotes>
      - <Pacote Nome="org.eclipse.aspectJ.example.bean">
        - <Aspecto Nome="BoundPoint">
            <AdviceAround Expressão="execution(void Point.setX(int)) and target(p)" />
            <AdviceAround Expressão="execution(void Point.setY(int)) and target(p)" />
          </Aspecto>
        </Pacote>
      </Pacotes>
    </Projeto>
  - <Projeto Nome="Coordination">
    - <Pacotes>
      - <Pacote Nome="org.eclipse.aspectj.examples.coordination">
        - <Aspecto Nome="Coordinator">
            <AdviceBefore Expressão="synchronizationPoint" />
            <AdviceAfter Expressão="synchronizationPoint" />
          </Aspecto>
        </Pacote>
      </Pacotes>
    </Projeto>
  + <Projeto Nome="ITW">
  + <Projeto Nome="Introduction">
  + <Projeto Nome="JASTVisitor">
  + <Projeto Nome="Observer">
  + <Projeto Nome="Spacewar">
  + <Projeto Nome="TJP">
  + <Projeto Nome="Telecom">
  + <Projeto Nome="Tracing">
  </Projetos>
```

Figura 55 – Resultado da consulta por aspectos e adendos.

4.4 Considerações Finais

As instanciações realizadas neste trabalho mostram que o processo proposto pode ser usado em diferentes contextos (paradigmas, linguagens, modelos de representacao, linguagens de consulta). Tanto a instanciação OO quanto a OA mostram que independente dos paradigmas e/ou linguagens alvo a serem consultadas, o processo para a obtenção dos resultados nos metadados permanece o mesmo.

Na instanciação OO, foi implementada uma ferramenta simples (OopExtract) cujas funcionalidades básicas são ler código-fonte e popular um banco de dados relacional e uma instância de ontologia. As consultas podem ser feitas com linguagens criadas para os respectivos modelos de representação (SQL e SPARQL, respectivamente).

A instanciação OA, foram usados documentos XML gerados pela ferramenta AOPJungle (parte integrante da dissertação do aluno Cristiano de Faveri), a qual fez uso do processo proposto neste trabalho.

5 CONCLUSÃO

Devido à crescente evolução das linguagens de programação, novos paradigmas vão surgindo fazendo com que os arquivos de código-fonte se tornem cada vez mais extensos, dificultando assim a localização de erros. Este problema se agrava quando é necessário encontrar um determinado elemento (uma classe ou método por exemplo) em algum arquivo de código-fonte. Como o objetivo de minimizar essa dificuldade, este trabalho propõe um processo de busca em repositórios de código-fonte. O processo pode ser utilizado de acordo com a necessidade e nível de precisão adequada a cada problema.

Este trabalho baseou-se em um estudo sobre alguns dos processos de *software* mais utilizados, paradigmas de programação Orientação a Objetos (OO) e Orientação a Aspectos (AO), linguagens de consulta e alguns trabalhos relacionados. A especificação do processo foi feita através da ferramenta EPF Composer, utilizando alguns elementos do *Software Process Engineering Metamodel* (SPEM).

Para exemplificar o uso do processo, foram feitas três instanciações: OO, modelo relacional e *Structured Query Language* (SQL), OO, ontologias e SPARQL, e por fim, OA, *eXtensible Markup Language* (XML) e XQuery. Para as duas instanciações OO, foi implementada uma ferramenta para extração de metadados e população de uma ontologia e um banco de dados relacional. Já na instanciação OA, foram usados documentos XML resultantes da ferramenta implementada por Cristiano de Faveri que utilizou o processo proposto neste trabalho.

Devido ao projeto de AOP Jungle estar em andamento, foram utilizadas linguagens de consultas existentes, entretanto, devido ao aumento da complexidade dos codigos, é notável a necessidade de linguagens de consultas mais específicas.

Logo, para trabalhos futuros, sugere-se a criação de uma linguagem de consulta específica para código-fonte *Domain-Specific Languages* (DSL), a qual faz parte do trabalho de Cristiano de Faveri, podendo esta ser usada para busca inclusive nos blocos de comandos, fornecendo uma busca mais refinada e precisa.

REFERÊNCIAS BIBLIOGRÁFICAS

ABREU, T. C. Programação Orientada a Aspectos - Um exemplo prático utilizando AspectJ. *Engenharia de Software Magazine* 10, 2009.

ACUÑA, S. T. et al. The software process: modelling, evaluation and improvement. In: CHANG, S. K. (Ed.). **Handbook of Software Engineering and Knowledge Engineering**. Singapore: World Scientific Publishing Company, v.1, 2000. p.193-237.

BERTOLLO, G.; SEGRINI, B.; FALBO, R. D. Definição de Processos de Software em um Ambiente de Desenvolvimento de Software Baseado em Ontologias. In: SIMPÓSIO BRASILEIRO DE QUALIDADE DE SOFTWARE, 5., 2006, . **Anais eletrônicos...** p. 15. Disponível em:. Acesso em: 29 maio 2006.

CIRACI, S.; BROEK, P. V.; AKSIT, M. Graph-Based Verification of Static Program Constraints. In: SYMPOSIUM ON APPLIED COMPUTING, 25., 2010, Sierre. **Proceedings…** Sierre, Switzerland, 2010. pp. 2265-2273.

COHEN, T.; YOSSI, J. JTL – the Java Tools Language. *Reverse Engineering* , pp. 89-108. 2006.

ECLIPSE FOUNDATION. **AspectJ**. 2011. Disponível em: <http://eclipse.org/aspectj/>. Acesso em: 29 jul. 2011.

_____. **Eclipse Java development tools**. 2012. Disponível em: <http://www.eclipse.org/jdt>. Acesso em: 29 fev. 2012.

_____. **Eclipse Process Framework Composer - Installation, Introduction, Tutorial and Manual, 1.0**. 2010. Disponível em: <http://www.eclipse.org/epf/general/getting_started.php>. Acesso em: 03 set. 2011.

_____. **Essentials of OpenUP - Eclipse Process Framework Project**. 2009. Disponível em: <http://www.eclipse.org/epf/general/>. Acesso em: 20 set. 2011.

_____. **OpenUp**. 2011. Disponível em: <http://epf.eclipse.org/wikis/openup/>. Acesso em: 04 set. 2011.

ELMASRI, R.; NAVTHE, S. B. **Sistemas de Banco de Dados**. São Paulo: Pearson Addison Wesley, 2011.

ERICKSON, M. **DeveloperWorks**. 2001. Disponível em: <http://www-128.ibm.com/developerworks/opensource/library/os-eclipse.html>. Acesso em: 08 fev. 2011.

FALBO, R. D.; BERTOLLO, G. Establishing a Common Vocabulary for Software Organizations Understand Software Processes. In: EDOC INTERNATIONAL WORKSHOP ON VOCABULARIES, ONTOLOGIES AND RULES FOR THE ENTERPRISE , 2005, Netherlands. **Proceedings…** Netherlands. p. 8.

FOWLER, M.; PARSONS, R. **Domain-Specific Languages**. Boston: Addison-Wesley, 2010.

FUNDAÇÃO DE AMPARO À PESQUISA DO RIO GRANDE DO SUL (FAPERGS). Uma linguagem de consulta para a busca de oportunidades de refatoração em software orientado a aspectos e em software orientado a objetos. (Edital FAPERGS 001), 2010.

GENVIGIR, E. C.; SANT'ANNA, N.; FILHO, B. L. Modelagem de processos de software através do SPEM - software process engineering meta model - conceitos e aplicação. In: WORKSHOP DOS CURSOS DE COMPUTAÇÃO APLICADA DO INPE, 3., pp. 85-90. 2003.

GRUBER, T. R. A Translation Approach to Portable Ontology Specifications. *Knowledge Systems Laboratory Technical Report*, v. 5. n. 2, p. 199-220. 1993.

GUARINO, N. Formal Ontology and Information Systems. In: FIRST INTERNATIONAL CONFERENCE, 1998, Trento. **Proceedings…** Trento, Itália , pp. 03-15, 1998.

JUBILEU, A. P. **Modelo de Gestão do Processo de Venda e Desenvolvimento de Software On-Demand para MPE's**. 2008. 330 f. Tese (Doutorado em Engenharia de Produção) - Escolaa de Engenharia de São Carlos, Universidade de São Paulo, São Paulo, 2008.

KICZALES, G. **Keynote talk at AOSD**. 2003. Disponível em: <http://www.cs.ubc.ca/~gregor/papers/kiczales-aosd-2003.ppt>. Acesso em: 23 jun. 2011.

KICZALES, G. et al. Aspect-Oriented Programming. In: EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, 15., 1997, Finlândia. **Proceedings…** Finlândia: Springer-Verlag, 1997.

KORTH, H. F.; SILBERSCHATZ, A.; SUDARSHAN, S. **Sistema de Banco de Dados**. São Paulo: Makron Books, 2006.

LIMA, J. C.; CARVALHO, C. L. **Ontologias - OWL (Web Ontology Language)**. Goiás: Instituto de Informática - Universidade Federal de Goiás, 2005.

MCCORMICK, E.; DE VOLDER, K. JQuery: finding your way through tangled code. In: COMPANION TO OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES, AND APPLICATIONS, 19., 2004, Nova Iorque. **Proceedings…** Nova Iorque: ACM Press, 2004, p. 9-10.

NUNES, C. M. Uma Linguagem de Domínio Específico para a Framework i*.
OMG, 2008. *Software & Systems Process Engineering Meta-Model Specification*, 2.0. Disponível em: <http://www.omg.org/spec/SPEM/2.0/PDF/>. Acesso em: 03 set. 2011.

PFEIFER, H. J.; SARDOS, A.; GURD, J. R. Complex Code Querying and Navigation for AspectJ. In: OOPSLA WORKSHOP ON ECLIPSE TECHNOLOGY EXCHANGE, 2005. **Proceedings…** 2005, p. 60-64.

PIVETA, E. K. (2011). **Página Pessoal - Eduardo Piveta - UFSM**. Disponível em: <http://www-usr.inf.ufsm.br/~piveta/teaching.html>. Acesso em: 18 ago. 2011.

PIVETA, E. K.; PRICE, R. T.; PIMENTA, M. S. (2009). Improving the Search for Refactoring Opportunities on Object-Oriented and Aspect-Oriented Software. Universidade Federal do Rio Grande do Sul .

PRESSMAN, R. S. **Engenharia de Software**. 6. ed. Porto Alegre: Mc Graw Hill - Bookman, 2010.

PROTÉGÉ. **Getting Started with Protégé 4.x OWL**. Disponível em: <http://protegewiki.stanford.edu/wiki/Protege4GettingStarted>. Acesso em: 06 dez. 2011.

REZENDE, A. M.; SILVA, C. C. **Programação Orientada a Aspectos em Java**. Rio de Janeiro: Brasport, 2005

RUSSEL, S.; NORVIG, P. **Artificial Intelligence:** A Modern Approach. 2. ed. New Jersey: Prentice Hall, 2002.

SEBESTA, R. W. Conceitos de linguagens de progamação. 9. ed. Porto Alegre: Bookman, 2011.

SMITH, M. K.; WELTY, C.; MCGUINNESS, D. L. OWL Web Ontology Language Guide. *W3C Recommendation*, 2004. Disponível em: <http://www.w3.org/TR/2004/REC-owl-guide-20040210/#Introduction>. Acesso em: 16 dez. 2011.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson, 2011.

THE APACHE SOFTWARE FOUNDATION (ASF). (2012). **Jena Ontology API**. Disponível em: < http://jena.apache.org/documentation/ontology/index.html>. Acesso em: 20 maio 2012.

VALLE, R.; BARBARÁ, S. **Análise e modelagem de processos de negócio:** foco na notação BPMN. 1. ed. São Paulo: Atlas, 2009.

VOLDER, K. D. **Type-Oriented Logic Meta Programming**. 1998. 226 f. Tese (Doutorado em Informática)- Vrije Universiteit Brussel, Wetenschappen, 1998.

W3C. **About W3C**. 2012. Disponível em: <http://www.w3.org/Consortium/>. Acesso em 30 jun. 2012.

W3C. **XML Path Language (XPath) 2.0.** 2. ed. 2010a. Disponível em: <http://www.w3.org/TR/xpath20/>. Acesso em: 02 jul. 2012.

W3C. **XQuery 1.0: An XML Query Language**. 2. ed. 2010b. Disponível em: <http://www.w3.org/TR/xquery/>. Acesso em: 02 jul. 2012.

WEGRZYNOWICKZ, P.; STENCEL, K. The Good, The Bad, and The Ugly – Three Ways to Use a Semantic Code Query System. In: OBJECT-ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES, AND APPLICATIONS , 24., 2009, Orlando. Proceedings… Orlando: ACM, 2009, p. 821-822.

WIKIPEDIA. **Benchmark (computação)**. Disponível em: <http://pt.wikipedia.org/wiki/Benchmark_%28computa%C3%A7%C3%A3o%29>. Acesso em: 19 ago. 2011.

WINCK, D. V.; JUNIOR, G. V. **AspectJ:** Programação Orientada a Aspectos com Java. São Paulo: Novatec Editora, 2006.

ANEXOS

Anexo A –*Open Unified Process* (OpenUP)

(continua)

O OpenUp é um processo unificado que utiliza a abordagem iterativa e incremental em um ciclo de vida estruturado. Tem por base quatro princípios fundamentais (ECLIPSE FOUNDATION, 2009):

- **Equilíbrio**: Compatibilidade entre as restrições impostas ao projeto e a maximização dos benefícios aos envolvidos.
- **Colaboração**: Alinhamento de interesses e compartilhamento do entendimento.
- **Foco na arquitetura**: Preocupação com a orquestração ou estrutura dos componentes do projeto, a fim de reduzir riscos e organização do desenvolvimento.
- **Evolução**: Utilizar boas práticas que permitam a obtenção de *feedbacks* dos envolvidos o mais breve possível, a fim de promover um valor incremental ao projeto.

O OpenUp incorpora ideias de processo ágil e foca na natureza colaborativa do desenvolvimento do projeto (JUBILEU, 2008).

A figura 56 mostra o ciclo de vida de um projeto que usa o OpenUp, onde a cada iteração é produzido um incremento que gera um *feedback* para melhor direcionamento do projeto.

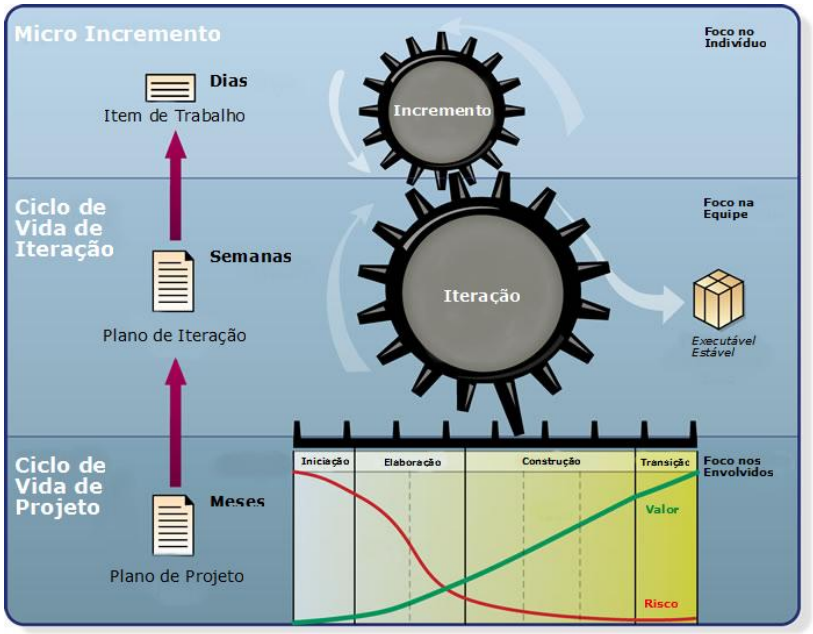


Figura 56 - Organização de projeto focado no OpenUp.

Fonte: OpenUp (2011).

Anexo A –*Open Unified Process* (OpenUP)

(continuação)

Da maneira como é originalmente estruturado, o OpenUp, deve ser utilizado em projetos pequenos com equipe também pequena, porém pode ser extensível a projetos de quaisquer dimensões e equipes geograficamente distribuídas (ECLIPSE FOUNDATION, 2011a).

Da mesma forma que o *Rational Unified Process* (RUP), o OpenUp é organizado em:

- **Disciplinas**: auxiliam na subdivisão das tarefas e organizam o fluxo de trabalho. Subdividem-se em: requisitos, arquitetura, desenvolvimento, teste, gestão de projeto e gestão de configuração e mudança.
- **Artefatos**: encontram-se organizados por domínios (atividades e comportamentos). São subdivididos em: arquitetura, desenvolvimento, gestão de projeto, requisitos e teste.
- **Papéis**: são as funções que as pessoas executam quando trabalham em equipe. Originalmente o OpenUp apresenta as seguintes denominações: arquiteto, gerente de projeto, analista, testador, qualquer papel, desenvolvedor e envolvido (Figura 57).
-

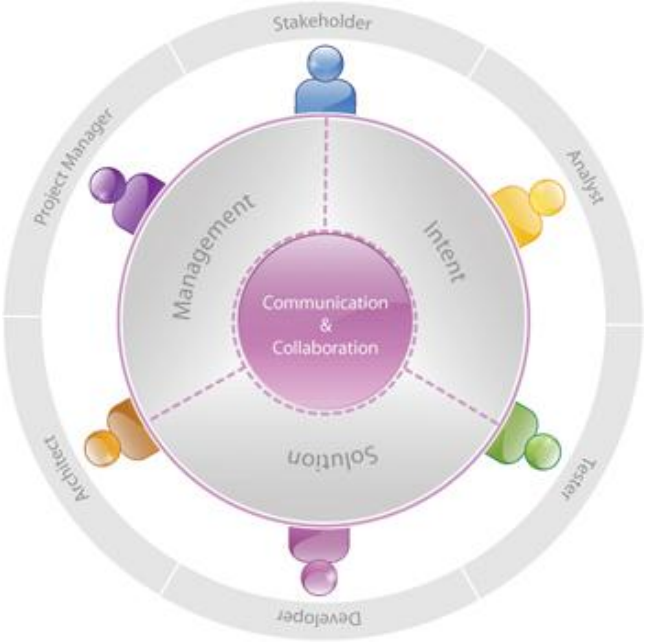


Figura 57 - Papéis e disciplinas do OpenUp.

Fonte: ECLIPSE FOUNDATION (2009).

Anexo A –*Open Unified Process* (OpenUP)

(continuação)

- **Ciclo de Vida**: descreve um processo de desenvolvimento que oferece suporte aos quatro princípios básicos do OpenUp. Cada fase tem seu início e fim bem definidos, como mostra a figura 58: Concepção (*Inception*), Elaboração (*Elaboration*), Construção (*Construction*) e Transição (*Transition*).

Figura 58 - Fases do OpenUp.

Fonte: ECLIPSE FOUNDATION (2009).

As fases do ciclo de vida dos projetos baseado no OpenUp são orientadas por objetivos específicos descritos na tabela 8 (ECLIPSE FOUNDATION, 2011a).

O OpenUp pode ser utilizado através do EPF *Composer* (seção seguinte) para modificar ou estender seus elementos. Estas alterações podem ser tanto simples alteração de *templates* de artefatos quanto uma inclusão de atividade necessária ao projeto. Também podem ser feitas alterações de fluxos de processo dependendo dos padrões específicos do projeto (ECLIPSE FOUNDATION, 2010).

Anexo A –*Open Unified Process* (OpenUP)

(conclusão)

Tabela 8 – Fases do ciclo de vida do OpenUp.

Fase	**Objetivos**
Concepção	Determinar a visão, o escopo do sistema, e seus limites e identificar quem está interessado no sistema e por que.
	Identificar as principais funcionalidades do sistema e decidir quais requisitos são mais críticos.
	Determinar pelo menos uma solução possível.
	Entender custo, cronograma e riscos associados ao projeto.
Construção	Desenvolver de forma iterativa um produto completo que esteja pronto para ser entregue à comunidade de usuários.
	Minimizar os custos de desenvolvimento e consiga algum grau de paralelismo.
Elaboração	Obter um entendimento mais detalhado dos requisitos.
	Projetar, implementar, validar e estabelecer a linha de base da arquitetura.
	Atenuar os riscos essenciais e produzir um cronograma e uma estimativa de custos precisos.
Transição	Executar o teste Beta para validar se as expectativas dos usuários foram atendidas.
	Obter a concordância dos envolvidos de que a implantação está completa.
	Melhorar o desempenho de projetos futuros com as lições aprendidas.

Anexo B – *Domain-Specific Languages* (DSLs)

O objetivo principal de uma DSL é melhorar a legibilidade do código fonte, facilitar sua alteração, tendo como resultado, uma melhora na produtividade do programador. Outro benefício de uma DSL é, que se bem projetada, pode ser bem interpretada por pessoas do negócio e, consequentemente permitir uma melhor compreensão do código que implementa suas regras de negócios (FOWLER; PARSONS, 2010).

A ideia básica de uma DSL é ser uma linguagem focada em um problema em particular, ao invés de ser uma linguagem de propósito geral, que é voltada para qualquer tipo de problema de *software*. Muito tem se falado sobre DSLs, e são utilizadas desde a criação dos computadores (FOWLER; PARSONS, 2010).

Uma DSL tem como finalidade especificar e modelar conceitos num determinado propósito, tendo várias vantagens em relação às linguagens de propósito geral, tais como permitir expressar a solução de um problema na linguagem desejada e ao nível de abstração desejado. Para a criação de uma DSL com sucesso, normalmente é necessário iniciar pela especificação de sua sintaxe recorrendo a um metamodelo que será fornecido com entrada para uma bancada de linguagem (linguagem *workbench*[10]) que pode gerar o editor correspondente. Com um editor apropriado para a linguagem pode-se especificar modelos com a notação proposta (NUNES, 2009).

Diferente de uma linguagem de programação como C# ou Java, que pode implementar qualquer ideia, uma DSL está restrita um domínio como: consulta (SQL), formatação de texto (HTML), estruturação de dados (XML) e outros. Caracteriza-se por certas particularidades, tais como palavras e métodos de uma área de propósitos específicos e seus domínios gramaticais ou léxicos para exprimir suas finalidades.

[10] Conjunto de programas ou outras operações, a fim de avaliar o desempenho relativo de um objeto, normalmente executando uma série de testes padrões (WIKIPEDIA, 2011).

Anexo C – Orientação a Aspectos

(continua)

A POA, assim como a POO, introduz um novo paradigma para facilitar e melhorar o desenvolvimento de *software* (KICZALES et al., 1997).

A POA lida com um problema específico: capturar unidades consistentes de um sistema de *software* que as limitações dos modelos de programação tradicionais forçam a ficar espalhados por diversos pontos do sistema. Logo, a utilização deste paradigma de programação consiste na identificação dos interesses sistêmicos (requisitos não funcionais) espalhados no código e centralizá-los, de uma bem definida e coerente, favorecendo uma implementação com separação bem definida de interesses (KICZALES et al., 1997).

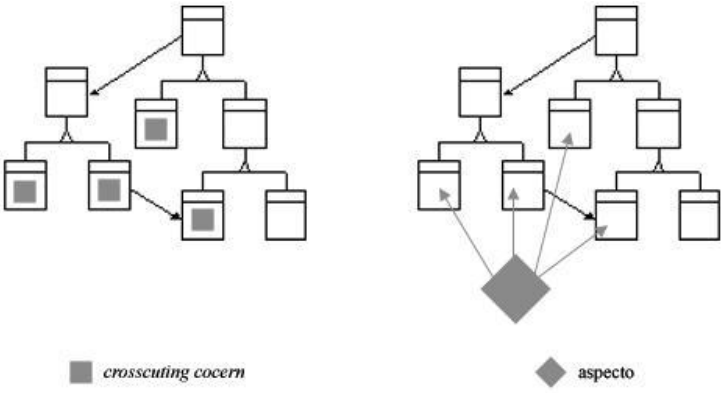


Figura 59 – Diferença entre interesses transversais espalhados no sistema e a abordagem de modularização através dos aspectos.

Fonte: Abreu (2009).

Para alguns requisitos de *software* são mapeados em preocupações que afetam diversas classes de uma forma sistemática, modificando a semântica e/ou o desempenho de um aplicativo (PIVETA; PRICE; PIMENTA, 2009). Logo, pode-se dizer que finalidade de POA é fornecer mecanismos de composição e de abstração para modularizar interesses transversais. Estes interesses, por sua vez, são as características relevantes de uma aplicação e podem ser divididos em diversos aspectos, a fim de representarem os requisitos do sistema (WINCK; JUNIOR, 2006).

Anexo C – Orientação a Aspectos

(conclusão)

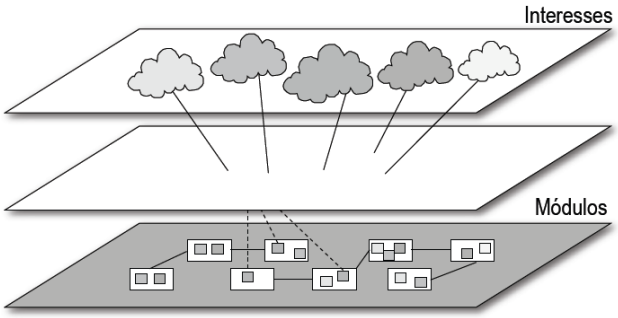


Figura 60 – Separação de Interesses.

Fonte: Winck e Junior (2006).

Anexo D – AspectJ

(continua)

A linguagem AspectJ, segundo (KICZALES, 2003), é uma extensão orientada a aspectos, genérica, da linguagem Java.

Em AspectJ, a unidade de modularidade é um aspecto, onde cada um define uma função específica que pode afetar diversas partes de um sistema.

Um aspecto, como uma classe Java, pode definir membros (atributos e métodos) e uma hierarquia de aspectos, através das seguintes definições:

- **Aspecto (*Aspect*)**: É semelhante a uma classe (POO), encapsula os diversos pontos de junção e seus adendos, são definidos em arquivos separados e constituem a unidade principal da programação orientada a aspecto, podem possuir os mesmos elementos que uma classe possui e alguns elementos adicionais, tais como conjunto de pontos de junção e adendos (REZENDE; SILVA, 2005).
- **Interesses (Concerns)**: Sistemas de *software* consistem de um conjunto de "áreas de interesse" ou interesses distintos como, por exemplo, funcionais (lógica de negócio) e não-funcionais (desempenho, persistência de dados, logging, autenticação de usuários, segurança, verificação de erros, etc.). Segundo Piveta, Price, e Pimenta (2009), existem também as preocupações relacionadas com o processo de desenvolvimento de *software* (Também chamadas de atributos de qualidade), como clareza de entendimento, facilidade de manutenção, rastreabilidade, simplicidade de evolução do *software*, etc.
- **Pontos de Junção (*Joinpoints*)**: Representam eventos de interesse do fluxo de execução. Quando a execução passa por um joinpoint o aspecto pode agir naquele ponto. Exemplo de pontos de junção: invocação de métodos, alteração de atributos e exceções.
- **Conjunto de Pontos de Junção (*Pointcuts*)**: Usados para representar um conjunto de joinpoints, pois podem acontecer muitas ocorrências de pontos de junção de um mesmo tipo. O AspectJ utiliza expressões regulares na definição de conjunto de pontos de junção (ECLIPSE FOUNDATION, 2011).
- **Inter-Tipos (*Inter-Types*)**: São declarações que refletem o estado ou comportamento de uma classe, aspecto ou interface. Evidenciam-se três tipos mais comuns de declarações: atributos, métodos e construtores.
- **Adendos (*Advices*)**: Os adendos constituem a porção de código que será executado quando alguma regra definida no ponto de atuação for válida. No adendo podemos

Anexo D – AspectJ

(conclusão)

especificar qual a característica temporal de execução (*before*, *after*, *around*) (KICZALES et al., 1997).

```
public aspect TraceAspect {
        private Logger logger = Logger.getLogger("trace");
                pointcut traceMethods() : execution(* *.*(..)) &&
        !within(TraceAspect);
                before() : traceMethods() {
                        Signature sig = thisJoinPointStaticPart.getSignature();
                        logger.logp(Level.INFO, sig.getDeclaringType().getName(),
                        sig.getName(), "Entering");
                }
}
public class ShoppingCart {
        private List items = new Vector();
        public void addItem(Item item) {
                items.add(item);
        }
        public void removeItem(Item item) {
                items.remove(item);
        }
        public void empty() {
                items.clear();
        }
}
```

Figura 61 – Exemplo de Aspecto.

Fonte: Piveta (2011).

Anexo E – XQuery

A linguagem XQuery foi desenvolvida pelo W3C[11] para consultas a documentos XML, permitindo acesso a partes específicas destes, tais como nós ou conjuntos de nós. Por meio dela, podem-se acessar as informações armazenadas em arquivos XML de maneira similar às consultas SQL. XQuery 1.0 é uma extensão de XPath[12] 2.0 e qualquer expressão sintaticamente correta será executada com sucesso em processadores de ambas as especificações.

A principal aplicação de XQuery é a extração de informações existentes em documentos XML, entre elas pode-se destacar: extração de informações de um SGBD para uso em Web Services, geração de relatórios de dados armazenados em um Banco de Dados XML, seleção e transformação de dados XML para publicação na Web e divisão de documentos XML que representam transações múltiplas.

A estrutura básica da maioria das consultas XQuery é a expressão FLWOR.

FLWOR significa "for, let, where, order by, return", que são as palavras-chave utilizadas neste tipo de expressão (W3C, 2010b). A cláusula “for” especifica uma iteração por meio dos nós dos documentos XML, sendo que o restante da expressão FLWOR é avaliado uma única vez para cada iteração.

A cláusula “let” é usada para definir o valor de uma variável, que tem o nome sempre precedido pelo símbolo de cifrão ($). A cláusula “where” especifica restrições aos nós que estão sendo iterados pela cláusula “for”.

A cláusula “order by” classifica os resultados da consulta. A cláusula “return” especifica o formato e os valores do resultado da consulta.

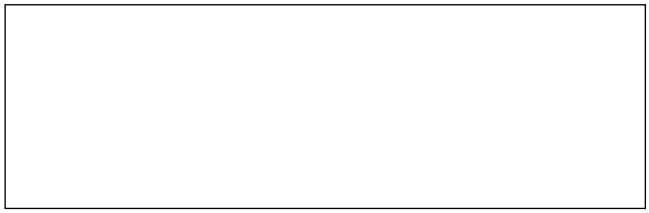


Figura 62 – Exemplo de consulta utilizando XQuery.

Fonte: W3C (2010b).

[11] World Wide Web Consortium: Consórcio internacional, liderado por Tim Berners-Lee (inventor da WWW) e pelo diretor executivo Jeffrey Jaffe, onde organizações filiadas, uma equipe de especialistas e a comunidade em geral, trabalham juntos em um processo para desenvolver padrões de qualidade para a Web (W3C, 2012).

[12] XPath - linguagem para endereçamento de elementos de documentos XML desenvolvida para ser utilizada por diversos *parsers* (W3C, 2010a).

APÊNDICES

Apêndice A – Código-fonte da ferramenta OopExtract

(continua)

Classe OOPExtractExe:

```
package br.ufsm.OopExtract.principal;

import java.io.File;
import com.hp.hpl.jena.ontology.OntModel;
import br.ufsm.OopExtract.database.Conexao;
import br.ufsm.OopExtract.Ontologia.CarregarOntologia;

public class OOPExtractExe {
	public static void main(String[] args) {
		String dir = "d:/workspace/ExemplosJava";
		String fonte="";
		CarregarOntologia ROnto = new CarregarOntologia();
		OntModel ontoop = ROnto.carregar();
		Conexao dbcon = new Conexao();
		dbcon.ConectarFb();
		dbcon.ExcluirDadosFb();
		VisitarArquivo visitor = new VisitarArquivo();
		File diretorio = new File(dir);
		File fList[] = diretorio.listFiles();
		for ( int i = 0; i < fList.length; i++ ){
			fonte=fList[i].getPath();
			System.out.println(fonte);
			visitor.Visitar(fonte,ontoop,dbcon);
		}
		dbcon.DesconectarFb();
	}
}
```

Classe AtribuiArquivo:

```
package br.ufsm.OopExtract.principal;

import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class AtribuirArquivo {
	public static String BuscarArquivo (String file) {
		byte[] buffer = new byte[(int) new File(file).length()];
		BufferedInputStream f = null;
		try {
			f = new BufferedInputStream(new FileInputStream(file));
			f.read(buffer);
		} catch (IOException e) {
			e.printStackTrace();
		} finally {
			if (f != null)
				try {
					f.close();
				} catch (IOException e) {/**/}
		}
		return new String(buffer);
	}
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

```
 public void ExcluirArquivo(String caminho){
     File file = new File(caminho);
     file.delete();
 }
}
```

Classe VisitaArquivo:

```
package br.ufsm.OopExtract.principal;

import java.util.HashMap;
import org.eclipse.jdt.core.dom.AST;
import org.eclipse.jdt.core.dom.ASTParser;
import org.eclipse.jdt.core.dom.CompilationUnit;
import com.hp.hpl.jena.ontology.OntModel;
import br.ufsm.OopExtract.database.Conexao;
import br.ufsm.OopExtract.oopvisitor.ObjectVisitor;

public class VisitarArquivo {
 public void Visitar(String fonte, OntModel m, Conexao c){
        String arq = AtribuirArquivo.BuscarArquivo(fonte);
        ASTParser p = ASTParser.newParser(AST.JLS3);
        p.setCompilerOptions(new HashMap());
        p.setKind(ASTParser.K_COMPILATION_UNIT);
        p.setResolveBindings(true);
        p.setBindingsRecovery(true);
        p.setSource(arq.toCharArray());
        ObjectVisitor visitor = new ObjectVisitor(m,c);
        CompilationUnit compUnit = (CompilationUnit) p.createAST(null);
        compUnit.accept(visitor);
 }
}
```

Classe Conexao:

```
package br.ufsm.OopExtract.database;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class Conexao {
 public Connection con;
 public Boolean conectado=false;
 public void ConectarFb() {
        String msg="";
        if (conectado==false){
              try {
                    Class.forName("org.firebirdsql.jdbc.FBDriver");
                    con =
DriverManager.getConnection("jdbc:firebirdsql:localhost:D:/Aulas/Mestrado/"+

"Implementacao/ConsultaSQL/CONSULTA.FDB",
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

```
"SYSDBA","masterkey");
                        con.setAutoCommit(true);
                        msg="Conectado Firebird!";
                        conectado=true;
                  }
                  catch(Exception e){
                        conectado=false;
                        msg = "Houve um erro de conexão Firebird: " +
e.getMessage();
                  }
            }
            System.out.println(msg);
      }
      public void DesconectarFb() {

            if (conectado){
                  try {
                        con.close();
                  } catch (Exception e) {/**/}
            }
      }
      public void ExcluirDadosFb(){
            Statement statement = null;
            String msgf = "";
            if (conectado){
                  try {
                        String sqlins = "execute procedure STP_EXCLUIR_DADOS;";
                        statement = con.createStatement();
                        statement.executeUpdate(sqlins);
                        statement.close();
                        msgf="Firebird - Exclusão de Dados Ok!";
                  }
                  catch(Exception e){
                        msgf = "Firebird - Houve um erro ao excluir dados: " +
e.getMessage();
                  }
            }
            else{
                  msgf="Firebird - DESCONECTADO!";
            }
            System.out.println(msgf);
      }
                                    }
```

Classe Insercao:

```
package br.ufsm.OopExtract.database;

import java.sql.ResultSet;
import java.sql.Statement;
import br.ufsm.OopExtract.database.Conexao;
import br.ufsm.OopExtract.estrutura.clAnotacao;
import br.ufsm.OopExtract.estrutura.clAtributo;
import br.ufsm.OopExtract.estrutura.clClasse;
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

```
import br.ufsm.OopExtract.estrutura.clExcecao;
import br.ufsm.OopExtract.estrutura.clMetodo;

public class Insercao {
 public int InserirClasse(clClasse classe,Conexao con){
        int idpk=-1;
        Statement statement = null;
        ResultSet result = null;
        String msg = "", msgvlr="";
        if (classe.getImplementacoes().length()>0){
              msgvlr=classe.getModificador()+" "+classe.getTipo()+"
"+classe.getNome()+" implements "+classe.getImplementacoes()+"
"+Integer.toString(classe.getIdsuperclasse());
        }
        else{
              msgvlr=classe.getModificador()+" "+classe.getTipo()+"
"+classe.getNome()+" "+Integer.toString(classe.getIdsuperclasse());
        }
        if (con.conectado){
              try {
                    String sqlins = "execute procedure
stp_insere_classe('"+classe.getNome()+"','"+classe.getModificador()+"','"+cla
sse.getTipo()+"','"+


classe.getImplementacoes()+"','"+classe.getNomesuperclasse()+"',"+classe.getI
dsuperclasse()+");";
                    String sqlpk = "select gen_id(GEN_CLASSES_ID,0) from
rdb$database";
                    statement = con.con.createStatement();
                    statement.executeUpdate(sqlins);
                    result=statement.executeQuery(sqlpk);
                    result.next();
                    idpk=result.getInt(1);
                    statement.close();
                    msg="Firebird - Classe Ok! ("+msgvlr+")";
              }
              catch(Exception e){
                    msg = "Firebird - Houve um erro ao inserir Classe
("+msgvlr+"): " + e.getMessage();
              }
        }
        else{
              msg="Firebird - DESCONECTADO!";
        }
        System.out.println(msg);
        return idpk;
 }
 public int InserirAtributo(clAtributo atributo,Conexao con){
        int idpk=-1;
        Statement statement = null;
        ResultSet result = null;
        String msg = "", msgvlr="";
        msgvlr=atributo.getModificador()+" "+atributo.getTipo()+"
"+atributo.getNome()+" "+Integer.toString(atributo.getIdclasse());
        if (con.conectado){
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

```
                    try {
                          String sqlc = "execute procedure stp_insere_atributo
('"+atributo.getNome()+"','"+atributo.getModificador()+"','"+atributo.getTipo()+"'
,"+atributo.getIdclasse()+");";
                          String sqlpk = "select gen_id(GEN_ATRIBUTOS_ID,0) from
rdb$database";
                          statement = con.con.createStatement();
                          statement.executeUpdate(sqlc);
                          result=statement.executeQuery(sqlpk);
                          result.next();
                          idpk=result.getInt(1);
                          statement.close();
                          msg="Firebird - Atributo Ok! ("+msgvlr+")";
                    }
                    catch(Exception e){
                          msg = "Firebird - Houve um erro ao inserir Atributo
("+msgvlr+"): " + e.getMessage();
                    }
             }
             else{
                    msg="Firebird - DESCONECTADO!";
             }
             System.out.println(msg);
             return idpk;
      }
      public int InserirMetodo(clMetodo metodo,Conexao con){
             int idpk=-1;
             Statement statement = null;
             ResultSet result = null;
             String msg = "", msgvlr="";
             msgvlr=metodo.getModificador()+" "+metodo.getRetorno()+"
"+metodo.getNome()+" ("+metodo.getParametros()+")
"+Integer.toString(metodo.getIdclasse());
             if (con.conectado){
                    try {
                          String sqlins = "execute procedure stp_insere_metodo
('"+metodo.getNome()+"','"+metodo.getModificador()+"','"+metodo.getRetorno()+

"','"+metodo.getParametros()+"',"+metodo.getIdclasse()+");";
                          String sqlpk = "select gen_id(GEN_METODOS_ID,0) from
rdb$database";
                          statement = con.con.createStatement();
                          statement.executeUpdate(sqlins);
                          result=statement.executeQuery(sqlpk);
                          result.next();
                          idpk=result.getInt(1);
                          statement.close();
                          msg="Firebird - Método Ok! ("+msgvlr+")";
                    }
                    catch(Exception e){
                          msg = "Firebird - Houve um erro ao inserir Método
("+msgvlr+"): " + e.getMessage();
                    }
             }
             else{
                    msg="Firebird - DESCONECTADO!";
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

```
        }
        System.out.println(msg);
        return idpk;
 }
 public int InserirExcecao(clExcecao excecao,Conexao con){
        int idpk=-1;
        Statement statement = null;
        ResultSet result = null;
        String msg = "", msgvlr="";
        msgvlr=excecao.getNome()+" "+Integer.toString(excecao.getIdmetodo());
        if (con.conectado){
              try {
                    String sqlc = "execute procedure
stp_insere_excecao('"+excecao.getNome()+"',"+excecao.getIdmetodo()+");";

                    String sqlpk = "select gen_id(GEN_EXCECOES_ID,0) from
rdb$database";
                    statement = con.con.createStatement();
                    statement.executeUpdate(sqlc);
                    result=statement.executeQuery(sqlpk);
                    result.next();
                    idpk=result.getInt(1);
                    statement.close();
                    msg="Firebird - Exceção Ok! ("+msgvlr+")";
              }
              catch(Exception e){
                    msg = "Firebird - Houve um erro ao inserir Exceção
("+msgvlr+"): " + e.getMessage();
              }
        }
        else{
              msg="Firebird - DESCONECTADO!";
        }
        System.out.println(msg);
        return idpk;
 }
 public int InserirAnotacao(clAnotacao anotacao,Conexao con){
        int idpk=-1;
        Statement statement = null;
        ResultSet result = null;
        String msg = "", msgvlr="";
        msgvlr=anotacao.getNome()+"
"+Integer.toString(anotacao.getIdclasse());
        if (con.conectado){
              try {
                    String sqlc = "execute procedure
stp_insere_anotacao('"+anotacao.getNome()+"',"+anotacao.getIdclasse()+");";

                    String sqlpk = "select gen_id(GEN_ANOTACOES_ID,0) from
rdb$database";
                    statement = con.con.createStatement();
                    statement.executeUpdate(sqlc);
                    result=statement.executeQuery(sqlpk);
                    result.next();
                    idpk=result.getInt(1);
                    statement.close();
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

```
                        msg="Firebird - Anotação Ok! ("+msgvlr+")";
                  }
                  catch(Exception e){
                        msg = "Firebird - Houve um erro ao inserir Anotação
("+msgvlr+"):" + e.getMessage();
                  }
            }
            else{
                  msg="Firebird - DESCONECTADO!";
            }
            System.out.println(msg);
            return idpk;
      }
}
```

Classe Busca:

```
package br.ufsm.OopExtract.database;

import java.sql.ResultSet;
import java.sql.Statement;
import br.ufsm.OopExtract.database.Conexao;

public class Busca {
            public int BuscarClasse(String nomeclasse, Conexao con){
                  String msgf="";
                  int id=-1;
                  if (con.conectado){
                        try {
                              String sqlc = "select max(id_classe) from CLASSES
"+
                                          "where NOME= '"+nomeclasse+"';";

                              Statement statement = con.con.createStatement();

                              ResultSet resultado =
statement.executeQuery(sqlc);
                              id=resultado.getInt(1);
                              statement.close();
                              msgf="Firebird - Busca ok!";
                        }
                        catch(Exception e){
                              id=0;
                              msgf = "Firebird - Houve um erro ao recuperar a
classe através do nome: " + e.getMessage();
                  }
                  }
                  else{
                        msgf="Firebird - DESCONECTADO!";
                  }
            System.out.println(msgf);
            return id;
            }
}
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

Classe clAnotacao:

```
package br.ufsm.OopExtract.estrutura;

public class clAnotacao {


	public int getIdanotacao() {
		return idanotacao;
	}
	public void setIdanotacao(int idanotacao) {
		this.idanotacao = idanotacao;
	}
	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = validacao.removerchar(nome);
	}
	public int getIdclasse() {
		return idclasse;
	}
	public void setIdclasse(int idclasse) {
		this.idclasse = idclasse;
	}

	private clValidacao validacao = new clValidacao();
	private String nome;
	private int idclasse;
	private int idanotacao;
}
```

Classe clAtributo:

```
package br.ufsm.OopExtract.estrutura;


public class clAtributo {

	public int getIdatributo() {
		return idatributo;
	}
	public void setIdatributo(int idatributo) {
		this.idatributo = idatributo;
	}
	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = validacao.removerchar(nome);
	}
	public String getModificador() {
		return modificador;
	}
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

```
	public void setModificador(String modificador) {
		this.modificador = validacao.removerchar(modificador);
	}
	public String getTipo() {
		return tipo;
	}
	public void setTipo(String tipo) {
		this.tipo = validacao.removerchar(tipo);
	}
	public int getIdclasse() {
		return idclasse;
	}
	public void setIdclasse(int idclasse) {
		this.idclasse = idclasse;
	}

	private clValidacao validacao = new clValidacao();
	private String nome;
	private String modificador;
	private String tipo;
	private int idclasse;
	private int idatributo;
}
```

Classe clClasse:

```
package br.ufsm.OopExtract.estrutura;

public class clClasse {

	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = validacao.removerchar(nome);
	}
	public String getModificador() {
		return modificador;
	}
	public void setModificador(String modificador) {
		this.modificador = validacao.removerchar(modificador);
	}
	public String getTipo() {
		return tipo;
	}
	public void setTipo(String tipo) {
		this.tipo = validacao.removerchar(tipo);
	}
	public String getImplementacoes() {
		return implementacoes;
	}
	public void setImplementacoes(String implementacoes) {
		this.implementacoes = validacao.removerchar(implementacoes);
	}
	public String getNomesuperclasse() {
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

```
		return nomesuperclasse;
	}
	public void setNomesuperclasse(String nomesuperclasse) {
		this.nomesuperclasse = validacao.removerchar(nomesuperclasse);
	}
	public int getIdclasse() {
		return idclasse;
	}
	public void setIdclasse(int idclasse) {
		this.idclasse = idclasse;
	}
	public int getIdsuperclasse() {
		return idsuperclasse;
	}
	public void setIdsuperclasse(int idsuperclasse) {
		this.idsuperclasse = idsuperclasse;
	}
	private clValidacao validacao = new clValidacao();
	private String nome;
	private String modificador;
	private String tipo;
	private String implementacoes;
	private String nomesuperclasse;
	private int idclasse;
	private int idsuperclasse;
}
```

Classe clExecao:

```
package br.ufsm.OopExtract.estrutura;

public class clExcecao {

	public int getIdexcecao() {
		return idexcecao;
	}
	public void setIdexcecao(int idexcecao) {
		this.idexcecao = idexcecao;
	}
	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = validacao.removerchar(nome);
	}
	public int getIdmetodo() {
		return idmetodo;
	}
	public void setIdmetodo(int idmetodo) {
		this.idmetodo = idmetodo;
	}

	private clValidacao validacao = new clValidacao();
	private String nome;
	private int idmetodo;
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

```
private int idexcecao;

}
```

Classe clMetodo:

```
package br.ufsm.OopExtract.estrutura;

public class clMetodo {

	public int getIdmetodo() {
		return idmetodo;
	}
	public void setIdmetodo(int idmetodo) {
		this.idmetodo = idmetodo;
	}
	public String getRetorno() {
		return retorno;
	}
	public void setRetorno(String retorno) {
		this.retorno = retorno;
	}
	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = validacao.removerchar(nome);
	}
	public String getModificador() {
		return modificador;
	}
	public void setModificador(String modificador) {
		this.modificador = validacao.removerchar(modificador);
	}
	public String getParametros() {
		return parametros;
	}
	public void setParametros(String parametros) {
		this.parametros = validacao.removerchar(parametros);
	}
	public int getIdclasse() {
		return idclasse;
	}
	public void setIdclasse(int idclasse) {
		this.idclasse = idclasse;
	}
	private clValidacao validacao = new clValidacao();
	private String nome;
	private String modificador;
	private String retorno;
	private String parametros;
	private int idclasse;
	private int idmetodo;
}
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

Classe clValidacao:

```
package br.ufsm.OopExtract.estrutura;

public class clValidacao {
 public String removerchar(String str){
        str=str.replace("'", "");
        str=str.replace("´", "");
        str=str.replace("]", "");
        str=str.replace("[", "");
        return str;
 }
 public boolean validarstr(String str){
        if ((str!="") && (str!=null) && (str!="[]")){
               return true;
        }
        else{
               return false;
        }
 }
}
```

Classe ObjectVisitor:

```
package br.ufsm.OopExtract.oopvisitor;

import java.util.List;
import org.eclipse.jdt.core.dom.ASTVisitor;
import org.eclipse.jdt.core.dom.FieldDeclaration;
import org.eclipse.jdt.core.dom.MethodDeclaration;
import org.eclipse.jdt.core.dom.TypeDeclaration;
import org.eclipse.jdt.core.dom.VariableDeclarationFragment;
import com.hp.hpl.jena.ontology.OntModel;
import br.ufsm.OopExtract.database.*;
import br.ufsm.OopExtract.estrutura.*;
import br.ufsm.OopExtract.Ontologia.InserirOntologia;

public class ObjectVisitor extends ASTVisitor{
 public clAnotacao anotacao = new clAnotacao();
 public clAtributo atributo = new clAtributo();
 public clExcecao excecao = new clExcecao();
 public clMetodo metodo = new clMetodo();
 public clValidacao validacao = new clValidacao();
 public InserirOntologia ROnto = new InserirOntologia();
 public int idclasse, idmetodo;
 public Conexao con;
 public OntModel ontom;
 public ObjectVisitor(OntModel m, Conexao c) {
        ontom=m;
        con=c;
 }

 public boolean visit(TypeDeclaration node) { //Navegação em nodos tipo
classe e interface.
        clClasse classe = new clClasse();
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

```
            Busca buscar = new Busca();
            Insercao inserir = new Insercao();
            String nomeclasse="", nomesuperclasse="", modificadores="",
tipoobjeto = "class", implementacoes="", msgvlr="", msg="";
            int idsuper=0;

            for (int i=0; i<node.modifiers().size(); i++) {
                  if (node.modifiers().get(i).toString().contains("@")==false){

      modificadores=modificadores+node.modifiers().get(i).toString();
                        if (i<node.modifiers().size()-1){
                              modificadores=modificadores+" ";
                        }
                  }
            }
            nomeclasse=node.getName().toString();
            if (validacao.validarstr(nomeclasse)){
                    if (node.isInterface()){
                          tipoobjeto = "interface";
                          implementacoes=node.getClass().toString();
                    }
                    try {
                          nomesuperclasse = node.getSuperclassType().toString();
                          idsuper = buscar.BuscarClasse(nomesuperclasse, con);
                          msgvlr=nomesuperclasse+" "+Integer.toString(idsuper);


                    } catch (Exception e) {
                        msg = "Firebird - Houve um erro ao buscar superclasse
("+msgvlr+"): " + e.getMessage();
                        idsuper=0;
                    }
                    if (validacao.validarstr(nomeclasse)){
                          classe.setNome(nomeclasse);
                          classe.setModificador(modificadores);
                          classe.setTipo(tipoobjeto);
                          classe.setImplementacoes(implementacoes);
                          classe.setIdsuperclasse(idsuper);
                          classe.setNomesuperclasse(nomesuperclasse);
                          classe.setIdclasse(0);
                          idclasse=inserir.InserirClasse(classe, con);
                          classe.setIdclasse(idclasse);
                          ROnto.InserirOntoClasse(classe,ontom);
                    }
                    else{
                          idclasse=0;
                    }
            }
            return super.visit(node);
      }
      @Override
      public void endVisit(TypeDeclaration node) { //Final da navegação em classes
ou interfaces.
            super.endVisit(node);
      }
      @Override
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

```
 public boolean visit(FieldDeclaration node) { //Navegação em nodos tipo
atributo de classe.
        clAtributo atributo = new clAtributo();
        clAnotacao anotacao = new clAnotacao();
        Insercao inserir = new Insercao();
        String nome="", modificadores="", tipoatributo = "", nomeanotacao="",
msgvlr="";
        int idatributo=0, idanotacao=0;

        if (idclasse>0){
              try {
                    tipoatributo=node.getType().toString();

                    for (int i=0; i<node.modifiers().size(); i++) {
                          nomeanotacao=node.modifiers().get(i).toString();
                          if (nomeanotacao.contains("@")){
                                anotacao.setIdclasse(idclasse);
                                anotacao.setNome(nomeanotacao);
                                anotacao.setIdanotacao(0);

 idanotacao=inserir.InserirAnotacao(anotacao, con);

                                anotacao.setIdanotacao(idanotacao);
                                ROnto.InserirOntoAnotacao(anotacao,ontom);

                          }
                          else{
                                modificadores=modificadores+nomeanotacao;
                                if (i<node.modifiers().size()-1){
                                      modificadores=modificadores+" ";
                                }
                          }

                    }
                    List<VariableDeclarationFragment> lista =
node.fragments();
                    for (VariableDeclarationFragment tipo : lista) {
                          nome=tipo.getName().toString();

                          if (validacao.validarstr(nome)){

                                atributo.setIdclasse(idclasse);
                                atributo.setModificador(modificadores);
                                atributo.setNome(nome);
                                atributo.setTipo(tipoatributo);
                                atributo.setIdatributo(0);


 idatributo=inserir.InserirAtributo(atributo,con);

                                atributo.setIdatributo(idatributo);
                                ROnto.InserirOntoAtributo(atributo,ontom);

                          }
                    }
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

```
                  } catch (Exception e) {/**/}

            }
            return super.visit(node);
      }
      @Override
      public void endVisit(FieldDeclaration node) { //Final da navegação em
atributos de classe.
            super.endVisit(node);
      }
      @Override
      public boolean visit(MethodDeclaration node) { //Navegação em nodos tipo
método.
            clMetodo metodo = new clMetodo();
            clExcecao excecao = new clExcecao();
            Insercao inserir = new Insercao();
            String
vazio="void",retorno="",modificadores="",param="",nomeexcecao="",nome="";
            int idex=0,idmet=0;

            if (idclasse>0){
                  try {
                        retorno=node.getReturnType2().toString();
                        nome=node.getName().toString();

                        for (int i=0; i<node.modifiers().size(); i++) {
                              if
(node.modifiers().get(i).toString().contains("@")==false){

      modificadores=modificadores+node.modifiers().get(i).toString();
                                    if (i<node.modifiers().size()-1){
                                          modificadores=modificadores+" ";
                                    }
                              }
                        }
                        for (int i=0; i<node.parameters().size(); i++) {
                              if
(node.parameters().get(i).toString().contains("@")==false){

      param=param+node.parameters().get(i).toString();
                              }
                              if (i<node.parameters().size()-1){
                                    param=param+",";
                              }
                        }
                        if (validacao.validarstr(nome)){

                              metodo.setIdclasse(idclasse);
                              metodo.setModificador(modificadores);
                              metodo.setNome(nome);
                              metodo.setParametros(param);
                              metodo.setRetorno(retorno);
                              metodo.setIdmetodo(0);

                              idmetodo=inserir.InserirMetodo(metodo,con);
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

```
                                    metodo.setIdmetodo(idmetodo);
                                    ROnto.InserirOntoMetodo(metodo,ontom);

                                    for (int i=0;
i<node.thrownExceptions().size(); i++) {

 nomeexcecao=node.thrownExceptions().get(i).toString();

                                          if
(validacao.validarstr(nomeexcecao)) {
                                                excecao.setIdmetodo(idmetodo);
                                                excecao.setNome(nomeexcecao);
                                                excecao.setIdexcecao(0);

 idex=inserir.InserirExcecao(excecao,con);
                                                excecao.setIdexcecao(idex);

 ROnto.InserirOntoExcecao(excecao,ontom);
                                          }
                              }
                        }
                  }catch (Exception e) {/**/}
        }
        return super.visit(node);
 }
 @Override
 public void endVisit(MethodDeclaration node) { //Final da navegação em
métodos.
        super.endVisit(node);     }
}
```

Classe CarregarOntologia:

```
package br.ufsm.OopExtract.Ontologia;

import java.io.File;
import java.io.FileOutputStream;
import br.ufsm.OopExtract.principal.AtribuirArquivo;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntModelSpec;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.ModelMaker;

public class CarregarOntologia {
 private String urib ="file:d:/workspace/OOPN2.owl";
 private String urlb = "http://www.owl-
ontologies.com/Ontology1338318212.owl";
 private String pasta = "d:/workspace/";
 private String arquivo = "ConsultaOOP.owl";
 public String getUrib() {
        return urib;
 }
 public String getUrlb() {
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

```
		return urlb;
	}
	public String getPasta() {
		return pasta;
	}
	public String getArquivo() {
		return arquivo;
	}
	public OntModel carregar(){
		String msg="Ontologia - Leitura ok!";
		OntModelSpec s = new OntModelSpec(OntModelSpec.OWL_MEM);
		OntModel m = ModelFactory.createOntologyModel(s, null);
		try{
			m.read (urib, "RDF/XML");
			m.write(System.out, "RDF/XML-ABBREV");
		}
		catch(Exception e){
			msg = "Ontologia - Houve um erro na leitura: " +
e.getMessage();
		}
		System.out.println(msg);
		return m;
	}
	public void CriarOntologia(){
		String msg="Ontologia - Criação ok!";
		AtribuirArquivo arq = new AtribuirArquivo();
		arq.ExcluirArquivo(pasta+arquivo);
		try {
			ModelMaker maker = ModelFactory.createFileModelMaker(pasta);
			Model base = maker.createModel(arquivo, false);
			OntModelSpec spec = new OntModelSpec(OntModelSpec.OWL_DL_MEM);
			spec.setImportModelMaker(maker);
			ModelFactory.createOntologyModel(spec, base);
		} catch (Exception e) {
			msg = "Ontologia - Houve um erro na criação: " +
e.getMessage();
		}
		System.out.println(msg);
	}
	public void SalvarOntologia(OntModel m){
		String msg="Ontologia - Salvar ok!";
		try {
			File file= new File(pasta+arquivo);
			m.write(new FileOutputStream(file));
		} catch (Exception e) {
			msg = "Ontologia - Houve um erro ao salvar: " + e.getMessage();
		}
		System.out.println(msg);
	}
}
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

Classe InserirOntologia:

```
package br.ufsm.OopExtract.Ontologia;

import br.ufsm.OopExtract.estrutura.clAnotacao;
import br.ufsm.OopExtract.estrutura.clAtributo;
import br.ufsm.OopExtract.estrutura.clClasse;
import br.ufsm.OopExtract.estrutura.clExcecao;
import br.ufsm.OopExtract.estrutura.clMetodo;

import com.hp.hpl.jena.ontology.DatatypeProperty;
import com.hp.hpl.jena.ontology.Individual;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntProperty;
import com.hp.hpl.jena.rdf.model.Resource;

public class InserirOntologia {
 public CarregarOntologia ontocar = new CarregarOntologia();
 public void AtualizarOnto(OntModel m){
       CarregarOntologia exonto = new CarregarOntologia();
       exonto.CriarOntologia();
       exonto.SalvarOntologia(m);
 }
 public void InserirOntoClasse(clClasse classe,OntModel m){
       String id,nome,modif,tipo,impl,nomesuper,idsuper,msg="Ontologia -
classe Ok!";
       if (classe.getIdclasse()>0){
             id=Integer.toString(classe.getIdclasse());
             nome=classe.getNome();
             modif=classe.getModificador();
             tipo=classe.getTipo();
             impl=classe.getImplementacoes();
             nomesuper=classe.getNomesuperclasse();
             idsuper=Integer.toString(classe.getIdsuperclasse());

             String urlb = ontocar.getUrlb();
             Resource cl = m.getResource(urlb+"#CLASSES");
             String lbl = modif+" "+tipo+" "+nome+" "+id;
             Individual ind = m.createIndividual(urlb+"#CL_"+id,cl);

             ind.addLabel(lbl,"EN");
             DatatypeProperty pId = m.getDatatypeProperty(urlb+"#ID");
             ind.addProperty(pId, id,"EN");
             DatatypeProperty pNome = m.getDatatypeProperty(urlb+"#NOME");

             ind.addProperty(pNome, nome,"EN");
             DatatypeProperty pMod =
m.getDatatypeProperty(urlb+"#MODIFICADOR");
             ind.addProperty(pMod, modif,"EN");
             DatatypeProperty pImp =
m.getDatatypeProperty(urlb+"#IMPLEMENTACOES");
             ind.addProperty(pImp, impl,"EN");
             DatatypeProperty pTipo =
m.getDatatypeProperty(urlb+"#TIPO_OBJETO");
             ind.addProperty(pTipo, tipo,"EN");
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

```
                    DatatypeProperty pNomesc =
m.getDatatypeProperty(urlb+"#NOME_SUPERCLASSE");
                    ind.addProperty(pNomesc, nomesuper,"EN");
                    DatatypeProperty pIds =
m.getDatatypeProperty(urlb+"#ID_SUPERCLASSE");
                    ind.addProperty(pIds, idsuper,"EN");
                    try{
                          AtualizarOnto(m);
                    }
                    catch(Exception e){
                          msg = "Ontologia - Houve um erro ao inserir Classe: " +
e.getMessage();
                    }
                    System.out.println(msg);
             }
      }
      public void InserirOntoAtributo(clAtributo atributo,OntModel m){
             String id,nome,modif,tipo,idc,msg="Ontologia - atributo Ok!";
             if (atributo.getIdatributo()>0){
                    id=Integer.toString(atributo.getIdatributo());
                    nome=atributo.getNome();
                    modif=atributo.getModificador();
                    tipo=atributo.getTipo();
                    idc=Integer.toString(atributo.getIdclasse());
                    String urlb = ontocar.getUrlb();
                    Resource cl = m.getResource(urlb+"#ATRIBUTOS");

                    String lbl = modif+" "+tipo+" "+nome+" "+id;
                    Individual ind = m.createIndividual(urlb+"#AT_"+id,cl);

                    ind.addLabel(lbl,"EN");
                    DatatypeProperty pId = m.getDatatypeProperty(urlb+"#ID");

                    ind.addProperty(pId, id,"EN");
                    DatatypeProperty pNome = m.getDatatypeProperty(urlb+"#NOME");

                    ind.addProperty(pNome, nome,"EN");
                    DatatypeProperty pMod =
m.getDatatypeProperty(urlb+"#MODIFICADOR");
                    ind.addProperty(pMod, modif,"EN");
                    DatatypeProperty pTipo = m.getDatatypeProperty(urlb+"#TIPO");

                    ind.addProperty(pTipo, tipo,"EN");
                    DatatypeProperty pIdc =
m.getDatatypeProperty(urlb+"#ID_CLASSE");
                    ind.addProperty(pIdc, idc,"EN");
                    OntProperty ppr =
m.getObjectProperty(urlb+"#AtributodeClasse");
                    OntProperty pprinv =
m.getObjectProperty(urlb+"#PossuiAtributo");
                    Individual indinv = m.getIndividual(urlb+"#CL_"+idc);

                    ind.addProperty(ppr, indinv);
                    indinv.addProperty(pprinv, ind);
                    try{
                          AtualizarOnto(m);
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

```
                }
                catch(Exception e){
                        msg = "Ontologia - Houve um erro ao inserir Atributo: "
+ e.getMessage();
                }
                System.out.println(msg);
        }
 }
 public void InserirOntoAnotacao(clAnotacao anotacao,OntModel m){
        String id,nome,idc,msg="Ontologia - anotação Ok!";
        if (anotacao.getIdanotacao()>0){
                id=Integer.toString(anotacao.getIdanotacao());
                nome=anotacao.getNome();
                idc=Integer.toString(anotacao.getIdclasse());
                String urlb = ontocar.getUrlb();
                Resource cl = m.getResource(urlb+"#ANOTACOES");

                String lbl = nome+" "+id;
                Individual ind = m.createIndividual(urlb+"#AN_"+id,cl);

                ind.addLabel(lbl,"EN");
                DatatypeProperty pId = m.getDatatypeProperty(urlb+"#ID");

                ind.addProperty(pId, id,"EN");
                DatatypeProperty pNome = m.getDatatypeProperty(urlb+"#NOME");

                ind.addProperty(pNome, nome,"EN");
                DatatypeProperty pIdc =
m.getDatatypeProperty(urlb+"#ID_CLASSE");
                ind.addProperty(pIdc, idc,"EN");
                OntProperty ppr =
m.getObjectProperty(urlb+"#AnotacaodeClasse");
                OntProperty pprinv =
m.getObjectProperty(urlb+"#PossuiAnotacao");
                Individual indinv = m.getIndividual(urlb+"#CL_"+idc);

                ind.addProperty(ppr, indinv);
                indinv.addProperty(pprinv, ind);
                try{
                        AtualizarOnto(m);
                }
                catch(Exception e){
                        msg = "Ontologia - Houve um erro ao inserir Anotação: "
+ e.getMessage();
                }
                System.out.println(msg);
        }
 }
 public void InserirOntoMetodo(clMetodo metodo,OntModel m){
        String id,nome,modif,tipo,par,idc,msg="Ontologia - método Ok!";
        if (metodo.getIdmetodo()>0){
                id=Integer.toString(metodo.getIdmetodo());
                nome=metodo.getNome();
                modif=metodo.getModificador();
                tipo=metodo.getRetorno();
                par=metodo.getParametros();
```

Apêndice A – Código-fonte da ferramenta OopExtract

(continuação)

```
                    idc=Integer.toString(metodo.getIdclasse());
                    String urlb = ontocar.getUrlb();
                    Resource cl = m.getResource(urlb+"#METODOS");
                    String lbl = modif+" "+tipo+" "+nome+"("+par+") "+id;

                    Individual ind = m.createIndividual(urlb+"#ME_"+id,cl);

                    ind.addLabel(lbl,"EN");
                    DatatypeProperty pId = m.getDatatypeProperty(urlb+"#ID");

                    ind.addProperty(pId, id,"EN");
                    DatatypeProperty pNome = m.getDatatypeProperty(urlb+"#NOME");

                    ind.addProperty(pNome, nome,"EN");
                    DatatypeProperty pMod =
m.getDatatypeProperty(urlb+"#MODIFICADOR");
                    ind.addProperty(pMod, modif,"EN");
                    DatatypeProperty pTipo =
m.getDatatypeProperty(urlb+"#TIPO_RETORNO");
                    ind.addProperty(pTipo, tipo,"EN");
                    DatatypeProperty pParam =
m.getDatatypeProperty(urlb+"#PARAMETROS");
                    ind.addProperty(pParam, par,"EN");
                    DatatypeProperty pIdc =
m.getDatatypeProperty(urlb+"#ID_CLASSE");
                    ind.addProperty(pIdc, idc,"EN");
                    OntProperty ppr = m.getObjectProperty(urlb+"#MetododeClasse");

                    OntProperty pprinv = m.getObjectProperty(urlb+"#PossuiMetodo");
                    Individual indinv = m.getIndividual(urlb+"#CL_"+idc);

                    ind.addProperty(ppr, indinv);
                    indinv.addProperty(pprinv, ind);
                    try{
                          AtualizarOnto(m);
                    }
                    catch(Exception e){
                          msg = "Ontologia - Houve um erro ao inserir Método: " +
e.getMessage();
                    }
                    System.out.println(msg);
              }
        }
        public void InserirOntoExcecao(clExcecao excecao,OntModel m){
              String id,nome,idm,msg="Ontologia - exceção Ok!";
              if (excecao.getIdexcecao()>0){
                    id=Integer.toString(excecao.getIdexcecao());
                    nome=excecao.getNome();
                    idm=Integer.toString(excecao.getIdmetodo());
                    String urlb = ontocar.getUrlb();
                    Resource cl = m.getResource(urlb+"#EXCECOES");

                    String lbl = nome+" "+id;
                    Individual ind = m.createIndividual(urlb+"#EX_"+id,cl);

                    ind.addLabel(lbl,"EN");
```

Apêndice A – Código-fonte da ferramenta OopExtract

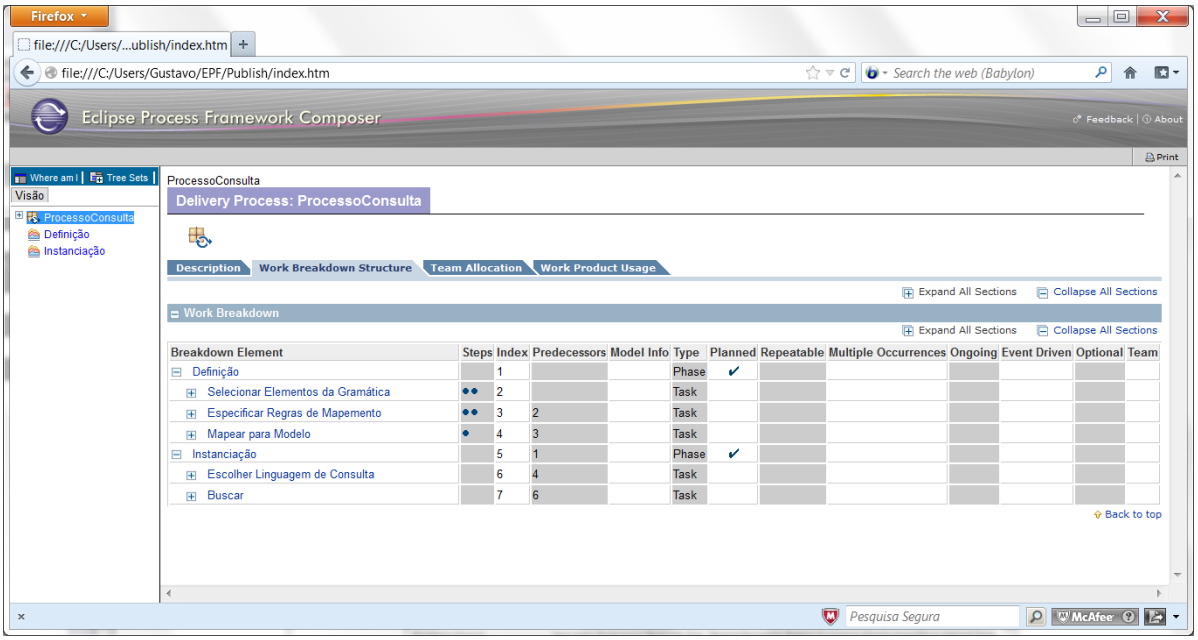
(conclusão)

```
                DatatypeProperty pId = m.getDatatypeProperty(urlb+"#ID");

                ind.addProperty(pId, id,"EN");
                DatatypeProperty pNome = m.getDatatypeProperty(urlb+"#NOME");

                ind.addProperty(pNome, nome,"EN");
                DatatypeProperty pIdm =
m.getDatatypeProperty(urlb+"#ID_METODO");
                ind.addProperty(pIdm, idm,"EN");
                OntProperty ppr = m.getObjectProperty(urlb+"#ExcecaodeMetodo");

                OntProperty pprinv =
m.getObjectProperty(urlb+"#PossuiExcecao");
                Individual indinv = m.getIndividual(urlb+"#ME_"+idm);

                ind.addProperty(ppr, indinv);
                indinv.addProperty(pprinv, ind);
                try{
                      AtualizarOnto(m);
                }
                catch(Exception e){
                      msg = "Ontologia - Houve um erro ao inserir Exceção: " +
e.getMessage();
                }
                System.out.println(msg);
        }
  }
}
```

Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer

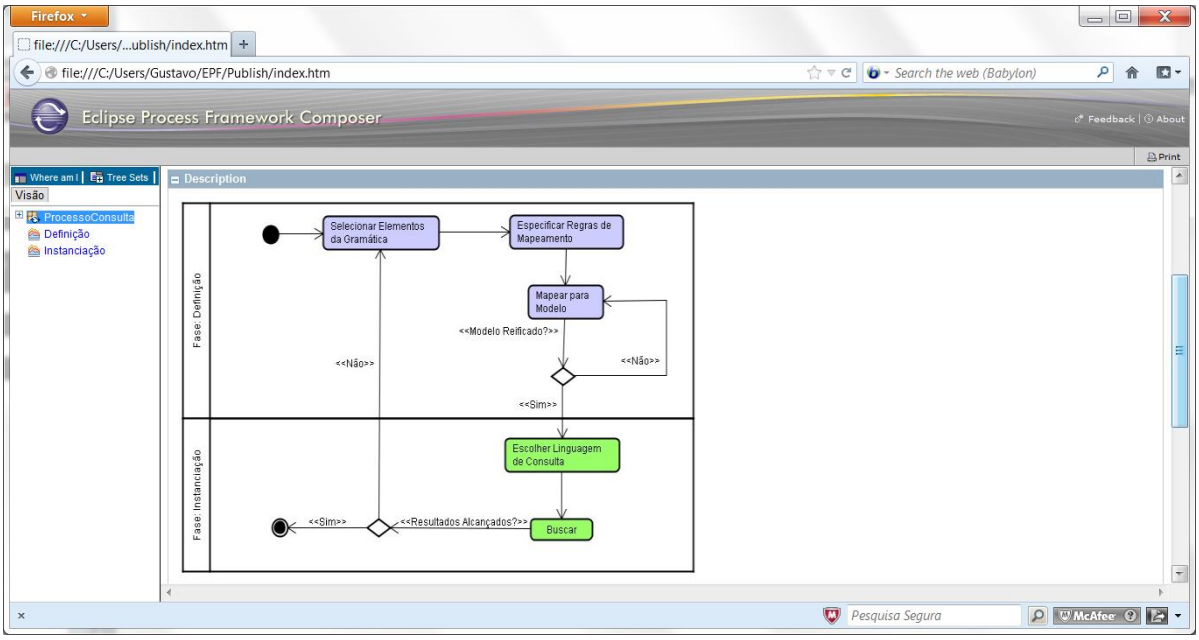
(continua)

Interface Inicial:



Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer (continuação)

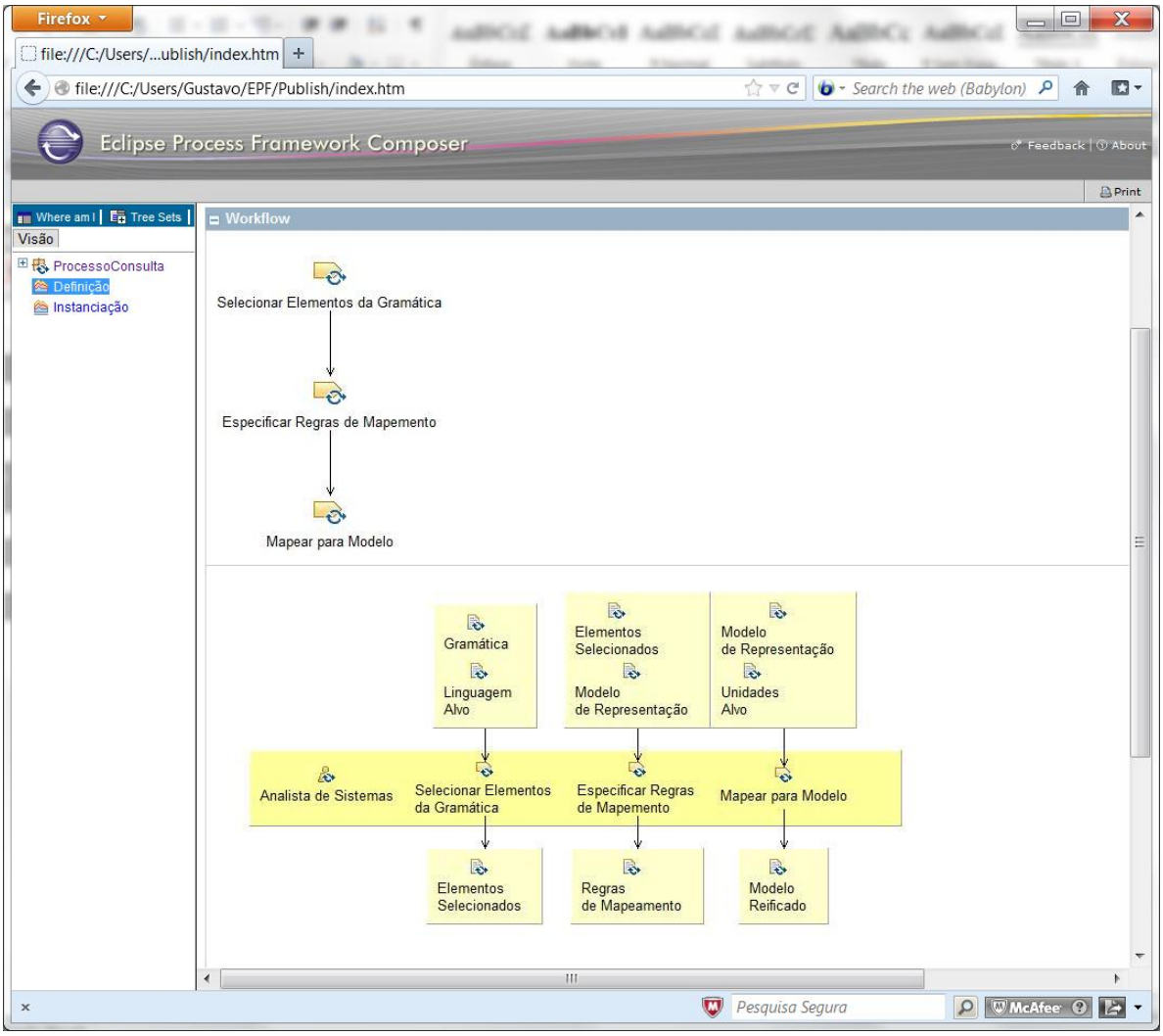
Interface Diagrama de Atividades:



Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer

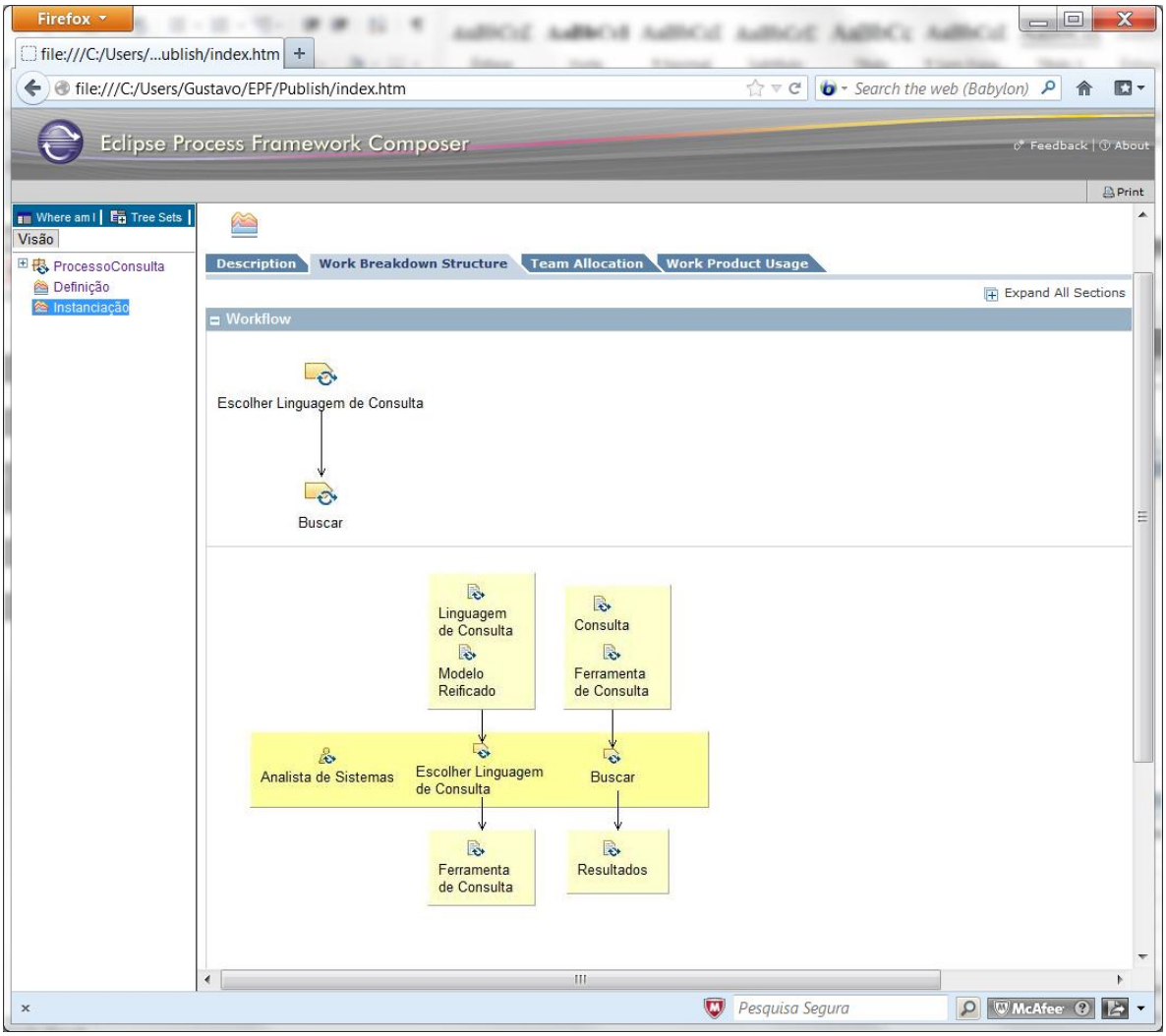
(continuação)

Interface Fase de Definição:



Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer
(continuação)

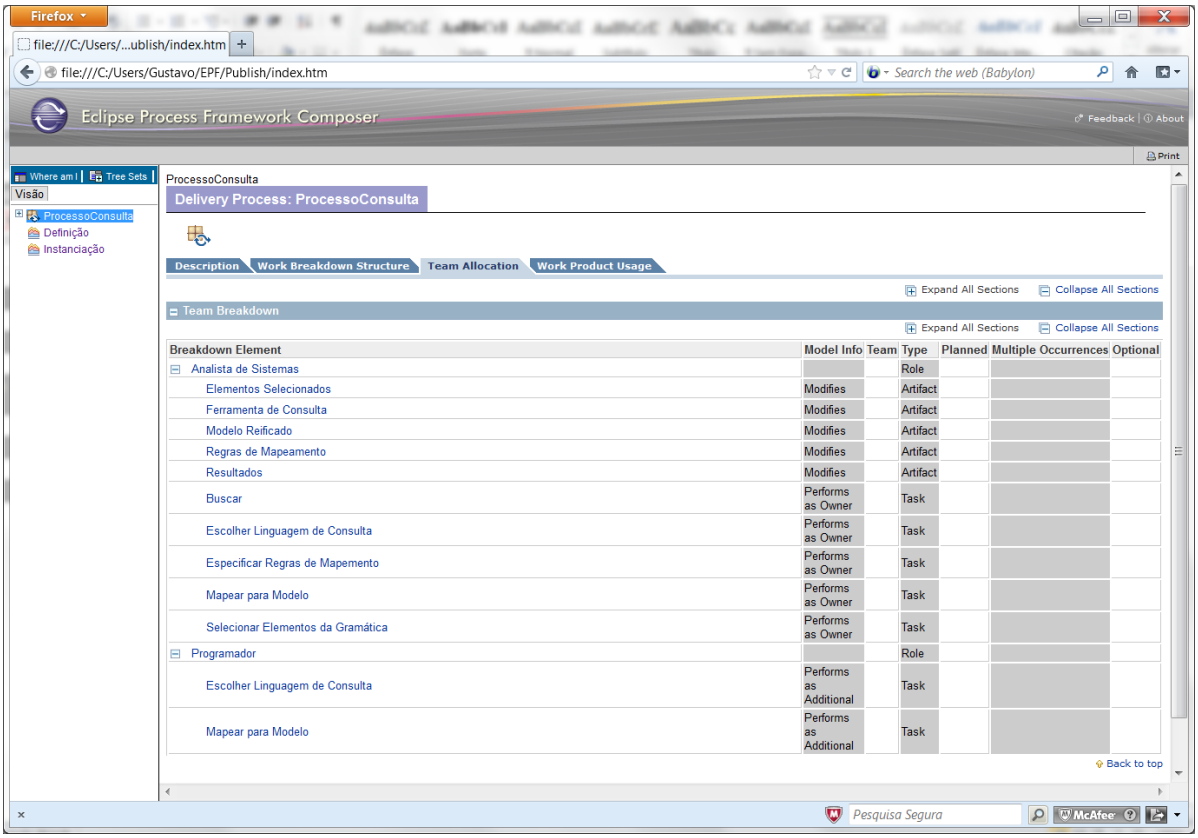
Interface Fase de Instanciação:



Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer

(continuação)

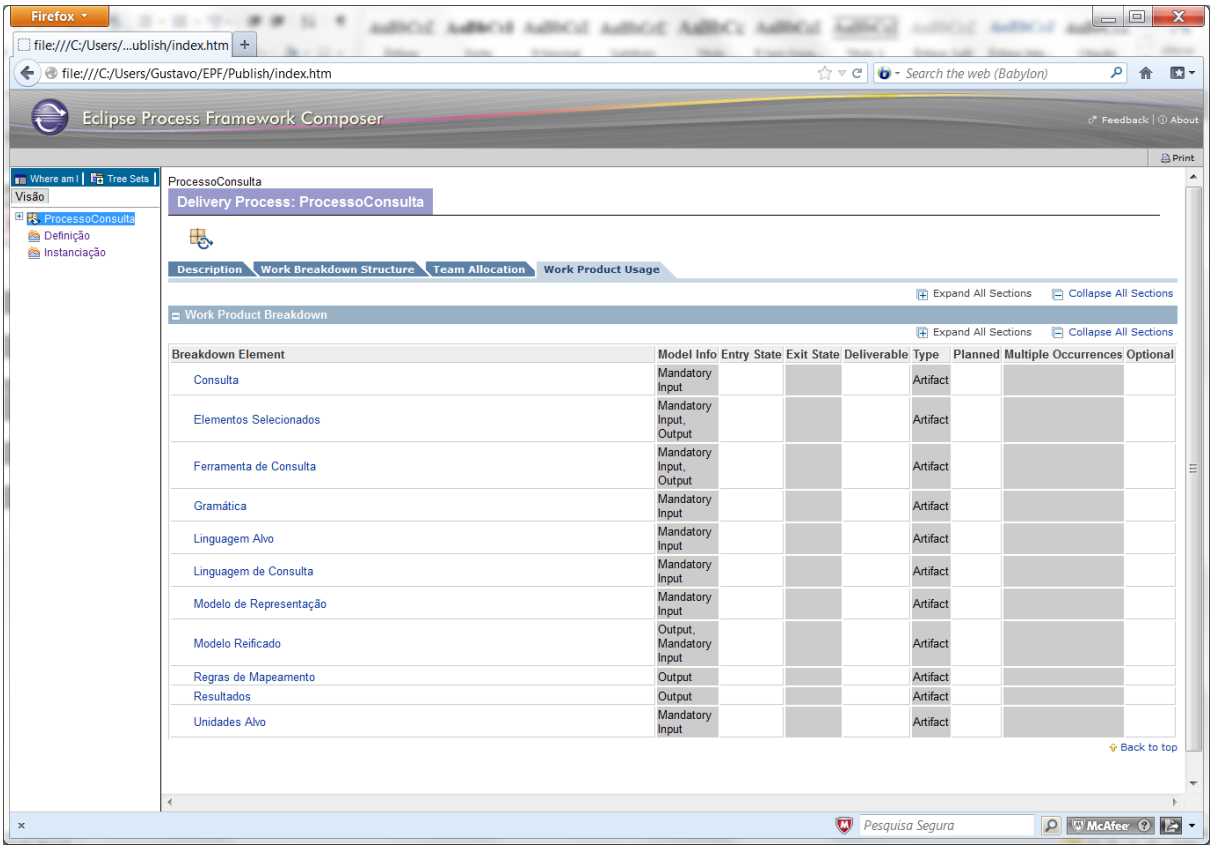
Interface Responsabilidades:



Breakdown Element	Model Info	Team	Type	Planned	Multiple Occurrences	Optional
Analista de Sistemas			Role			
Elementos Selecionados	Modifies		Artifact			
Ferramenta de Consulta	Modifies		Artifact			
Modelo Reificado	Modifies		Artifact			
Regras de Mapeamento	Modifies		Artifact			
Resultados	Modifies		Artifact			
Buscar	Performs as Owner		Task			
Escolher Linguagem de Consulta	Performs as Owner		Task			
Especificar Regras de Mapemento	Performs as Owner		Task			
Mapear para Modelo	Performs as Owner		Task			
Selecionar Elementos da Gramática	Performs as Owner		Task			
Programador			Role			
Escolher Linguagem de Consulta	Performs as Additional		Task			
Mapear para Modelo	Performs as Additional		Task			

Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer
(continuação)

Interface Artefatos:

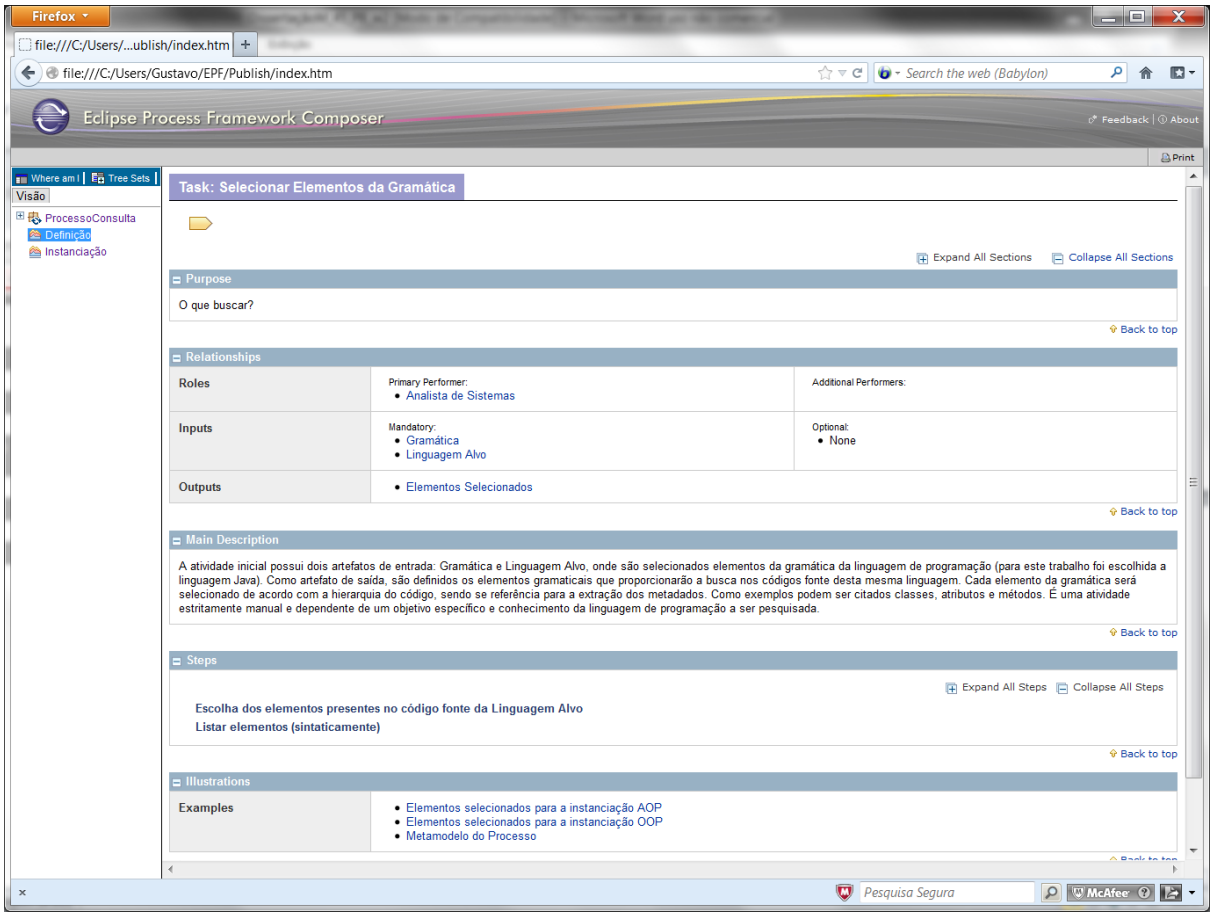


Breakdown Element	Model Info	Entry State	Exit State	Deliverable	Type	Planned	Multiple Occurrences	Optional
Consulta	Mandatory Input				Artifact			
Elementos Selecionados	Mandatory Input, Output				Artifact			
Ferramenta de Consulta	Mandatory Input, Output				Artifact			
Gramática	Mandatory Input				Artifact			
Linguagem Alvo	Mandatory Input				Artifact			
Linguagem de Consulta	Mandatory Input				Artifact			
Modelo de Representação	Mandatory Input				Artifact			
Modelo Reificado	Output, Mandatory Input				Artifact			
Regras de Mapeamento	Output				Artifact			
Resultados	Output				Artifact			
Unidades Alvo	Mandatory Input				Artifact			

Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer

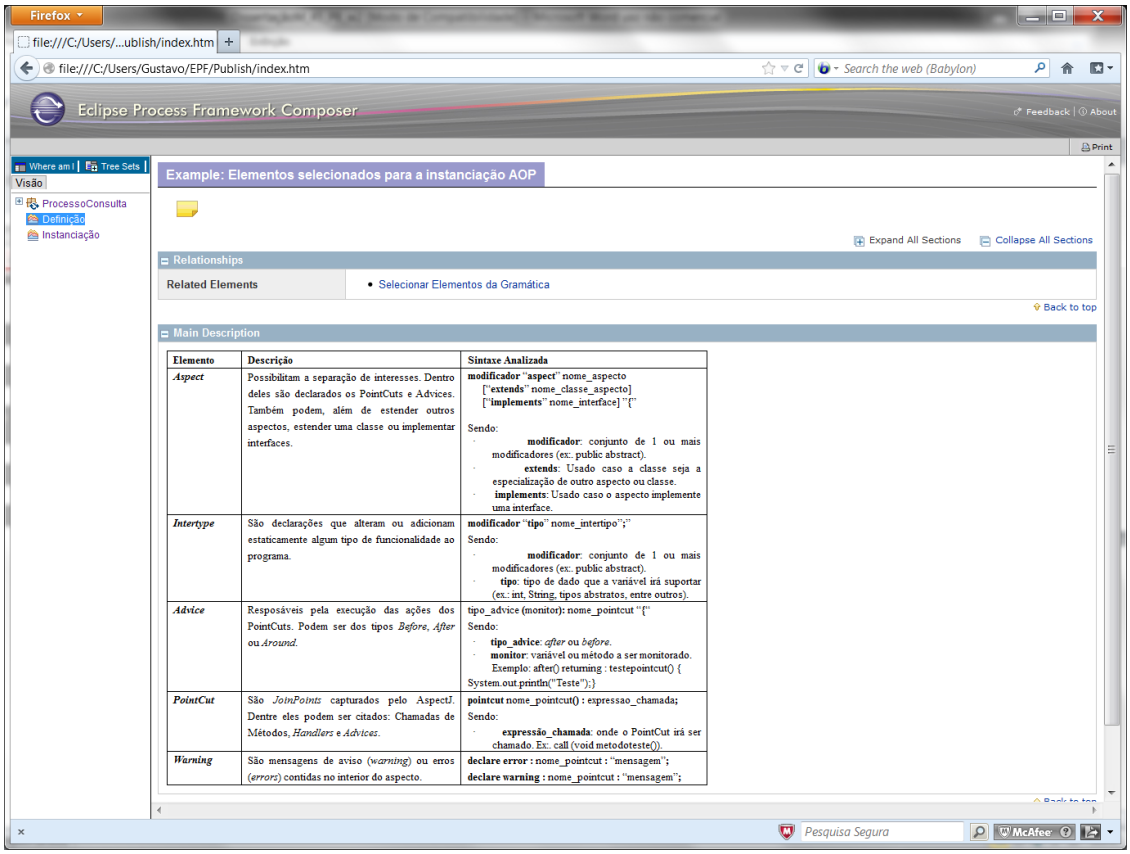
(continuação)

Interface Task (Selecionar Elementos da Gramática):



Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer (continuação)

Interface Exemplo (Elementos selecionados para a instanciação AOP):

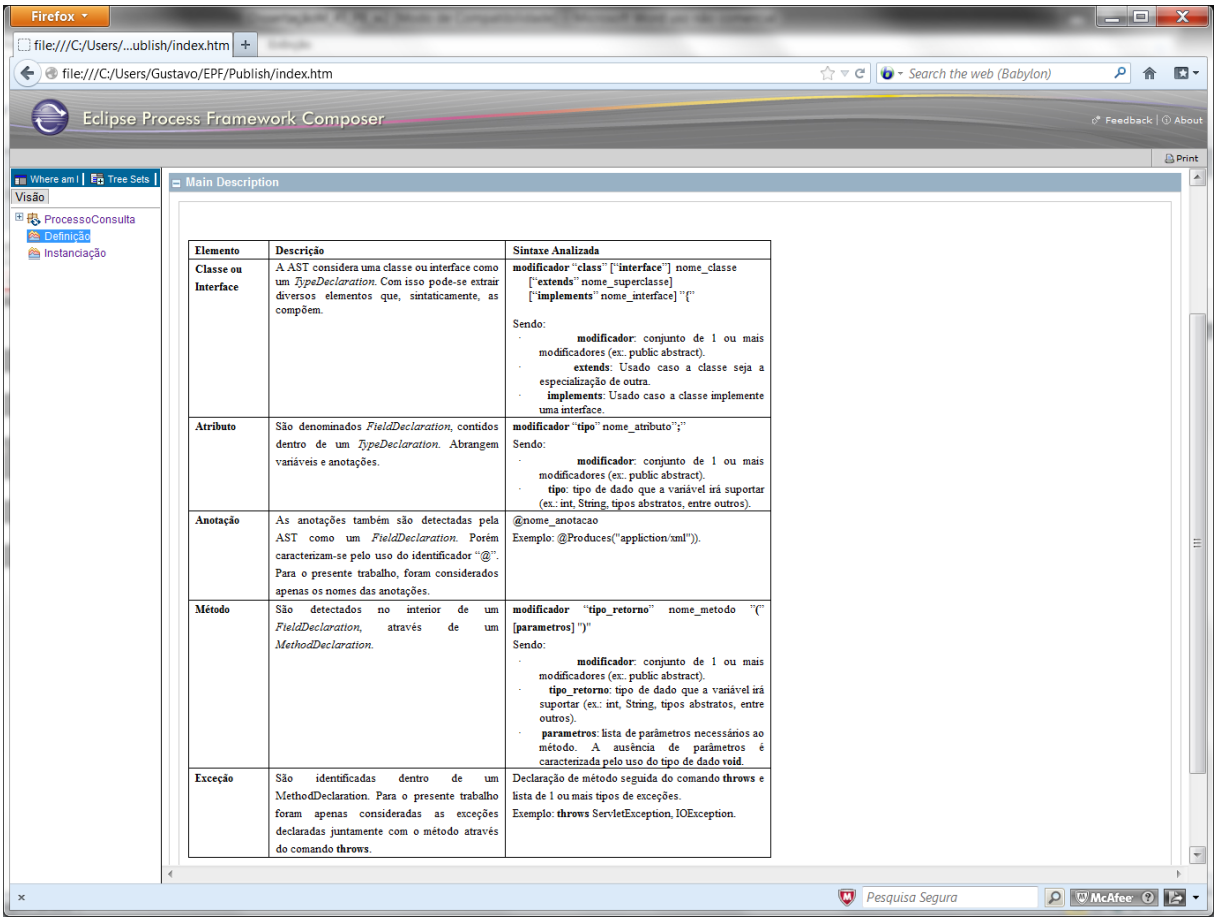


Elemento	Descrição	Síntaxe Analizada
Aspect	Possibilitam a separação de interesses. Dentro deles são declarados os PointCuts e Advices. Também podem, além de estender outros aspectos, estender uma classe ou implementar interfaces.	**modificador** "**aspect**" nome_aspecto ["**extends**" nome_classe_aspecto] ["**implements**" nome_interface] "{" Sendo: · **modificador**: conjunto de 1 ou mais modificadores (ex:. public abstract). · **extends**: Usado caso a classe seja a especialização de outro aspecto ou classe. · **implements**: Usado caso o aspecto implemente uma interface.
Intertype	São declarações que alteram ou adicionam estaticamente algum tipo de funcionalidade ao programa.	**modificador** "**tipo**" nome_intertipo";" Sendo: · **modificador**: conjunto de 1 ou mais modificadores (ex:. public abstract). · **tipo**: tipo de dado que a variável irá suportar (ex.: int, String, tipos abstratos, entre outros).
Advice	Responsáveis pela execução das ações dos PointCuts. Podem ser dos tipos *Before*, *After* ou *Around*.	tipo_advice (monitor): nome_pointcut "{" Sendo: · **tipo_advice**: *after* ou *before*. · **monitor**: variável ou método a ser monitorado. Exemplo: after() returning : testepointcut() { System.out.println("Teste");}
PointCut	São *JoinPoints* capturados pelo AspectJ. Dentre eles podem ser citados: Chamadas de Métodos, *Handlers* e *Advices*.	**pointcut** nome_pointcut() : expressao_chamada; Sendo: · **expressão_chamada**: onde o PointCut irá ser chamado. Ex:. call (void metodoteste()).
Warning	São mensagens de aviso (*warning*) ou erros (*errors*) contidas no interior do aspecto.	**declare error** : nome_pointcut : "mensagem"; **declare warning** : nome_pointcut : "mensagem";

Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer

(continuação)

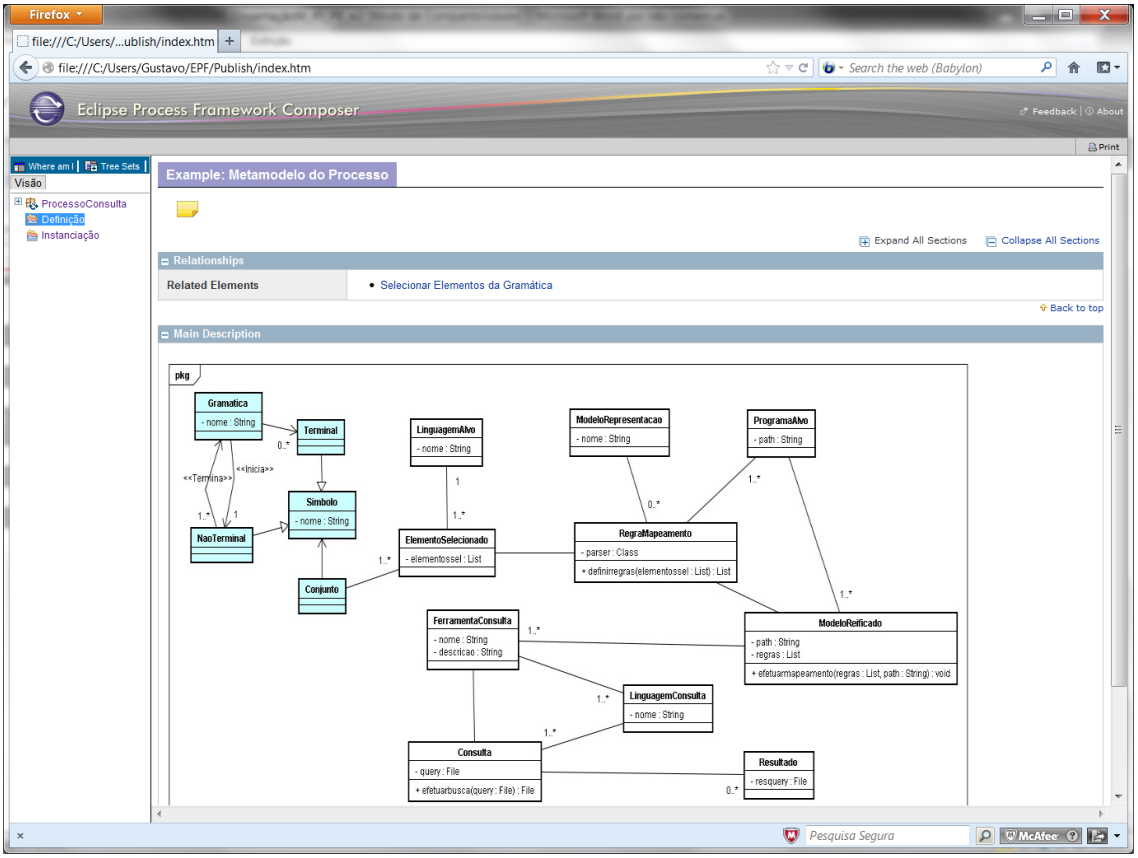
Interface Exemplo (Elementos selecionados para a instanciação OOP):



Elemento	Descrição	Sintaxe Analizada
Classe ou Interface	A AST considera uma classe ou interface como um *TypeDeclaration*. Com isso pode-se extrair diversos elementos que, sintaticamente, as compõem.	**modificador** "**class**" ["**interface**"] nome_classe ["**extends**" nome_superclasse] ["**implements**" nome_interface] "{" Sendo: · **modificador**: conjunto de 1 ou mais modificadores (ex:. public abstract). · **extends**: Usado caso a classe seja a especialização de outra. · **implements**: Usado caso a classe implemente uma interface.
Atributo	São denominados *FieldDeclaration*, contidos dentro de um *TypeDeclaration*. Abrangem variáveis e anotações.	**modificador** "**tipo**" nome_atributo";" Sendo: · **modificador**: conjunto de 1 ou mais modificadores (ex:. public abstract). · **tipo**: tipo de dado que a variável irá suportar (ex.: int, String, tipos abstratos, entre outros).
Anotação	As anotações também são detectadas pela AST como um *FieldDeclaration*. Porém caracterizam-se pelo uso do identificador "@". Para o presente trabalho, foram considerados apenas os nomes das anotações.	@nome_anotacao Exemplo: @Produces("appliction/xml")).
Método	São detectados no interior de um *FieldDeclaration*, através de um *MethodDeclaration*.	**modificador** "**tipo_retorno**" nome_metodo "(" [**parametros**] ")" Sendo: · **modificador**: conjunto de 1 ou mais modificadores (ex:. public abstract). · **tipo_retorno**: tipo de dado que a variável irá suportar (ex.: int, String, tipos abstratos, entre outros). · **parametros**: lista de parâmetros necessários ao método. A ausência de parâmetros é caracterizada pelo uso do tipo de dado **void**.
Exceção	São identificadas dentro de um MethodDeclaration. Para o presente trabalho foram apenas consideradas as exceções declaradas juntamente com o método através do comando **throws**.	Declaração de método seguida do comando **throws** e lista de 1 ou mais tipos de exceções. Exemplo: **throws** ServletException, IOException.

Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer (continuação)

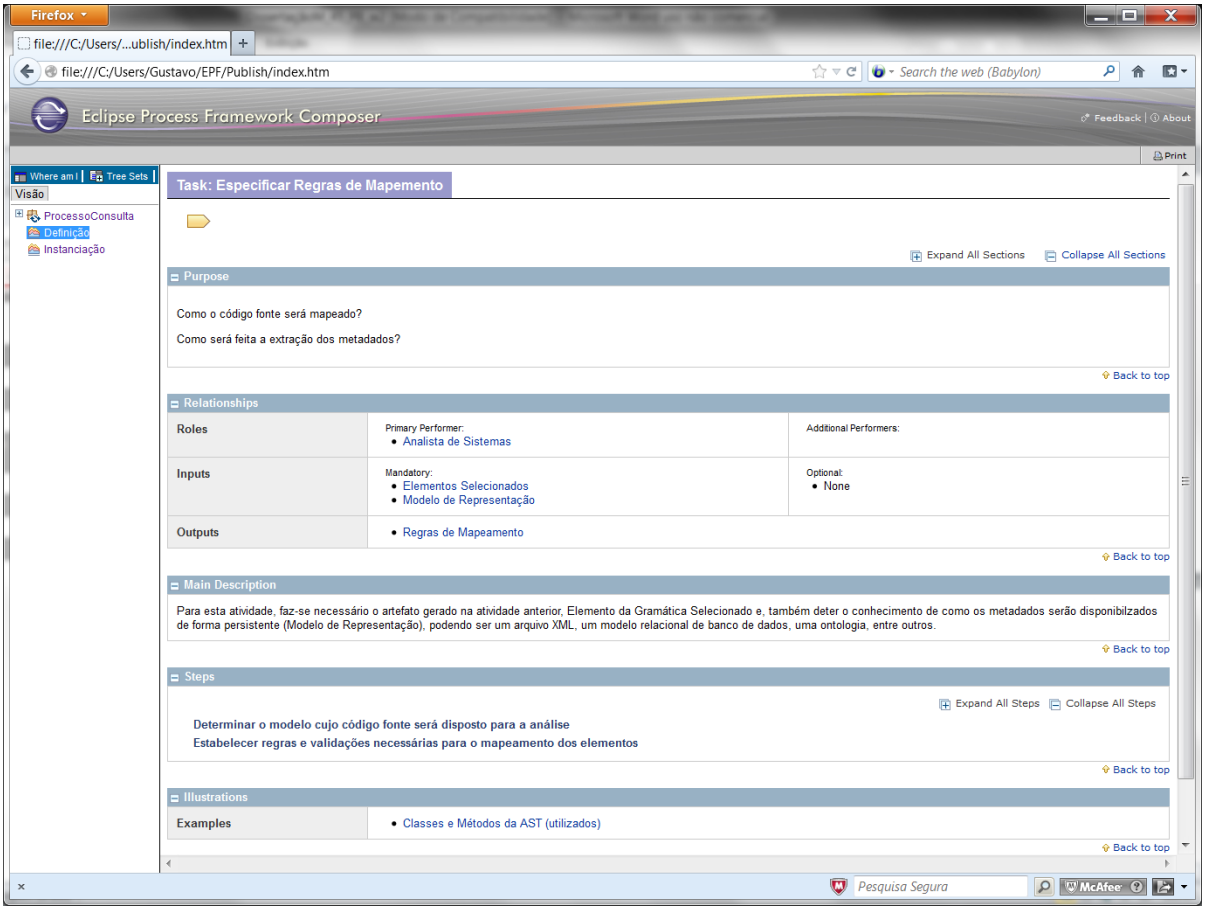
Interface Exemplo (Metamodelo do processo):



Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer

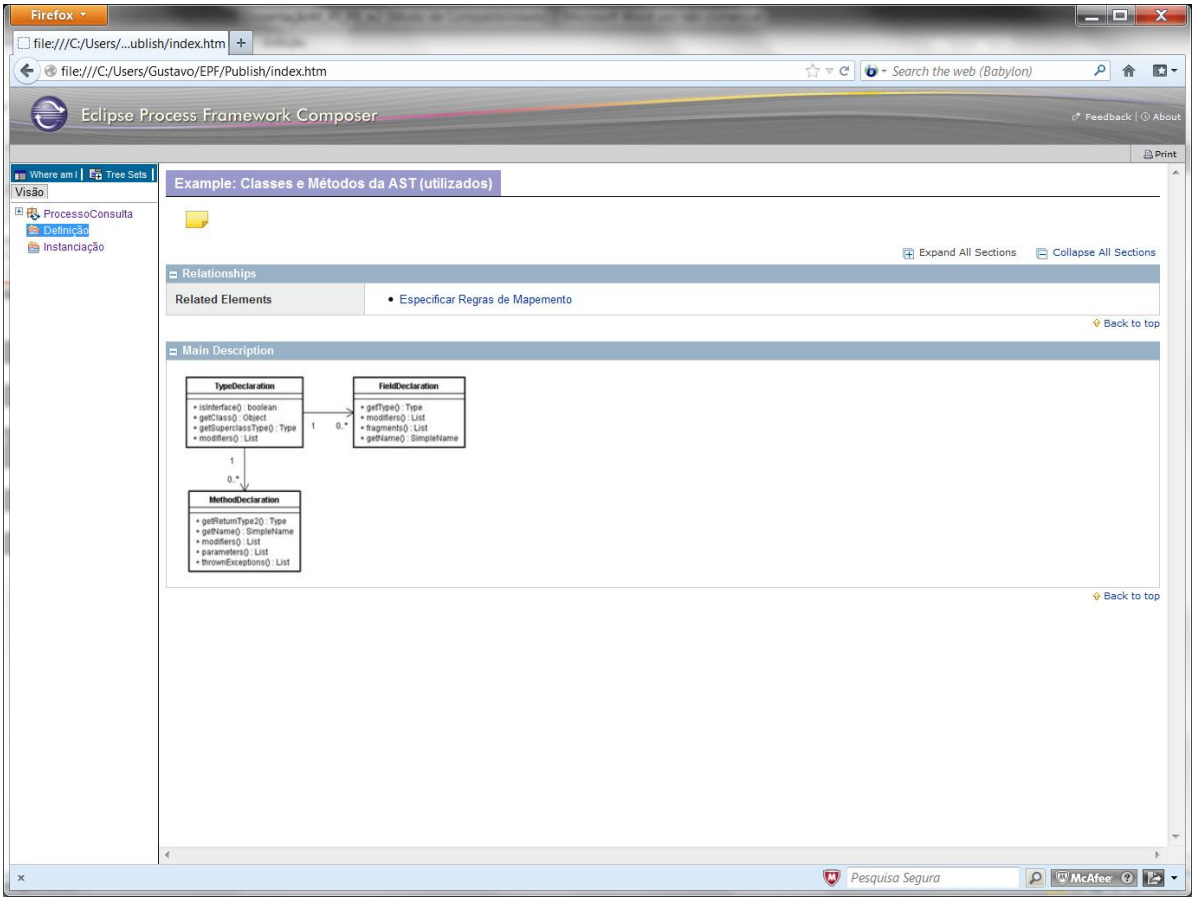
(continuação)

Interface Task (Especificar Regras de Mapeamento):



Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer (continuação)

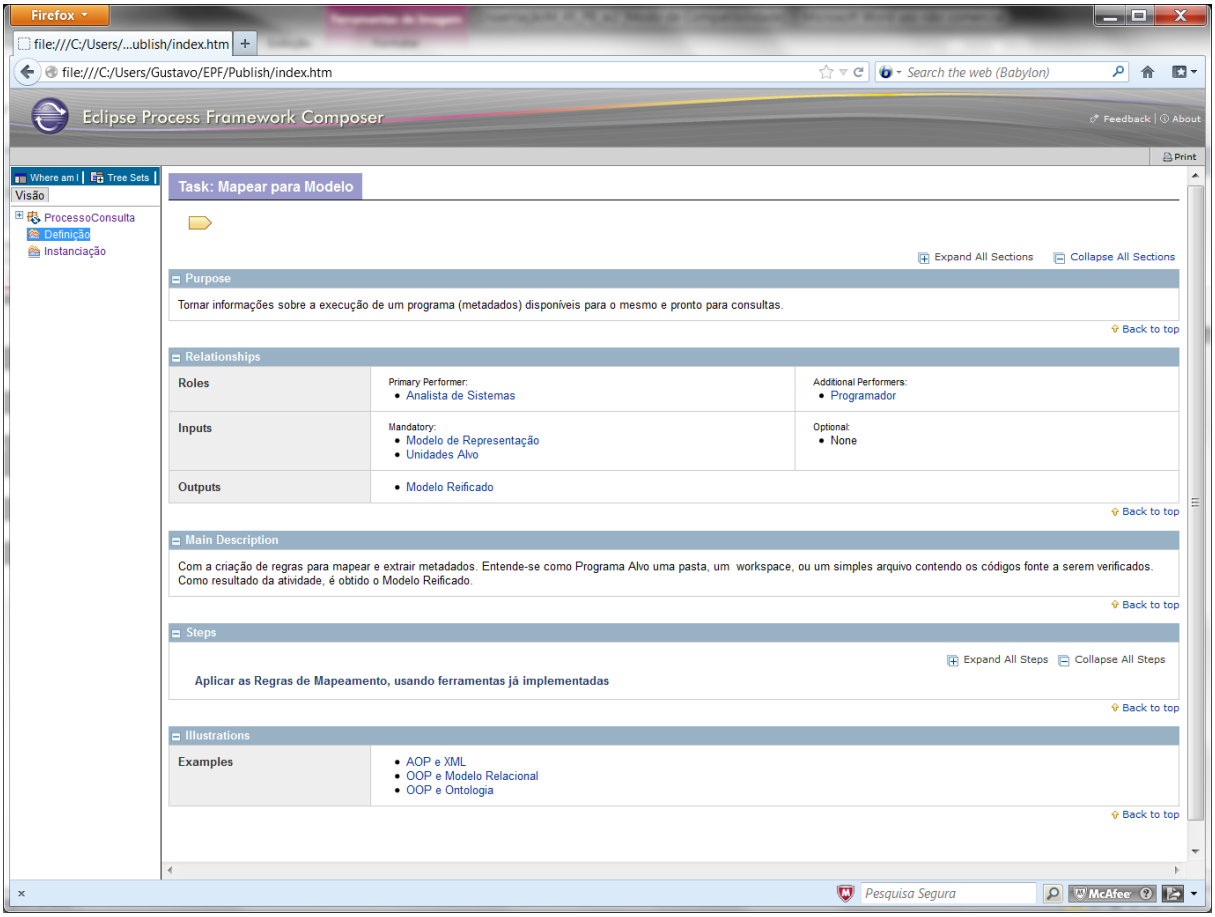
Interface Exemplo (Classes e Métodos da AST utilizados):



Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer

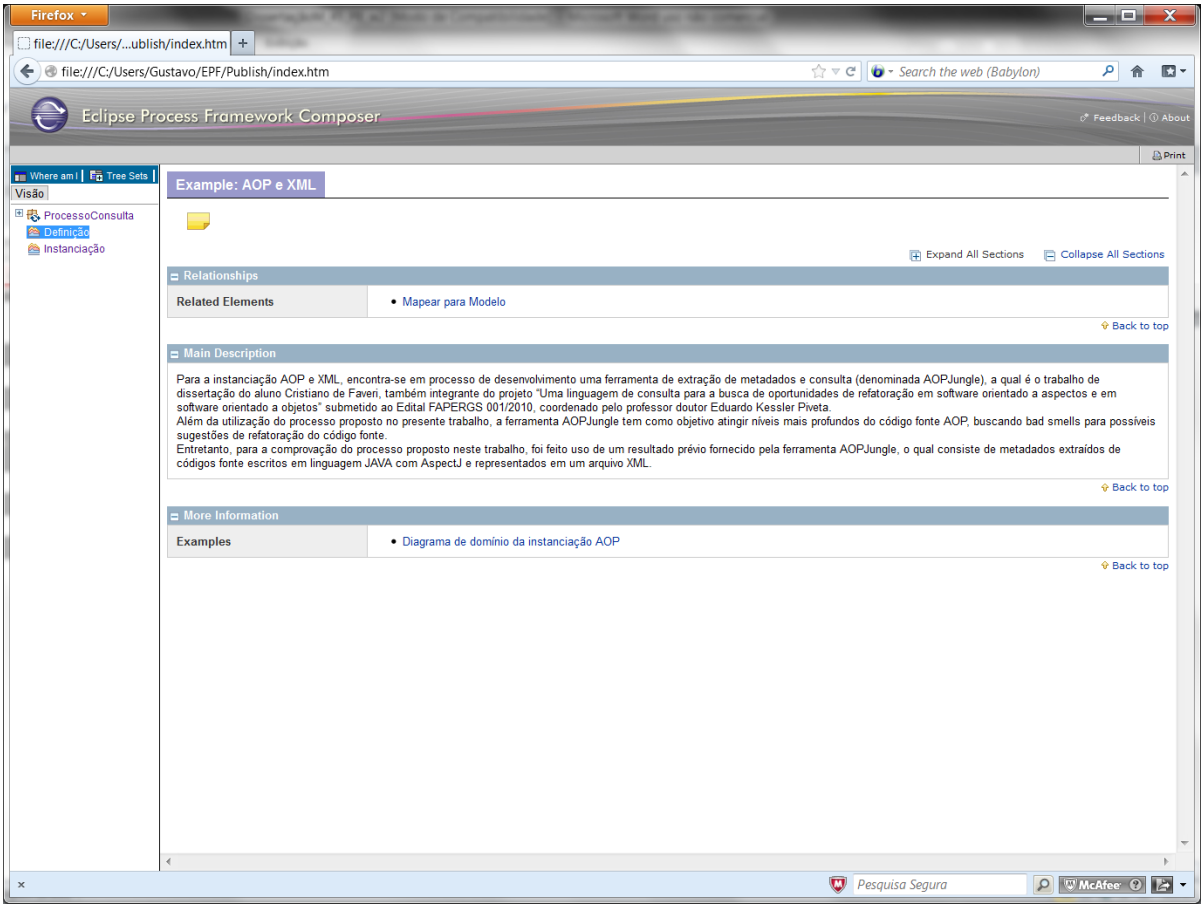
(continuação)

Interface Task (Mapear para Modelo):



Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer (continuação)

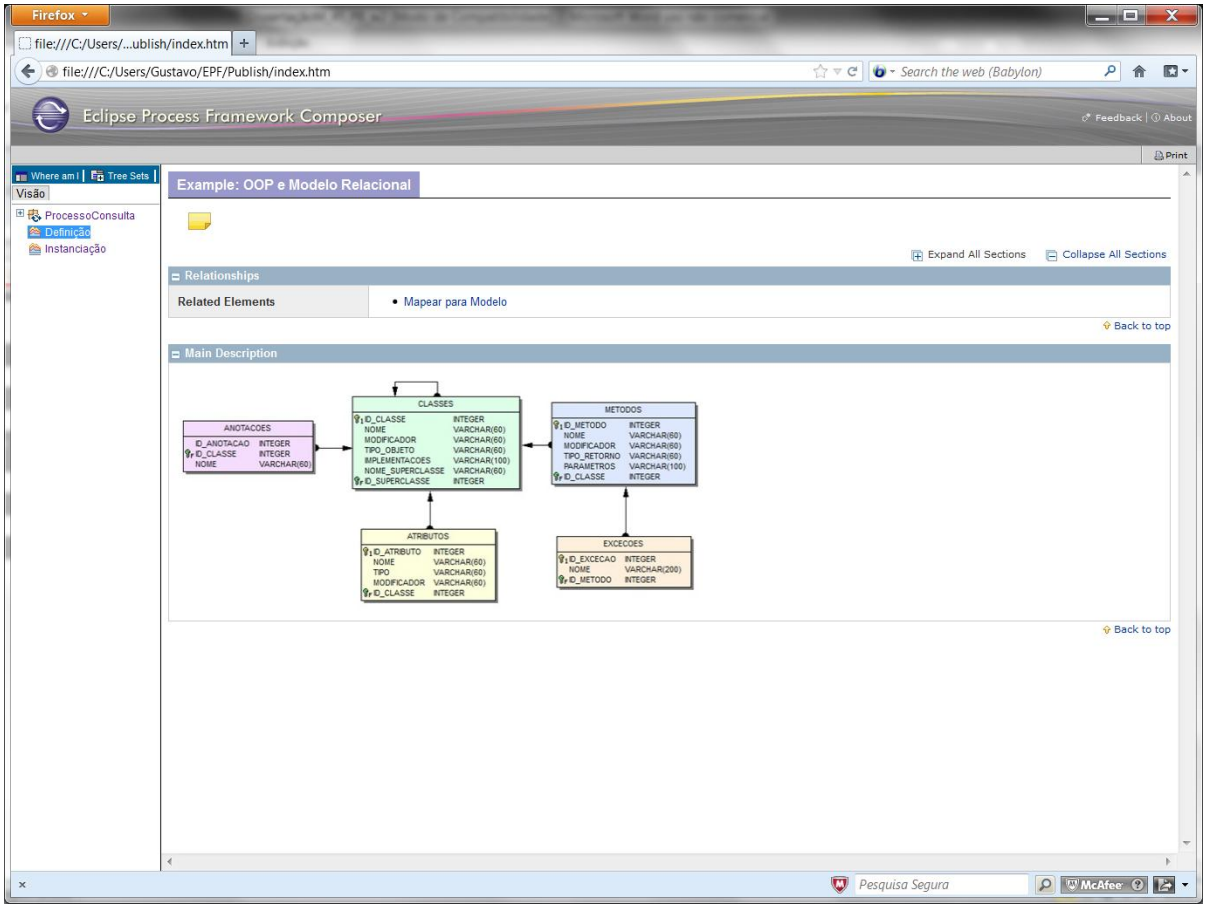
Interface Exemplo (AOP e XML):



Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer

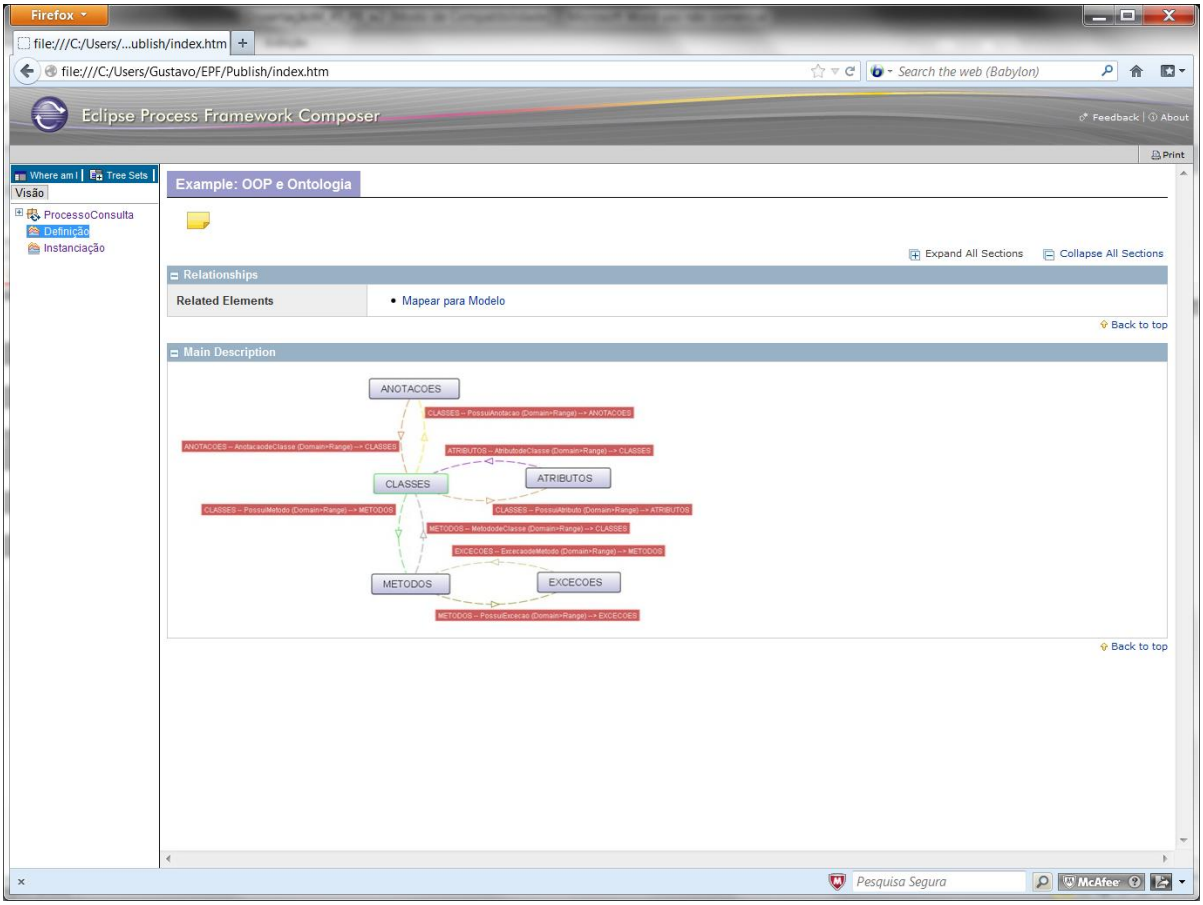
(continuação)

Interface Exemplo (OOP e Modelo Relacional):



Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer (continuação)

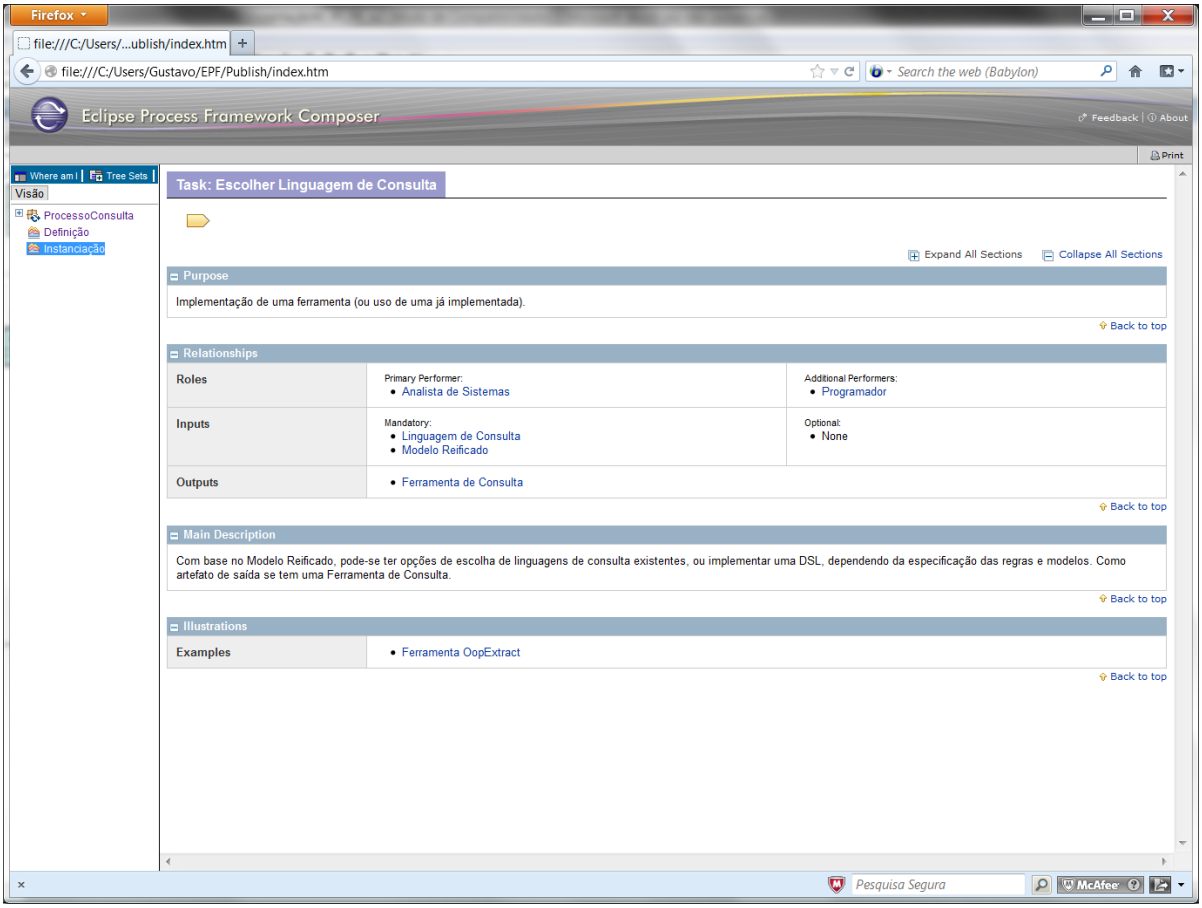
Interface Exemplo (OOP e Ontologia):



Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer

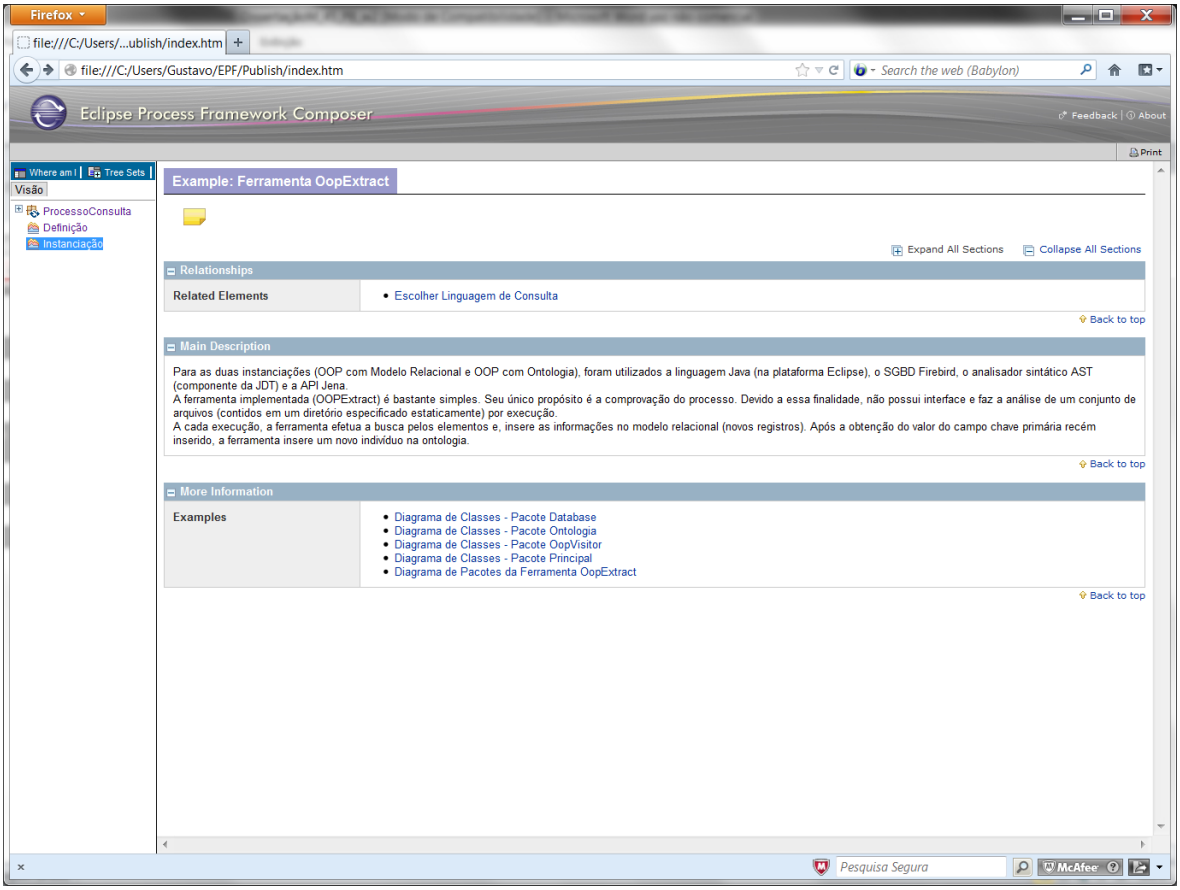
(continuação)

Interface Task (Escolher Linguagem de Consulta):



Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer (continuação)

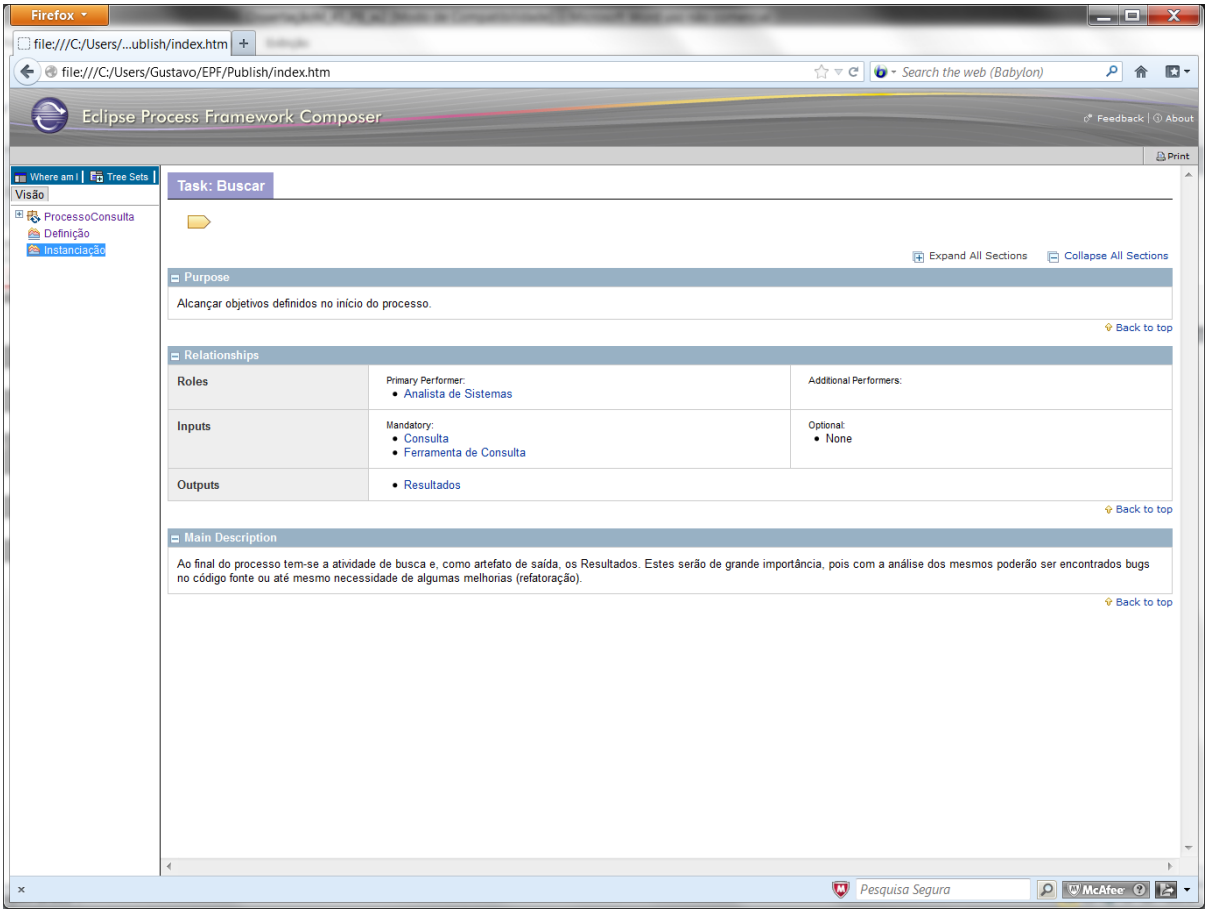
Interface Exemplo (Ferramenta OopExtract):



Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer

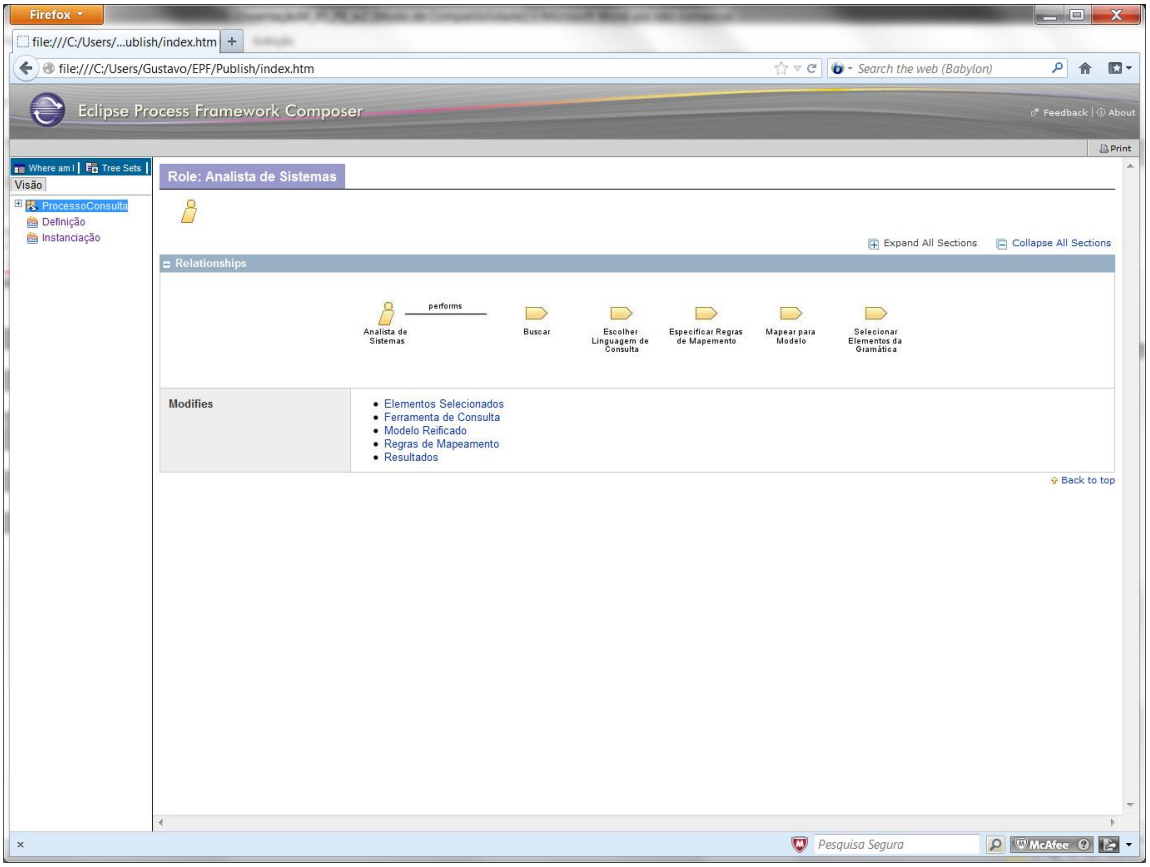
(continuação)

Interface Task (Buscar):



Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer
(continuação)

Interface Responsabilidades do Analista de Sistemas:



Apêndice B – Principais interfaces geradas pela ferramenta EPF Composer

(conclusão)

Interface Responsabilidades do Programador: