UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Fernando Quatrin Campagnolo

UMA EXTENSÃO PARA A LINGUAGEM DE CONSULTA AQL

Santa Maria, RS
2017

Fernando Quatrin Campagnolo

UMA EXTENSÃO PARA A LINGUAGEM DE CONSULTA AQL

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

Orientador: Prof. Eduardo Kessler Piveta

Santa Maria, RS
2017

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

Campagnolo, Fernando Quatrin
Uma extensão para a linguagem de consulta AQL / Fernando Quatrin Campagnolo.- 2017.
159 f.; 30 cm

Orientador: Eduardo Kessler Piveta
Dissertação (mestrado) - Universidade Federal de Santa Maria, Centro de Tecnologia, Programa de Pós-Graduação em Ciência da Computação, RS, 2017

1. AQL 2. Linguagem de Consulta 3. AOP I. Piveta, Eduardo Kessler II. Título.

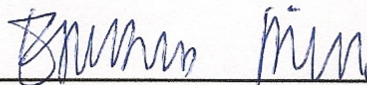
©2017
Todos os direitos autorais reservados a Fernando Quatrin Campagnolo. A reprodução de partes ou do todo deste trabalho só poderá ser feita mediante a citação da fonte.
End. Eletr.: fcampagnolo@inf.ufsm.br

Fernando Quatrin Campagnolo

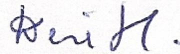
UMA EXTENSÃO PARA A LINGUAGEM DE CONSULTA AQL

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

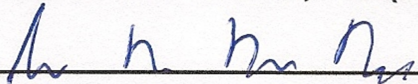
Aprovada em 31 de agosto de 2017



Eduardo Kessler Piveta, Dr. (UFSM)
(Presidente/Orientador)



Deise de Brum Saccol, Dra. (UFSM)



André Rauber Du Bois, Dr. (UFPEL)

Santa Maria, RS
2017

AGRADECIMENTOS

Quero agradecer a todos que, de alguma forma, contribuíram para o desenvolvimento deste trabalho.

Primeiramente, agradeço ao meu orientador, professor Eduardo Kessler Piveta, pela oportunidade e confiança a mim concedidas desde o trabalho de conclusão do curso de Sistemas de Informação até a finalização desta dissertação. Foram inúmeros conselhos, ensinamentos, ideias e sugestões essenciais para minha formação e para o desenvolvimento deste trabalho.

Aos meus amigos e colegas do grupo de pesquisa de Linguagens de Programação e Bancos de Dados da UFSM: Cristiano, Jânio e José Carlos. Obrigado pela ajuda e discussões.

Ao incondicional apoio e incentivo dos meus pais Antônio José Campagnolo e Lisete Terezinha Quatrin Campagnolo. Obrigado por entenderem as noites em claro, as minhas angústias e as minhas preocupações. Sem vocês, nada seria possível.

Aos meus irmãos Ângela Quatrin Campagnolo e Leonardo Quatrin Campagnolo, pela motivação e aquele carinho especial de irmãos.

À minha noiva Mariana Braga da Silva, por sua compreensão, apoio, carinho e amor.

Aos meus amigos, colegas de mestrado e de trabalho que se fizeram presentes nos mais diversos momentos, me apoiando, me orientando e me animando em momentos de dificuldade.

We can't solve problems by using the same kind of thinking we used when we created them.

(Albert Einstein)

RESUMO

UMA EXTENSÃO PARA A LINGUAGEM DE CONSULTA AQL

AUTOR: Fernando Quatrin Campagnolo
ORIENTADOR: Eduardo Kessler Piveta

Os sistemas de software são constantemente modificados e adaptados às novas funcionalidades. Tais modificações comumente aumentam sua complexidade e podem diminuir sua qualidade. Uma das maneiras de auxiliar na gerência desta complexidade e manter um sistema de software atualizado é a aplicação de transformações em programas, mais especificamente, a aplicação de refatorações. A fim de buscar por oportunidades de refatoração nos sistemas, os desenvolvedores podem usar linguagens de consulta em código fonte. Porém, é comum encontrarmos linguagens de consulta descontinuadas, que oferecem poucos recursos e/ou são proprietárias. Esta dissertação tem como objetivo estender a linguagem AQL (*Aspect Query Language*) (FAVERI, 2013), uma linguagem de consulta projetada para realizar buscas em programas orientados a aspectos e programas orientados a objetos. De forma a fornecer novos recursos para a linguagem AQL, um conjunto de melhorias foi especificado e implementado, incluindo: (i) a expansão dos elementos buscados, a fim de ter uma granularidade de busca mais fina; (ii) a criação de instruções para manipular programas na linguagem alvo (inserção, atualização e remoção de elementos); (iii) algumas melhorias quanto à validação e a certas cláusulas da linguagem; e (iv) a extensão do *framework* AOPJungle, que fornece informações dos sistemas analisados para a implementação de referência de AQL. Para avaliar a aplicabilidade dos novos recursos da linguagem, foi realizado um estudo de caso usando a extensão da linguagem AQL como apoio na busca por oportunidades de refatoração e na aplicação de refatoração em programas orientados a objetos.

Palavras-chave: AQL. Linguagem de Consulta. AOP

// ABSTRACT

AN EXTENSION TO THE AQL QUERY LANGUAGE

LISTA DE FIGURAS

Figura 2.1 – Processo proposto por Piveta (2009) 24
Figura 2.2 – Exemplo no Graphviz (FOWLER; PARSONS, 2010) 28
Figura 2.3 – Principais cláusulas da AQL (FAVERI, 2013) 34
Figura 2.4 – Visão geral da implementação de referência de AQL integrada ao Eclipse (FAVERI, 2013) 36
Figura 2.5 – Arquitetura do AOPJungle (FAVERI, 2013) 37
Figura 2.6 – Metamodelo original e simplificado do AOPJungle 38
Figura 2.7 – Arquitetura do Compilador AQL (FAVERI, 2013) 39
Figura 2.8 – Ilustração da transformação da cláusula `where` (FAVERI, 2013) 41
Figura 3.1 – Grafo de sintaxe do elemento `ObjectType` 46
Figura 3.2 – Grafo de sintaxe das cláusulas principais da especificação original de AQL 50
Figura 3.3 – Grafo de sintaxe das cláusulas principais da extensão de AQL 50
Figura 3.4 – Modelo HQL implementado no *Eclipse Modeling Framework* 52
Figura 3.5 – Grafo de sintaxe da cláusula `delete` 53
Figura 3.6 – Grafo de sintaxe da cláusula `insert` 55
Figura 3.7 – Grafo de sintaxe para referenciar um ou mais objetos 57
Figura 3.8 – Grafo de sintaxe da cláusula `update` 60
Figura 3.9 – Grafo de sintaxe da cláusula `order by` 61
Figura 3.10 – Informando um erro na consulta para o usuário 64
Figura 3.11 – Estrutura básica do metamodelo do AOPJungle (TEIXEIRA JÚNIOR, 2014) 65
Figura 3.12 – Novos elementos adicionados no metamodelo do AOPJungle 66
Figura 4.1 – Mostrando informações do elemento `for each` 71
Figura 4.2 – Mostrando informações da chamada ao método `Collections.sort` 75
Figura 4.3 – Mostrando informações de elementos `for each` que possuam condicionais 78
Figura 4.4 – Mostrando informações do método `contar` da classe `Cliente` 82
Figura 4.5 – Mostrando informações do método `contar` da interface `Mostrar` 83
Figura 4.6 – Mostrando informações da classe `Pagamento` 87

LISTA DE TABELAS

Tabela 2.1 – Fases de transformação de código (FAVERI, 2013) 40

LISTA DE ABREVIATURAS E SIGLAS

AJDT	*AspectJ Development Tools*
AO	*Aspect-Oriented*
AQL	*Aspect Query Language*
DML	*Data Manipulation Language*
DSL	*Domain Specific Language*
EMF	*Eclipse Modeling Framework*
GPL	*General Purpose Language*
HQL	*Hibernate Query Language*
IDE	*Integrated Development Environment*
LHS	*Left Hand Side Rule*
NAC	*Negative Application Condition*
OO	*Object-Oriented*
PAC	*Positive Application Condition*
RHS	*Right Hand Side Rule*

SUMÁRIO

1 INTRODUÇÃO ... **19**
2 REVISÃO DE LITERATURA ... **23**
2.1 REFATORAÇÃO ... 23
2.2 LINGUAGENS ESPECÍFICAS DE DOMÍNIO ... 25
2.3 LINGUAGENS DE CONSULTA EM CÓDIGO ... 29
2.3.1 JQuery ... 30
2.3.2 Lost ... 30
2.3.3 JTL ... 31
2.3.4 .QL ... 31
2.3.5 CppDepend ... 31
2.3.6 Jackpot ... 32
2.3.7 PMD ... 32
2.3.8 ActiveAspect ... 33
2.3.9 Mylar ... 33
2.4 AQL ... 33
2.4.1 Sintaxe da AQL ... 34
2.5 ARQUITETURA DA IMPLEMENTAÇÃO DE REFERÊNCIA DE AQL ... 35
2.5.1 AOPJungle ... 36
2.5.2 Compilador de AQL ... 38
2.6 CONSIDERAÇÕES FINAIS ... 42
3 EXTENSÕES PARA AQL ... **43**
3.1 MODIFICANDO A GRANULARIDADE DAS BUSCAS ... 45
3.1.1 Adicionando novos elementos de busca ... 47
3.2 NOVAS CLÁUSULAS ... 50
3.2.1 Adicionando cláusulas para manipulação de dados ... 51
3.2.2 A cláusula `delete` ... 52
3.2.2.1 Implementando a cláusula `delete` ... 53
3.2.3 A cláusula `insert` ... 55
3.2.3.1 Implementando a cláusula `insert` ... 57
3.2.4 A cláusula `update` ... 59
3.2.4.1 Implementando a cláusula `update` ... 60
3.3 APRIMORANDO A CLÁUSULA `ORDER BY` ... 61
3.3.1 Implementado melhorias na Cláusula `Order By` ... 62
3.4 MELHORANDO VALIDAÇÕES ... 62
3.4.1 Implementando Validações ... 63
3.5 MODIFICAÇÕES REALIZADAS NO *FRAMEWORK* AOPJUNGLE ... 64
3.6 CONSIDERAÇÕES FINAIS ... 67
4 ESTUDO DE CASO ... **69**
4.1 APLICANDO *CONVERT ENHANCED FOR TO LAMBDA ENHANCED FOR* 69
4.2 APLICANDO *CONVERT COLLECTIONS.SORT TO SORT* ... 74
4.3 APLICANDO *CONVERT ENHANCED FOR WITH IF TO LAMBDA FILTER* . 77
4.4 APLICANDO *CONVERT ABSTRACT INTERFACE METHOD TO DEFAULT METHOD* ... 81
4.5 APLICANDO *CHAIN CONSTRUCTORS* ... 85
4.6 CONSIDERAÇÕES FINAIS ... 90

5 CONCLUSÃO 91
5.1 TRABALHOS FUTUROS 92
REFERÊNCIAS BIBLIOGRÁFICAS 93
ANEXO A – SINTAXE ORIGINAL DE AQL EM XTEXT 99
ANEXO B – SINTAXE ATUAL DE AQL EM XTEXT 101
ANEXO C – CLÁUSULAS GERADAS PELO COMPILADOR AQL 103
ANEXO D – CLASSES MODIFICADAS DO COMPILADOR AQL 107
ANEXO E – CLASSES MODIFICADAS DO AOPJUNGLE 125

1 INTRODUÇÃO

Os sistemas de software estão em constante evolução. Constantemente, pode ser necessário que eles sejam melhorados, modificados e adaptados a novos requisitos. Para realizar tais modificações, os desenvolvedores geralmente precisam compreender, implementar e avaliar as modificações realizadas no sistema. Dependendo da quantidade de linhas de código, da complexidade do sistema e da quantidade de módulos existentes, os desenvolvedores passam mais tempo estudando o código fonte do que efetuando tais modificações (PRESSMAN, 2011; SOMMERVILLE, 2011). Essas modificações tendem a aumentar a complexidade dos sistemas e a diminuir sua qualidade. Quanto mais complexo for um sistema, mais custosa será sua manutenção e a implementação de novos requisitos (MENS; TOURWE, 2004).

Uma das maneiras de auxiliar na gerência desta complexidade e manter um sistema atualizado é a aplicação de transformações em programas, mais especificamente, a aplicação de refatorações. Refatoração é o processo de melhoria de software que modifica a estrutura de um sistema sem modificar seu comportamento externamente observável e também o nome dado às transformações individuais aplicadas aos artefatos de software (OPDYKE, 1992; FOWLER et al., 1999).

Existem alguns processos que descrevem as atividades necessárias para selecionar, avaliar e aplicar refatorações em um sistema de software. Um destes processos é proposto por Piveta (2009), o qual divide as atividades em dois grupos: (i) atividades de preparação e (ii) atividades de busca. Inicialmente, nas atividades de preparação, os desenvolvedores selecionam as refatorações, os modelos de qualidade e as funções heurísticas a serem utilizadas. Posteriormente, nas atividades de busca, os desenvolvedores utilizam os resultados das atividades anteriores para realizar buscas por oportunidades de refatoração em seus sistemas.

Na atividade de buscas por oportunidades de refatoração, os desenvolvedores podem utilizar linguagens de consulta em código fonte e mecanismos de buscas, tais como .QL (SEEMLE, 2017) e JQuery (MCCORMICK; VOLDER, 2004) para sistemas orientados a objetos e ActiveAspect (COELHO; MURPHY, 2005) para sistemas orientados a aspectos. Mylar (KERSTEN; MURPHY, 2005), Sourcerer (BAJRACHARYA; LOPES, 2014) e Mycroft (URMA; MYCROFT, 2015) são mecanismos de busca e análise de grandes volumes de código e Open Hub Code Search (HUB, 2017), Semmle (SEEMLE, 2017) e Krugle (GROUP, 2017) são motores de busca não estruturada para código fonte.

Entretanto, essas linguagens e mecanismos possuem ao menos uma das limitações a seguir: (i) apenas a possibilidade de realizar consultas com granularidade grossa nos códigos fontes, (ii) suas implementações foram descontinuadas, não existem versões correntes ou não possuem versão de código aberto, (iii) focam mais na apresentação visual

que nos recursos linguísticos de busca, (iv) permitem apenas busca não estruturada, (v) são específicos de linguagem ou (vi) focam mais em aspectos de plataforma/escalabilidade do que de linguagem propriamente ditos.

Recentemente, no Laboratório de Linguagens de Programação e Bancos de Dados da UFSM, foi projetada, especificada e implementada uma DSL (*Domain-Specific Language*) (FOWLER; PARSONS, 2010) para realizar consulta em código fonte orientado a aspectos e orientado a objetos, denominada AQL (*Aspect Query Language*) (FAVERI, 2013). AQL é uma linguagem declarativa que utiliza conceitos de OQL (*Object Query Language*) e um metamodelo para realizar buscas em programas. Entretanto, essa linguagem possui algumas limitações que podem ser exploradas, incluindo a falta de uma DML (*Data Manipulation Language*), a granularidade grossa das consultas e certa margem para aperfeiçoamento de algumas cláusulas. Além disso, na implementação de referência da linguagem AQL, existem validações que podem ser melhoradas.

Com base nas limitações citadas, essa dissertação tem como principal objetivo especificar e implementar um conjunto de melhorias para a linguagem AQL (FAVERI, 2013), através da extensão da linguagem AQL, incluindo: (i) novos tipos de objetos que permitem realizar buscas por unidades de código contidas dentro de métodos e construtores, (ii) novas cláusulas que permitem a inserção, atualização e remoção de elementos dos sistemas, (iii) aprimoramento da cláusula *order by* e (iv) novas validações.

De forma a avaliar a extensão proposta, foram implementadas duas extensões de ferramentas existentes e foi conduzido um estudo de caso:

- **Extensão do compilador AQL**: foram modificados e adicionados elementos na gramática do compilador AQL para dar suporte as cláusulas de manipulação de dados e aos novos tipo de objetos. Foram adicionados elementos no metamodelo intermediário que representa o HQL *Hibernate Query Language*. Foram adicionadas regras de transformação e emissão para as novas cláusulas, para a cláusula aprimorada e para os novos tipos de objetos. E foram adicionadas regras de validação para os elementos modificados e adicionados na gramática.

- **Extensão do *framework* AOPJungle**: o AOPJungle fornece métricas e informações de programas orientados a objetos (OO) e orientados a aspectos (OA) que podem ser acessadas através de consultas. A implementação de referência da linguagem AQL utiliza o metamodelo do AOPJungle como referência para emissão de suas consultas. Dessa forma, foram adicionadas informações de unidades de código contidas dentro de métodos e construtores em seu modelo de dados.

- **Estudo de caso:** para mostrar o uso da extensão da linguagem AQL como apoio nas atividades de busca por oportunidades de refatoração e a aplicação de refatoração em código orientado a objetos foi realizado um estudo de caso utilizando a extensão do compilador AQL e a extensão do *framework* AOPJungle.

Esta dissertação está organizada como segue. O Capítulo 2 apresenta um embasamento teórico acerca dos seguintes temas: refatoração, linguagens específicas de domínio, linguagens de consulta em código, especificações da linguagem AQL e arquitetura de AQL. O Capítulo 3 apresenta os novos elementos adicionados na linguagem, a especificação sintática das novas cláusulas da AQL, exemplos de seu uso e as melhorias propostas para a linguagem. O Capítulo 4 apresenta os detalhes da implementação da extensão da linguagem AQL e da extensão do *framework* AOPJungle. O Capítulo 5 apresenta um estudo de caso utilizando algumas refatorações e algumas das novas funcionalidades implementadas na linguagem AQL. O Capítulo 6 resume as contribuições desta dissertação e aponta algumas sugestões para trabalhos futuros.

2 REVISÃO DE LITERATURA

Este capítulo apresenta uma revisão de literatura referente aos assuntos necessários para o entendimento deste trabalho: refatoração (Seção 2.1), linguagens específicas de domínio (Seção 2.2), linguagens de consulta em código fonte (Seção 2.3) e a linguagem AQL (*Aspect Query Language*) (Seção 2.5).

2.1 REFATORAÇÃO

Os sistemas de software passam por diversas mudanças durante seu ciclo de vida, as quais podem modificar sua complexidade e qualidade. Uma das maneiras de reduzir a complexidade dos sistemas, melhorar incrementalmente seus atributos de qualidade e mantê-los atualizados é utilizar refatorações (ISO, 2001; KATAOKA et al., 2001; BOEHM; IN, 2009).

Uma refatoração é o processo de melhoria de software que modifica a estrutura de um sistema sem modificar seu comportamento externamente observável (OPDYKE, 1992). Por exemplo, a refatoração *Renomear Método* é utilizada para alterar nomes de métodos que não são entendíveis e todas as suas ocorrências no sistema, melhorando o entendimento do código sem mudar seu comportamento (FOWLER et al., 1999).

Piveta (2009) propôs um processo de refatoração que possui algumas atividades. Elas são divididas em dois grupos: (i) atividades de preparação: são atividades relacionadas ao gerenciamento dos modelos de qualidade, à seleção de refatorações e das regras heurísticas; (ii) atividades de busca: são atividades relacionadas à busca de oportunidade de refatoração, à avaliação dos seus efeitos, à priorização e à aplicação de refatorações.

A Figura 2.1 mostra as atividades deste processo através de um diagrama de atividades. Observam-se na parte superior da Figura 2.1 as atividades de preparação. São elas:

- **Selecionar modelos de qualidade:** este é o primeiro passo proposto no processo, que consiste em selecionar ou criar modelos de qualidade a partir dos quais os sistemas de software serão avaliados. Para isso, é necessário selecionar os atributos de qualidade e as métricas a serem utilizadas. São exemplos de atributos de qualidade: reusabilidade, legibilidade e compreensibilidade. São exemplos de métricas: linhas de código, números de operações em um módulo e profundidade de hierarquia de herança.
- **Selecionar refatorações:** os desenvolvedores devem selecionar um conjunto de refatorações contidas nos catálogos de refatorações de acordo com os atributos de

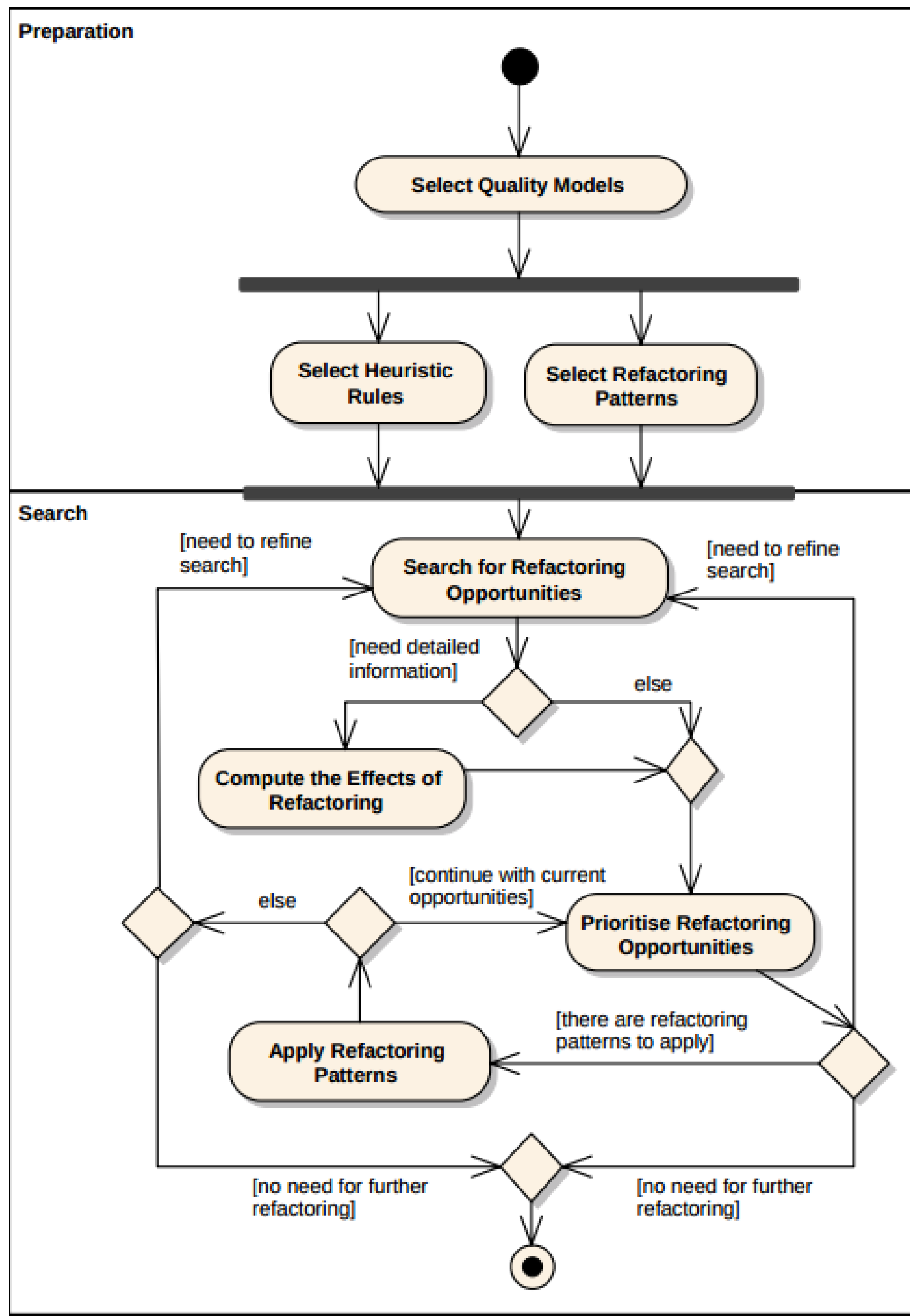


Figura 2.1 – Processo proposto por Piveta (2009)

qualidade.

- **Selecionar regras heurísticas:** este próximo passo consiste em selecionar ou criar funções heurísticas relacionadas aos atributos de qualidade e às métricas selecionadas na atividade selecionar modelos de qualidade.

Observa-se na parte inferior da Figura 2.1 as atividades de busca. São elas:

- **Buscar por oportunidades de refatoração:** consiste em identificar os locais, nos artefatos do sistema, onde as refatorações podem ser aplicadas. Uma oportunidade de refatoração é definida como uma associação entre um artefato de software, uma limitação e uma refatoração.
- **Avaliar os efeitos da refatoração:** geralmente é realizada por avaliações quantitativas, aplicando funções de impacto. Funções de impacto são funções matemáticas que estimam o valor de uma métrica ao se aplicar uma determinada refatoração antes de executá-la (BOIS; MENS, 2003) (BOIS, 2006). Por exemplo, o método A terá n linhas de código após aplicar a refatoração B, porém, a refatoração B ainda não foi aplicara ao método A.
- **Priorizar as oportunidades de refatoração:** consiste em, depois de avaliar os efeitos das refatorações associadas às oportunidades encontradas, ordenar e filtrar quais oportunidades de refatoração devem ser consideradas em um determinado sistema.
- **Aplicar refatorações:** nesta atividade, as refatorações devem ser aplicadas, verificando se os comportamentos externamente observáveis foram preservados.

2.2 LINGUAGENS ESPECÍFICAS DE DOMÍNIO

Uma Linguagens Específicas de Domínio (DSL) é uma linguagem de programação ou uma especificação executável de uma linguagem de computadores de expressividade limitada que tem como objetivo resolver, de forma otimizada, problemas de um domínio em particular (FOWLER; PARSONS, 2010). Ou ainda, uma DSL é uma fina camada sobre um modelo, que pode ser uma biblioteca ou um *framework* (DEURSEN; KLINT; VISSER, 2000). Comumente, elas são utilizadas para atender a um propósito específico de forma mais eficiente quando comparadas às linguagens de propósito geral. Fowler e Parsons (2010) definem quatro elementos-chave de uma DSL:

- Facilitar os desenvolvedores a instruir o computador a fazer algo;

- Possuir fluência;
- Possuir expressividade limitada;
- Ter um foco claro em um domínio pequeno.

As DSLs podem ser divididas em três categorias:

1. DSL externa: é uma linguagem caracterizada por possuir regras sintáticas e semânticas diferentes da linguagem hospedeira (FOWLER; PARSONS, 2010). Normalmente ela possui um conjunto de recursos limitado ao domínio para o qual ela foi projetada. Por exemplo, SQL (*Structured Query Language*) (SQL, 2017), expressões regulares (REGEXR, 2017), AWK (BENTLEY, 1989), XML (*eXtensible Markup Language*) (XML, 2017) e AQL (*Aspect Query Language*) (FAVERI, 2013).

2. DSL interna: é uma linguagem que utiliza as mesmas regras sintáticas e semânticas da linguagem hospedeira. Porém, ela utiliza apenas um subconjunto dos recursos da linguagem para tratar um pequeno aspecto do sistema como um todo. Por exemplo, Lisp quando usada para a criação e o uso de DSLs (LISP, 2017) e algumas bibliotecas de Ruby, tais como as bibliotecas utilizadas pelo *framework* Rails (RAILS, 2017).

3. Language Workbench: São IDEs (*Integrated Development Environments*) especializadas em definir e construir DSLs, tais como: Xtext[1] (EYSHOLDT; BEHRENS, 2010) e Meta-Programming System (MPS) (JETBRAINS, 2017).

Uma DSL ainda pode ser classificada quanto à sua aparência. DSLs textuais expressam o domínio utilizando caracteres, que são combinados em palavras, expressões e instruções seguindo regras gramaticais bem definidas. As DSLs não textuais permitem expressar o domínio por meio de modelos visuais, com o auxílio de símbolos gráficos além de elementos textuais, tais como, tabelas, figuras e fórmulas (FAVERI, 2013).

Um ponto importante a ser considerado quando se projeta ou se usa uma DSL são os riscos e as oportunidades que devem ser levados em consideração antes de sua adoção e desenvolvimento (MERNIK; HEERING; SLOANE, 2005; FOWLER; PARSONS, 2010). Algumas vantagens observadas quanto ao projeto e ao uso de DSLs:

- **Aprimoram a produtividade de desenvolvimento:** Por possuírem uma expressividade limitada, as DSLs podem aumentar a legibilidade do código. Quanto mais fácil a leitura de programas escritos em uma linguagem, mais fácil é compreender, modificar e encontrar erros nos sistemas de software escritos usando tal linguagem (FOWLER; PARSONS, 2010).

[1] https://eclipse.org/Xtext/

- **Facilitam a comunicação com especialistas em domínio:** Uma DSL possui um domínio limitado. Dessa forma, seu modelo pode ser projetado de forma que o especialista do domínio consiga compreender a lógica de uma determinada funcionalidade implementada por essa DSL. Isso facilita a comunicação e a correção de eventuais erros (FOWLER; PARSONS, 2010; DEURSEN; KLINT; VISSER, 2000).

- **Permitem mudança no contexto da execução:** Através de DSLs, pode-se executar código em um ambiente diferente do qual a linguagem hospedeira utiliza. Dessa forma, pode-se suprimir algumas limitações que a linguagem hospedeira possui (FOWLER; PARSONS, 2010).

- **Fornecem um modelo computacional alternativo:** Através de DSLs, pode-se utilizar a melhor abordagem para as diferentes situações, fugindo dos modelos tradicionais (FOWLER; PARSONS, 2010).

Entretanto, é importante salientar algumas desvantagens quando DSLs são usadas (FOWLER; PARSONS, 2010; DEURSEN; KLINT; VISSER, 2000; MERNIK; HEERING; SLOANE, 2005):

- O uso de muitas DSLs pode dificultar o entendimento e a inclusão de novos desenvolvedores. Eles precisarão entender as novas linguagens para conseguir compreender os sistemas.

- Projetar, implementar e manter uma DSL pode ser uma tarefa custosa quando uma biblioteca pode suprir as necessidades que originaram a DSL.

- Empresas que desenvolvem e utilizam suas próprias DSLs podem ter dificuldades em encontrar novos desenvolvedores quando tais DSLs são utilizadas apenas nos seus sistemas.

- Os desenvolvedores podem forçar a resolução de um problema utilizando uma DSL que não foi projetada para este fim. Aumentando a complexidade do código e dificultando sua manutenção.

Existem diversas DSLs criadas para os mais diversos domínios. Por exemplo, a linguagem DOT (DOT, 2017) é uma DSL textual para a descrição de grafos. Para renderizar esses grafos, são utilizados programas que processam, manipulam e geram a visualização a partir *scripts* em DOT, como por exemplo, Graphviz (GRAPHVIZ, 2017), Canviz (CANVIZ, 2017) e o Pydot (PYDOT, 2017).

A Listagem 2.1 mostra um exemplo de código DOT e a Figura 2.2 mostra o resultado gerado visualmente utilizando o Graphviz. Na Listagem 2.1 observamos que os nós, ou vértices, são declarados através da palavra-chave `node`. Já as arestas, ou arcos, são declarados usando o operador `'->'`. Tanto os nós quanto os arcos podem receber atributos listados entre colchetes (FOWLER; PARSONS, 2010).

```
digraph finite_state_machine {
    rankdir=LR;
    size="8,5"
    node [shape = doublecircle]; LR_0 LR_3 LR_4 LR_8;
    node [shape = circle];
    LR_0 -> LR_2 [ label = "SS(B)" ];
    LR_0 -> LR_1 [ label = "SS(S)" ];
    LR_1 -> LR_3 [ label = "S($end)" ];
    (...)
}
```

Listagem 2.1: Exemplo de gráfico utilizando Graphviz (FOWLER; PARSONS, 2010)

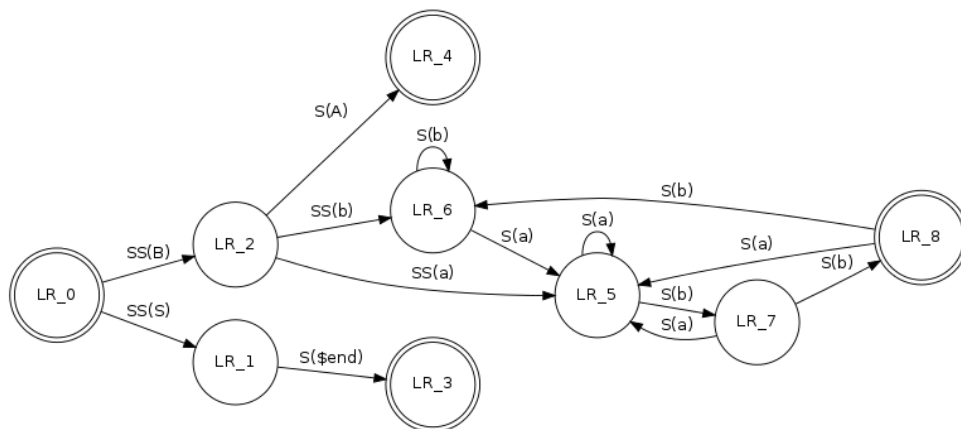


Figura 2.2 – Exemplo no Graphviz (FOWLER; PARSONS, 2010)

JMock (JMOCK, 2017) é uma DSL em Java que oferece suporte ao desenvolvimento guiado por testes (TDD - *Teste-Driven Development*) utilizando objetos simulados (*Mock Objects*). Esses objetos facilitam o projeto e os testes das interações entre os objetos em um sistema de software. Algumas características desta DSL são: (i) facilidade de definir objetos simulados e especificar interações entre eles e (ii) utilização de interfaces progressivas, as quais permitem usar funcionalidades disponibilizadas pelas IDEs, por exemplo o autocompletar (JMOCK, 2017).

A Listagem 2.2 mostra um exemplo de código em JMock. A linha 1, declara um objeto simulado `mainframe` que espera que o método `buy` (comprar) seja chamado uma vez. Na linha 2, o parâmetro da chamada deve ser igual à constante `QUANTITY` (quantidade). Quando chamado, este trecho de código retornará o valor da constante `TICKET` (bilhete).

```
mainframe.expects(once()).method("buy")
 .with(eq(QUANTITY)).will(returnValue(TICKET));
```

Listagem 2.2: Exemplo de teste utilizando JMock

CSS (*Cascading Style Sheets*) é uma linguagem declarativa comumente utilizada para formatar o estilo de documentos web, incluindo elementos de texto, imagens, vídeos

ou áudio (CSS, 2017). Os estilos são associados aos elementos de uma interface através de expressões que possuem regras de formatação. Estas regras definem o estilo dos elementos, por exemplo, a cor de fundo, a cor do texto, o estilo das bordas, dentre outros. Observa-se na Listagem 2.3 um exemplo de código CSS que estiliza as *tags* `h1`, `h2` e `p` de um documento *HTML* (*HyperText Markup Language*).

```
1 h1, h2 {
2    color: #fef;
3    font-family: sans-serif;
4 }
5 p { margin-botton: 10px; }
```

Listagem 2.3: Exemplo de folha de estilo CSS

Existem ainda DSLs com base em DSLs, como por exemplo, a linguagem SASS (*Syntactically Awesome StyleSheets*) (SASS, 2017) e LESS (LESS, 2017). Elas baseiam-se em CSS e oferecem recursos que facilitam a escrita e estendem as funcionalidades de CSS, fornecendo recursos tais como *mixins*, operações aritméticas e variáveis.

Outros domínios relevantes para DSLs têm sido explorados e estudados, tais como: (i) compiladores - YACC (*Yet Another Compiler-Compiler*) (JOHNSON, 2017) e Bison (BISON, 2017), (ii) pesquisa e manipulação de dados, tais como SQL (*Structured Query Language*) (DAMAS, 2007), HQL (HIBERNATE, 2017) e XQuery (WALMSLEY, 2007), (iii) formatação de texto, como LaTeX (LATEX, 2017) (iv) descrição de páginas web com HTML (W3C, 2017), (v) criação de interfaces gráficas com XAML (XAML, 2017), (vi) criação de cenários de testes com FIT (FIT, 2017) e (vii) estruturação de redes de dependências com Make (MAKE, 2017).

2.3 LINGUAGENS DE CONSULTA EM CÓDIGO

As linguagens de consulta em código são linguagens de computador utilizadas para realizar buscas em programas representados através de uma estrutura. Comumente são utilizadas nas atividades de reengenharia de software, incluindo buscas por informações contidas nos programas, na compreensão de sistemas e nas buscas por oportunidades de refatoração. Normalmente, essas linguagens fornecem recursos para recuperar informações, coletar métricas e navegar pelas estruturas dos sistemas (KULLBACH; WINTER, 1999; SILLITO; MURPHY; VOLDER, 2006).

Existem diversas abordagens para a realização de consulta em código fonte. As mais utilizadas são: (i) ferramentas de navegação por elementos de programas e (ii) ferramentas de consultas a um repositório de dados que contém informações de programas. Esta última abordagem pode utilizar um banco de dados relacional, um banco de dados

não relacional, XML ou DSLs projetadas para este fim (FOWLER; PARSONS, 2010; URMA; MYCROFT, 2012).

Existem diversas linguagens de consulta em código. As seções a seguir (2.3.1 a 2.3.9) descrevem as linguagens de consulta JQuery, Lost, JTL, .QL, CppDepend, Jackpot, PMD XPath e ActiveAspect.

2.3.1 JQuery

JQuery é linguagem para manipulação de código baseada em TyRuBa (VOLDER, 1998). Ela utiliza um banco de dados que armazena as informações dos sistemas e oferece mecanismos de manipulação através de um *plugin* para a plataforma Eclipse. Este *plugin* apresenta os resultados de uma consulta utilizando navegadores hierárquicos que mostram os nós e subnós da consulta, que podem ser pacotes, classes, métodos, variáveis, laços de repetição, dentre outras estruturas. A Listagem 2.4 mostra uma consulta realizada em JQuery, a qual busca todas as classes de um sistema que contém um método que inicia pela letra `"d"` (MCCORMICK; VOLDER, 2004).

```
1 class(?C), child(?C, ?M), child(?M, ?T),
2 type(?T), name(?M, ?name),
3 re_match(/^d/, ?name)
```

Listagem 2.4: Exemplo de consulta utilizando JQuery

2.3.2 Lost

Lost é uma linguagem de consulta baseada em OQL (*Object Query Language*) que permite realizar buscas em código Java e AspectJ (PFEIFFER; SARDOS; GURD, 2005). Ela é acessada através de um *plug-in* para a plataforma Eclipse que fornece uma navegação facilitada entre seus nós. Lost utiliza árvores para representar os sistemas de software e realizar as buscas. A Listagem 2.5 mostra um exemplo de consulta realizada em Lost, a qual busca todas as classes de um sistema que contém um método que inicia pela letra `"d"` (PFEIFFER; SARDOS; GURD, 2005).

```
1 select Class
2 where Class.hasMethod(Method)
3       && Method.hasName("^d")
```

Listagem 2.5: Exemplo de consulta realizada em Lost

2.3.3 JTL

JTL (*Java Tool Language*) é uma linguagem de consulta inspirada em *Query-by-Example* (ZLOOF, 1975) e projetada para selecionar elementos em código Java (COHEN; GIL; MAMAN, 2006). Para isso ela utiliza um banco de dados relacional que armazena as informações das ASTs. Ela pode ser utilizada tanto para programas orientados a objetos quanto para programas orientados a aspectos. A linguagem permite consultar módulos estruturais, suas inter-relações, comportamento condicional e outras diversas estruturas. O modelo armazenado é obtido através da inspeção de *bytecodes*. A Listagem 2.6 mostra uma consulta que busca por classes que tenham atributos do tipo `long` ou `int` e nenhum método abstrato (COHEN; GIL; MAMAN, 2006).

```
1  abstract class {
2  [long | int] field;
3  no abstract method;
4  }
```

Listagem 2.6: Exemplo de consulta utilizando JTL

2.3.4 .QL

A linguagem .QL é uma linguagem proprietária desenvolvida para realizar buscas por elementos em programas Java orientados a objetos. Ela utiliza um banco de dados relacional para armazenar as informações e consultas SQL para retorná-las. Uma consulta em .QL é transformada em SQL para ser executada em uma base de dados relacional. A .QL é utilizada na ferramenta de análise estática Semmle (SEEMLE, 2017). A Listagem 2.7 mostra um exemplo de consulta em .QL. Ela busca por tipos que definam métodos com o nome `compareTo`, mas que não possuam métodos com o nome `equals` (VERBAERE; HAJIYEV; MOOR, 2007).

```
1  from Method m
2  where m.hasName("compareTo") and
3  not(m.getDeclaringType().declaresMethod("equals"))
4  select m, "Is compareTo consistent with equals?"
```

Listagem 2.7: Exemplo de consulta utilizando .QL

2.3.5 CppDepend

CppDepend é uma ferramenta que facilita o gerenciamento de código C e C++ (CPPDEPEND, 2017). Ela usa Clang (CLANG, 2017) para realizar a análise sintática de

arquivos fontes e CQLinq (CQLINQ, 2017), que é baseado no Linq (LINQ, 2017), para realizar as consultas em código. É possível buscar pelas estruturas do sistema, como classes, métodos e tipos. Fornece 82 métricas que podem ser utilizadas nas consultas, tais como: número de linhas do código, porcentagem de comentários e número de variáveis. A Listagem 2.8 demonstra uma consulta em CQLinq. Ela busca por todos os métodos de uma aplicação, os quais possuem modificadores de acesso público e possuem mais de 30 linhas no corpo do método (CPPDEPEND, 2017).

```
1 from m in Application.Methods
2 where m.NbLinesOfCode > 30
3     && m.IsPublic
4 select m
```

Listagem 2.8: Exemplo de consulta utilizando CppDepend

2.3.6 Jackpot

Jackpot é um módulo para o NetBeans que permite realizar consultas e transformações em código Java (JACKPOT, 2017). Através dele, é possível detectar padrões, utilizando análise sintática, e implementar regras de transformação. Por exemplo, o método `loadImage` da classe `Utilities` foi substituído pelo método `loadImage` da classe `ImageUtilities`. A Listagem 2.9 mostra a automatização da alteração do método antigo para o novo.

```
1 org.openide.util.Utilities.loadImage($loc)
2 =>
3 org.openide.util.ImageUtilities.loadImage($loc)
4 ;;
```

Listagem 2.9: Exemplo de refatoração de um método com Jackpot

2.3.7 PMD

PMD é uma linguagem utilizada para realizar análise em código fonte (PMD, 2017). Ela oferece suporte a Java, JavaScript, PHP, entre outras. Através de expressões *XPath* é possível detectar potenciais anomalias, tais como código duplicado e expressões complexas. A Listagem 2.10 mostra um exemplo de código *PMD XPath*, o qual busca por todas as variáveis locais existentes em um sistema analisado (PMD, 2017).

```
1 //LocalVariableDeclaration
```

Listagem 2.10: Exemplo de consulta utilizando PMD XPath

2.3.8 ActiveAspect

A ferramenta ActiveAspect (COELHO; MURPHY, 2005) foi desenvolvida em AspectJ como um *plug-in* para o Eclipse. Ela apresenta informações sobre interesses transversais por meio de diagramas de estruturas. Nessa ferramenta, o usuário pode interagir com o modelo de visualização revelando novos relacionamentos e informações dos elementos. O ponto inicial de visualização é definido por um conjunto de partes da estrutura do programa. Essas estruturas são baseadas em propriedades estruturais, tais como o tipo de elemento (classe ou membro), o tipo de membro (campo, método ou construtor), a visibilidade do membro (pública, protegida, privada), entre outros.

2.3.9 Mylar

Mylar (KERSTEN; MURPHY, 2005) é uma ferramenta que prioriza a visualização dos elementos de um programa através de navegadores customizados. Ela não possui uma linguagem explícita para consulta e pode ser utilizada para programas Java e AspectJ. Através das atividades do programador, a ferramenta mapeia o grau de relevância dos elementos de acordo com a tarefa executada. Através deste mapa, a ferramenta preenche as visões de Java e AspectJ na plataforma Eclipse, usando um espectro de cores. A apresentação ocorre por meio de visões estruturadas.

2.4 AQL

A AQL (*Aspect Query Language*) é uma linguagem declarativa projetada para realizar consultas em código fonte tendo como referência um metamodelo que representa sistemas orientado a aspectos e sistemas orientado a objetos (FAVERI, 2013). O metamodelo é um conjunto de classes que armazenam informações do projeto e das unidades de compilação das ASTs dos projetos, tais como: pacotes do projeto, classes, atributos das classes, métodos das classes e demais estruturas de um programa java (FAVERI, 2013).

Os benefícios do uso desta linguagem são: (i) realizar consultas menores, quando comparadas a abordagens como SQL (*Structured Query Language*) ou OQL (*Object Query Language*), (ii) usar conceitos que se assemelham a OQL, mantendo assim o paradigma orientado a objetos para a realização de consultas, (iii) evoluir o metamodelo sem a necessidade de modificar a sintaxe da linguagem e (iv) permitir o uso de funções contextualizadas que simplificam as consultas e reduzem a quantidade de instruções. O uso de AQL (em sua forma original) é restrito a consulta de elementos e suas relações, não oferecendo recursos para atualização, inserção ou remoção de dados (FAVERI, 2013).

O código AQL a seguir mostra um exemplo do uso da linguagem AQL, o qual busca por adendos que afetam classes de nome `Company`, que estejam associados a pontos de corte do tipo `call` e que estejam declarados em aspectos localizados em pacotes iniciados por `br.ufsm.core`. A consulta retorna o nome do aspecto, a posição do adendo dentro do aspecto e o tipo do adendo (`before`, `after` ou `around`) (FAVERI, 2013).

```
1  find aspect a, class c
2  where a.advice.affects(c)
3    and c.name = 'Company'
4    and a.advice.pointcut.kind(call)
5    and a.pack.name like 'br.ufsm.core%'
6  returns a.fullQualifiedName, a.advice.position, a.advice.kind
```

O uso de AQL é indicado em, pelo menos, quatro finalidades. São elas: (i) em ferramentas de análise estática, (ii) em ferramentas de pesquisa em IDEs, (iii) em ambientes de desenvolvimento, como o AJDT (*AspectJ Development Tools*) (AJDT, 2017), e (iv) em consultas complexas, envolvendo conectivos lógicos e relacionais, realizadas diretamente pelo programador.

A estrutura da linguagem permite consultar objetos de maior ordem e seus respectivos atributos. Esses objetos são definidos em AQL como `project`, `package`, `class`, `interface`, `enum` e `aspect`, os quais representam os modelos de projeto, de pacote, de classe, de interface, de enumerações e de aspecto, respectivamente. O resultado de uma consulta retorna um objeto ou uma lista de objetos, os quais podem ser ordenados de forma ascendente ou descendente (FAVERI, 2013).

A AQL se difere de outras OQLs por inferir construções de junção entre objetos, desonerando o usuário de explicitar as associações necessárias para executar determinadas consultas. Uma vez que o modelo a ser consultado seja conhecido pela AQL, as junções necessárias podem ser inferidas pelo compilador, tornando as consultas mais concisas e consequentemente mais simples de serem escritas (FAVERI, 2013).

2.4.1 Sintaxe da AQL

A linguagem AQL é formada por cinco cláusulas principais: `find`, `where`, `returns`, `order by` e `group by` (FAVERI, 2013). A Figura 2.3 mostra o grafo de sintaxe das principais cláusulas da linguagem AQL (FAVERI, 2013).

Figura 2.3 – Principais cláusulas da AQL (FAVERI, 2013)

A cláusula `find` declara os objetos de maior ordem que farão parte da consulta. Ela é obrigatória e permite associar um ou mais pseudônimos (`alias`) a cada objeto. Cada pseudônimo representa um objeto diferente do mesmo tipo na consulta, permitindo dessa forma, a junção implícita de objetos do mesmo tipo. Objetos sem pseudônimos não são permitidos (FAVERI, 2013).

A cláusula `where` utiliza expressões para aplicar filtros na consulta através de lógica relacional e cálculos aritméticos. Os objetos utilizados na cláusula `where` devem ser declarados na cláusula `find`. Cada consulta pode ter apenas uma cláusula `where`, mas múltiplas expressões encadeadas através de conectivos `or` ou `and` (FAVERI, 2013).

A cláusula `returns` permite definir os objetos, atributos ou valores que serão retornados na consulta. Essa cláusula é obrigatória e pode retornar *n* expressões separadas por vírgula (,). Ela usa os objetos declarados na cláusula `find` e as palavras-chave `distinct` e `all` (opcionais) (FAVERI, 2013).

A cláusula `order by` fornece recursos para ordenar o resultado de uma consulta por uma ou mais propriedades. Os objetos devem estar na cláusula `find` e cada propriedade deve ser separada por vírgula. Ela permite o uso dos modificadores `asc` e `desc` para ordenar em ordem ascendente ou descendente, respectivamente. Por padrão, a ordenação é realizada de forma ascendente (FAVERI, 2013).

A cláusula `group by` permite agrupar os dados de uma consulta. É possível utilizar as seguintes funções de agregação nesta cláusula: `SUM`, `COUNT`, `AVG`, `MIN` e `MAX`.

Para comentar instruções em AQL, utilizam-se os mesmos caracteres da linguagem Java, dupla barra (`//`) para comentários de uma linha e `/*...*/` para comentários em bloco. Quantos aos pseudônimos, eles devem iniciar por uma letra (maiúscula ou minúscula) ou *underline* (_) seguido por letras, números ou *underline* (_). Eles não podem ser palavras reservadas da linguagem. Os literais são valores utilizados em expressões ou em parâmetros de funções na AQL. Ela reconhece duas categorias: numéricos e *string*. A AQL permite utilizar nomes qualificados para acessar métodos e atributos através do caractere ponto (.), assim como em Java (FAVERI, 2013).

2.5 ARQUITETURA DA IMPLEMENTAÇÃO DE REFERÊNCIA DE AQL

Para implementar a AQL, dois projetos foram desenvolvidos utilizando um conjunto de tecnologias relacionadas à plataforma Eclipse (FAVERI, 2013). O primeiro projeto, o AOPJungle, é responsável pela extração, abstração e mapeamento de informações dos programas para o metamodelo. Ele foi implementado utilizando as linguagens de programação Java e AspectJ.

O segundo projeto é a implementação do compilador e do editor de consultas para AQL. O compilador AQL realiza as transformações de AQL para HQL (*Hibernate Query*

Language). HQL é a linguagem de consulta do *framework* Hibernate, utilizada para realizar buscas por objetos persistidos em um banco de dados (BAUER; KING, 2004). Ambos, o compilador e o editor de consultas, foram construídos usando Xtext (EYSHOLDT; BEHRENS, 2010) .

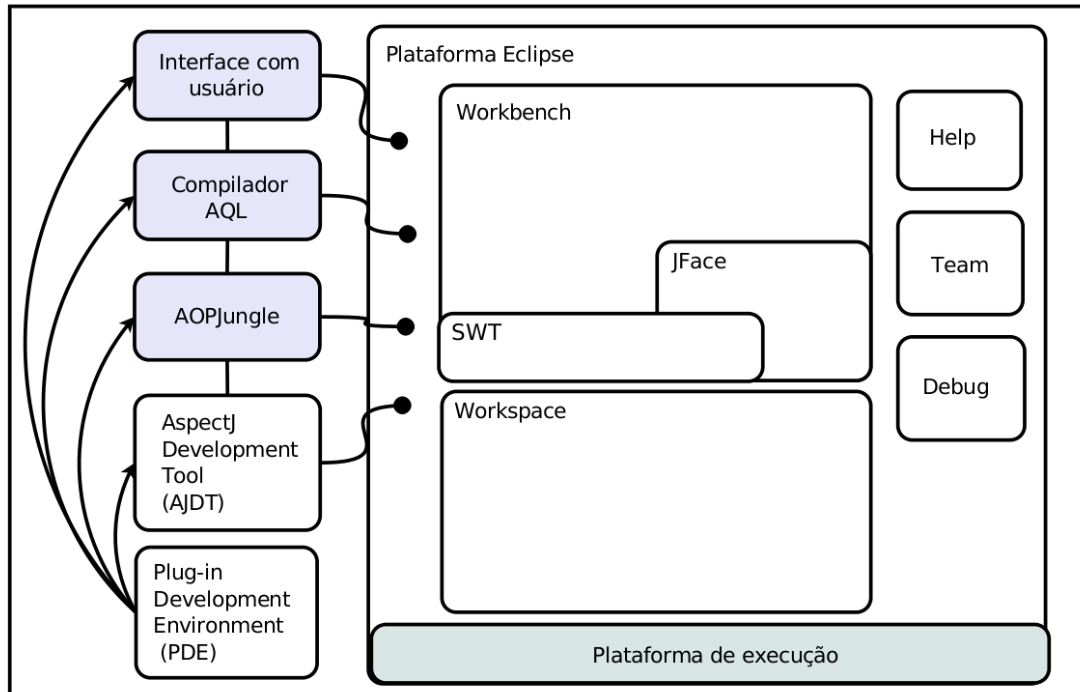


Figura 2.4 – Visão geral da implementação de referência de AQL integrada ao Eclipse (FAVERI, 2013)

Ambos os projetos foram implementados como *plugins* para a plataforma Eclipse. Tal plataforma fornece a interface com usuário, a plataforma de execução (*runtime platform*) e um conjunto de *plugins*, os quais são descobertos, carregados e executados pela plataforma de execução.

A Figura 2.4 mostra a visão geral da implementação de referência AQL integrada à Plataforma Eclipse. O módulo de interface com o usuário foi idealizado para a escrita das consultas em AQL. Uma vez que a consulta é escrita, o compilador inicia o processo de análise e transformação. A saída desta compilação gera o código fonte correspondente à consulta em HQL. O código HQL obtido é repassado para o *framework* AOPJungle, o qual realiza a execução da consulta no metamodelo retornando as informações solicitadas (FAVERI, 2013).

2.5.1 AOPJungle

O AOPJungle é um *framework* desenvolvido, fundamentalmente, para instanciar o metamodelo de programas escritos em AspectJ. A partir da representação de um programa

fonte como um modelo de objetos, o AOPJungle admite receber e executar consultas sobre este modelo. Sua arquitetura pode ser visualizada na Figura 2.5.

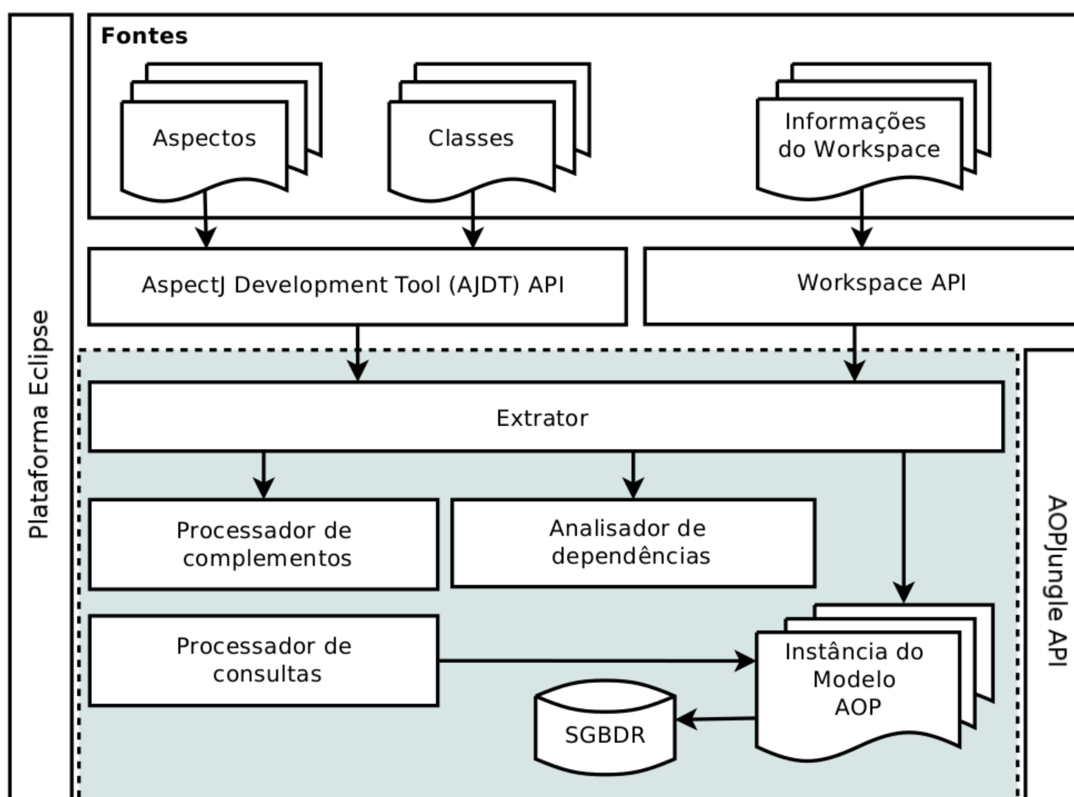


Figura 2.5 – Arquitetura do AOPJungle (FAVERI, 2013)

O módulo *Extrator* é responsável pela análise do código fonte de cada projeto do ambiente Eclipse e por instanciá-lo no metamodelo. Primeiramente, os projetos em AspectJ e Java, disponíveis no *workspace*, são identificados e mapeados para uma lista. A partir desta lista, o *Extrator* utiliza a *API* da *AJDT* (AJDT, 2017) para obter os pacotes e as unidades de compilação. Cada unidade de compilação representa um arquivo contendo o código fonte do programa a ser analisado. Uma vez que uma unidade de compilação está disponível, é possível percorrer sua AST (*Abstract Syntax Tree*) (KUHN; THOMANN, 2017) em busca das informações necessárias para a instanciação do metamodelo (FAVERI, 2013).

O AJDT fornece um mecanismo de acesso a ASTs para obter informações de cada nó em uma árvore. Esse mecanismo é implementado pelo padrão *Visitor* (GAMMA et al., 2000), que possibilita a execução de uma determinada lógica para cada elemento acessado. Para acessar a AST é necessário estender a classe abstrata `AjASTVisitor`[2]. Os nós são visitados de cima para baixo, da esquerda para a direita, de acordo com sua hierarquia, até alcançar os nós folha. Para cada nó visitado é realizado o mapeamento para o metamodelo (AJDT, 2017).

[2]https://github.com/eclipse/org.aspectj/blob/master/org.aspectj.ajdt.core/src/org/aspectj/org/eclipse/jdt/core/dom/AjASTVisitor.java

O *Analisador de dependências* é o módulo responsável por construir as associações e dependências entre os elementos do modelo. Ele é chamado após o *Extrator* a fim de acrescentar as informações de ligação entre os elementos, tais como: relações de adendos com os pontos de junção, declarações intertipos e modificações hierárquicas (FAVERI, 2013).

O *Processador de complementos* é responsável por gerenciar os módulos que serão executados após a instanciação do metamodelo, incluindo, por exemplo, o cálculo de métricas e estatísticas de um programa escrito em *AspectJ*. O processador de consultas é responsável pela execução das consultas no metamodelo instanciado. Para isso, o cliente deve indicar o manipulador do executor e o código fonte da consulta. Por exemplo, para executar consultas em HQL, deve-se realizar a seguinte chamada (FAVERI, 2013):

```
1       aopjungle.query ("hql", "consulta-hql...");
```

A Figura 2.6 mostra uma versão simplificada do metamodelo do AOPJungle. Observa-se que ele possui apenas informações relevantes para a linguagem AQL original (FAVERI, 2013), como informações de projeto, pacote, classe, interface, aspecto e tipos `enum`.

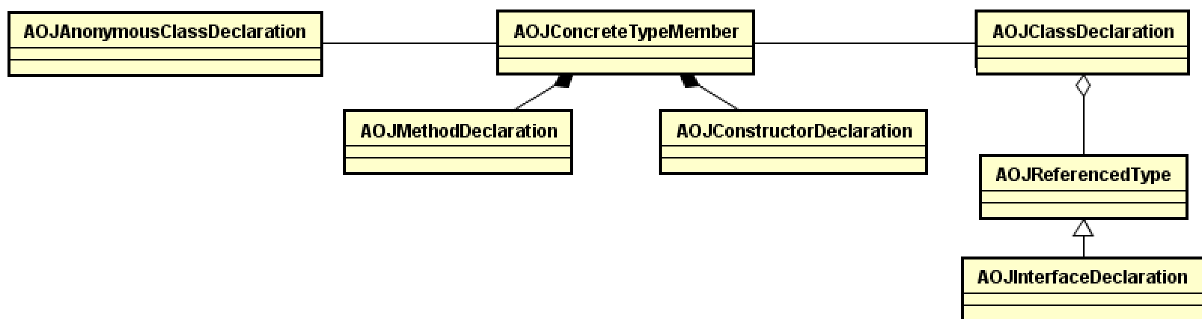


Figura 2.6 – Metamodelo original e simplificado do AOPJungle

2.5.2 Compilador de AQL

O compilador original de AQL foi desenvolvido utilizando o *framework* para desenvolvimento de linguagens de programação e DSLs Xtext (EYSHOLDT; BEHRENS, 2010). Ele oferece recursos que facilitam a geração de código por meio de *String Templates* (BETTINI, 2013). O compilador, inicialmente, recebe como entrada o fonte AQL, realiza três fases - análise estática, tradução e emissão - do processo de compilação e, por fim, devolve como saída o código fonte da linguagem alvo, neste caso, o código HQL. Essa arquitetura pode ser visualizada na Figura 2.7.

A sintaxe concreta da AQL foi definida através da formalização de uma gramática EBNF (*Extended Backus-Naur Form*) em Xtext. As regras desta gramática são classificadas como regras terminais e não terminais. As regras terminais são símbolos atômicos uti-

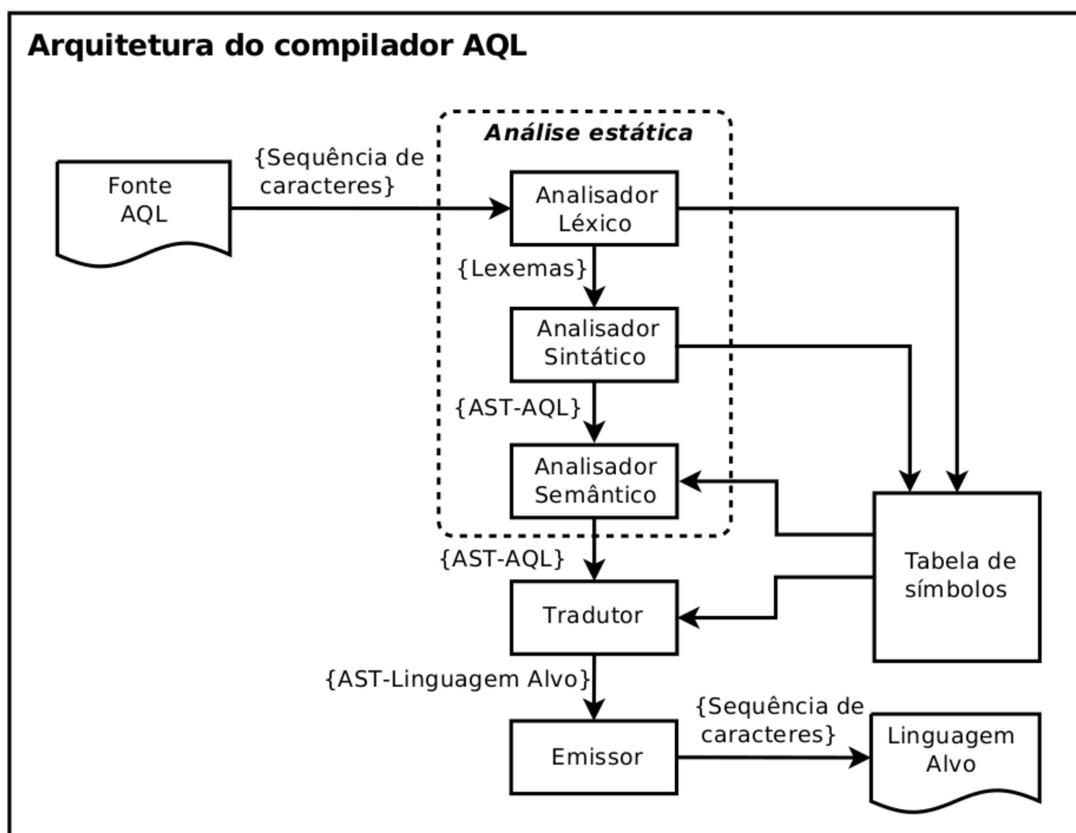


Figura 2.7 – Arquitetura do Compilador AQL (FAVERI, 2013)

lizados na linguagem. As regras não terminais produzem uma árvore de *tokens* terminais e não terminais. O Xtext deriva o modelo da AST da linguagem AQL a partir da descrição de sua gramática em EBNF. À medida que a sequência de caracteres é consumida, o modelo é instanciado com os objetos representativos de cada elemento da linguagem (BETTINI, 2013).

Na análise estática são realizadas as análises léxica, sintática e semântica da linguagem. Elas são realizadas por analisadores gerados automaticamente pelo Xtext, a partir da gramática formalmente descrita. O Xtext delega a função de geração dos analisadores léxico e sintático ao ANTLR (PARR, 2013), um programa que possibilita a geração de analisadores para serem utilizados na construção de compiladores e interpretadores. O processo de tradução da gramática para o formato reconhecido pelo ANTLR é realizado durante a geração dos artefatos de Xtext. A partir da gramática traduzida, o ANTLR gera os analisadores léxico e sintático correspondentes (BETTINI, 2013).

A análise semântica é parte da estrutura de validações customizadas do Xtext. Esse processo satisfaz duas áreas: (i) referências cruzadas: a análise de referências cruzadas é responsável por garantir que os identificadores sejam corretamente declarados antes de sua utilização. (ii) funções internas: toda função contextualizada em uma consulta AQL é mapeada como uma função interna, a qual possui um conjunto de regras associadas que devem ser observadas pelo analisador semântico. Ou seja, ele verifica se o objeto e a

função associada são compatíveis (FAVERI, 2013).

A tradução é responsável pela transformação do modelo AQL (fonte) para o modelo HQL (destino). Esse processo é realizado por meio da aplicação de regras de transformação entre os modelos. Essas podem ser: (i) transformações diretas, as quais possuem o padrão de origem idêntico ao padrão destino; ou (ii) transformações múltiplas, as quais não possuem equivalência semântica, demandando a criação de múltiplos padrões no destino. O processo de transformação é realizado em duas fases. A fase 1 resolve as transformações diretas e a fase 2 resolve as transformações múltiplas. Um exemplo de transformação pode ser observado na Tabela 2.1. A consulta gerada em HQL (fase 2) é executada no metamodelo gerado em memória pelo AOPJungle (FAVERI, 2013).

Tabela 2.1 – Fases de transformação de código (FAVERI, 2013)

Fase	Código

Entrada (AQL)	`find aspect a`
	`  where a.modifier (public,protected)`
	`  returns a.name, a.modifier.name`

Fase 1 (Intermediário)	`SELECT a.name, a.modifier.name`
	`  FROM AOJAspectDeclaration a`
	`  WHERE a.modifier(public,protected)`

Fase 2 (HQL)	`SELECT a.name, aojmodifier_2`
	`  FROM AOJAspectDeclaration a`
	`    LEFT JOIN a.modifiers aojmodifier_1`
	`    INNER JOIN aojmodifier_1.descriptors aojmodifier_2`
	`  WHERE aojmodifier_2 in ( 'PUBLIC', 'PROTECTED' )`

A transformação entre modelos baseia-se no princípio de que existam pré-condições, pós-condições e invariantes que devam ser satisfeitas antes e após o processo de transformação do modelo. Nesse sentido, um programa pode ser expresso como um grafo orientado atributivo tipado, no qual os vértices representam os elementos do programa e as arestas representam as relações entre esses elementos. Cada aresta pode ser dirigida e nomeada representando uma relação única entre dois elementos e os vértices podem conter atributos que os identificam unicamente (FAVERI, 2013).

A partir da representação dos modelos, AQL e HQL, as regras de transformação foram especificadas de acordo com alguns conceitos: (i) pré-condição (*Left Hand Side Rule* - LHS): representa um subgrafo do grafo de origem que será utilizado para descrever os padrões candidatos a transformação; (ii) pós-condição (*Right Hand Side Rule* - RHS): são os resultados pretendidos, obtidos pela manutenção, exclusão ou geração de novos elementos; (iii) condições de aplicação negativa (*Negative Application Condition* - NAC): são restrições aos cenários pretendidos na transformação do grafo. Uma NAC representa uma condição a qual a transformação somente ocorre quando a mesma for falsa, mesmo na presença de uma padrão candidato à transformação; (iv) condições de aplicação positiva (*Positive Application Condition* - PAC): é aplicada em uma pós condição, ou seja, somente ocorrerá a transformação caso uma condição no modelo de saída estiver presente (FA-

VERI, 2013).

Para apoiar a implementação das transformações entre os modelos, uma formalização das regras de transformação foi realizada com o *framework* EMorF (KLASSEN; WAGNER, 2012), o qual baseia-se na transformação algébrica de grafos e possui recursos para a transformação de modelos EMF (*Eclipse Modeling Framework*). A Figura 2.8 ilustra a transformação da cláusula *where* utilizando o *framework* EMorF (FAVERI, 2013).

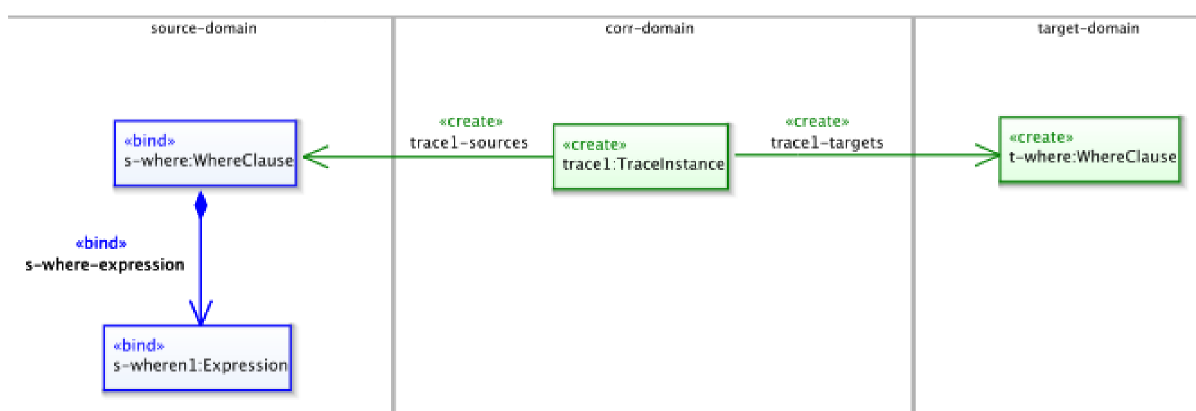


Figura 2.8 – Ilustração da transformação da cláusula `where` (FAVERI, 2013)

A emissão é a última etapa da compilação. Ela realiza a transformação da AST da HQL em texto. O emissor recebe uma AST e percorre seus nós e sub nós, identificando cada tipo e seu modelo correspondente dentre as expressões de modelo. A inspeção da AST é iniciada pelo primeiro nó da árvore (raiz), sendo percorrida de cima para baixo, da esquerda para a direita, até o último nó folha.

As Listagens 2.11 e 2.12 apresentam, respectivamente, uma consulta em AQL e sua correspondente em HQL. Ambas as consultas buscam por laços de repetição `for each` que implementam um filtro através de um condicional.

```
1  find if i
2      where i.owner.metaName = 'Foreach' and i.owner.isCollection = true
3      returns i.owner
```

Listagem 2.11: Busca por laços de repetição `for each` que implementam um filtro através de um condicional

```
1  SELECT aojowner_1
2      FROM AOJIfStatement e
3      LEFT JOIN e.owner aojowner_1
4      WHERE aojowner_1.metaName = 'Foreach' AND aojowner_1.isCollection = true
```

Listagem 2.12: Consulta HQL que busca por laços de repetição `for each` que implementam um filtro através de um condicional

Comparando o tamanho das consultas, observa-se que a consulta em AQL apresenta uma redução de expressões quando comparado à respectiva consulta em HQL. As

reduções ocorrem em consultas que possuem inferências de junção (FAVERI, 2013), como pode ser observado na linha 3 da Listagem 2.12. Nota-se que os códigos AQL e HQL são similares quando a inferência de junção não é necessária.

2.6 CONSIDERAÇÕES FINAIS

Este capítulo discutiu alguns conceitos necessários para o entendimento desta dissertação. Inicialmente, o conceito de refatoração foi apresentado, bem como as etapas do processo de aplicar refatorações em um sistema. Posteriormente, foram apresentados detalhes sobre DSLs, demonstrando seus principais conceitos e termos, as possíveis categorias de uma DSL, as vantagens e desvantagem em projetar ou utilizar DSLs e alguns exemplos de DSLs. Logo após, foram apresentadas algumas definições sobre linguagem de consulta em código fonte, algumas abordagens utilizadas e alguns exemplos destas linguagens. Por fim, este capítulo apresentou detalhes da arquitetura da linguagem AQL e os dois projetos utilizados, o *framework* AOPJungle e o compilador de referência de AQL, os quais foram utilizados para o desenvolvimento deste trabalho. O capítulo seguinte trata da especificação da extensão e das melhorias propostas para a linguagem AQL e sua implementação de referência.

3 EXTENSÕES PARA AQL

As linguagens de consulta em código fonte são linguagens de computador utilizadas para realizar buscas em estruturas que representam um sistema de software. Essas linguagens, normalmente, são utilizadas por desenvolvedores para compreender os sistemas, identificar seus defeitos e realizar buscas por oportunidades de refatoração.

Recentemente, no Laboratório de Linguagens de Programação e Bancos de Dados (GLPBD) da UFSM, foi projetada, especificada e implementada uma DSL para realizar consultas em código fonte orientado a aspectos denominada AQL (*Aspect Query Language*) (FAVERI, 2013). Analisando a linguagem AQL, foram identificadas possibilidades de melhorias, tais como: (i) a possibilidade de refinar a granularidade das consultas, (ii) a inserção de instruções DML (*Data Manipulation Language*), (iii) a possibilidade de melhorar algumas validações da linguagem e (iv) a possibilidade de aprimorar algumas cláusulas.

Este capítulo descreve as especificações e as implementações de um conjunto de melhorias proposto para a linguagem AQL a fim de suprir tais limitações. Dessa forma, os principais objetivos a serem alcançados pela extensão de AQL são:

- **Permitir buscas de granularidade mais fina.** Atualmente não é possível realizar buscas por unidades de código contidas dentro de métodos e construtores, como por exemplo, laços de repetição, condicionais e atribuições. A fim de disponibilizar tais possibilidades, foram realizadas algumas modificações na especificação da linguagem, bem como no compilador original da linguagem AQL e no metamodelo do *framework* AOPJungle. Tais modificações permitem realizar buscas com granularidade mais fina que a versão original da linguagem.

- **Permitir o uso de cláusulas para manipulação dos dados.** A especificação original da linguagem AQL não possui cláusulas para inserir, atualizar ou remover elementos. Através destes novos recursos, o desenvolvedor poderá modificar os sistemas analisados.

- **Aprimoramentos da linguagem.** Foram identificados e aprimorados alguns elementos da linguagem AQL e sua implementação de referência, tais como: (i) melhoria da cláusula `order by`, (ii) adição de novas validações e (iii) melhoria na exibição de erros.

- **Manter a facilidade de realizar consultas.** Uma das principais diferenças da AQL em relação às OQLs é a sua capacidade de inferir construções de junção entre objetos, a qual facilita a realização das buscas por elementos do metamodelo. A fim de manter esta facilidade, foram adicionadas funções de transformações para os novos elementos inseridos na linguagem.

Para isso os dois subprojetos que formam a implementação de referência da linguagem AQL foram modificados. O *framework* AOPJungle, responsável por extrair e fornecer informações sobre o código fonte dos sistemas analisados, e o compilador da AQL, responsável por realizar a transformação das instruções AQL para uma linguagem alvo.

A implementação original da linguagem AQL utilizava HQL como linguagem alvo, ou seja, uma consulta AQL era transformada em uma consulta HQL e executada pelo processador de consultas do AOPJungle. Entretanto, durante a implementação das cláusulas de manipulação de dados, verificou-se que, até o momento, as instruções de manipulação de dados da HQL são limitadas. De acordo com a documentação do Hibernate (HIBERNATE, 2017) as cláusulas de manipulação de dados:

- Não permitem a inserção de novos elementos, somente de elementos a partir de outros objetos já persistidos no banco de dados;

- O comando `insert` do HQL permite acessar somente as propriedades a nível da classe (as propriedades da superclasse não são permitidas);

- Os comandos `delete` e `update` não permitem o uso de junções (`joins`), tanto implícito, quanto explícito;

- De acordo com a documentação, os pontos de junções podem ser utilizados em subconsultas nas cláusulas `delete` e `update`, porém, durante o desenvolvimento, verificou-se que o Hibernate não consegue realizar a transformação de HQL para SQL quando pontos de junções são utilizados na subconsulta.

Dadas tais limitações, optou-se por utilizar Hibernate e Java para realizar as transformações das consultas de manipulação de dados, além de HQL. Ou seja, a linguagem alvo do compilador AQL passa ser HQL, Hibernate e Java. O uso destas tecnologias permitiu a implementação dos novos recursos[1].

A principal ferramenta utilizada na extensão do compilador AQL foi o *workbench* Xtext, o qual gerencia todo seu ciclo de implementação. Ele é estreitamente ligado ao projeto da linguagem Xtend[2]. Ela fornece recursos que facilitam a escrita de programas, por meio de novas funcionalidades, tais como: expressões lambdas, inferência de tipos e *String Templates* (XTEND, 2017).

Para executar as instruções geradas pelo compilador AQL, o *framework* AOPJungle foi estendido. Ele usa as ferramentas JDT - *Java Development Tools* e AJDT - *AspectJ Development Tools*, as quais fornecem meios para acessar às informações de unidades de compilação dos códigos Java e AspectJ (FAVERI, 2013).

[1] O módulo de consulta, desenvolvido no AOPJungle, precisa ser atualizado pois ele executa apenas consultas HQL.

[2] http://www.eclipse.org/xtend/

As próximas seções apresentam as especificações dos novos recursos e melhorias da linguagem AQL e as modificações realizadas no compilador AQL e no *framework* AOP-Jungle. Este capítulo está organizado da seguinte forma: A Seção 3.1 define os novos elementos de alta ordem da linguagem AQL e descreve os detalhes das modificações implementadas para adicionar tais recursos, a Seção 3.2 apresenta a definição e descreve os detalhes da implementação das novas cláusulas de manipulação de dados, a Seção 3.3 define melhorias e apresenta as modificações realizadas na cláusula `order by`, a Seção 3.4 define e descrever os detalhes das validações customizadas adicionadas na implementação de referência da linguagem AQL e a seção 3.5 descreve as modificações realizadas no *framework* AOPJungle para fornecer informações contidas dentro de métodos e construtores dos programas.

3.1 MODIFICANDO A GRANULARIDADE DAS BUSCAS

A estrutura da linguagem AQL permite realizar consultas selecionando elementos de maior ordem, também chamados de objetos recipientes, e seus respectivos atributos. Na especificação original da linguagem, esses elementos eram definidos como `project`, `package`, `class`, `interface`, `enum` e `aspect`, os quais representam, respectivamente, os modelos de projeto, de pacote, de classe, de interface, de enumeração e de aspecto.

Um dos objetivos deste trabalho é possibilitar consultas com granularidade mais fina, ou seja, possibilitar a realização de consultas por unidades menores de código, como por exemplo, buscar por métodos, laços de repetição, laços condicionais e expressões. Quando a linguagem AQL foi projetada, não havia interesse de realizar consultas com esses tipos de elementos, por isso eles não foram implementados.

A fim de diminuir a granularidade, novos elementos de maior ordem foram adicionados ao elemento `ObjectType` da gramática original da AQL (Anexo A). São eles: `method`, `constructor`, `foreach`, `while`, `if`, `switch`, e `expression`, os quais representam, respectivamente, métodos dentro de classes, construtores, laços de repetição, estruturas condicionais e expressões.

A Listagem 3.1 mostra três exemplos de buscas válidas em AQL utilizando os novos elementos propostos. A linha 1 busca por todos os métodos existentes nos sistemas analisados, a linha 2 busca por todos os laços de repetição `for each` e a linha 3 busca por todos os elementos condicionais `if`. Estas consultas retornam uma lista de objetos do respectivo tipo da consulta.

```
1 find method m returns m
2 find foreach f returns f
```

```
3  find if i returns i
```

Listagem 3.1: Exemplos de consultas `find` utilizando os novos tipos

A Figura 3.1 representa as modificações realizadas no elemento `ObjectType`. Os elementos usados na AQL original estão representados por retângulos com preenchimento branco, enquanto os elementos adicionados estão representados por retângulos com preenchimento cinza.

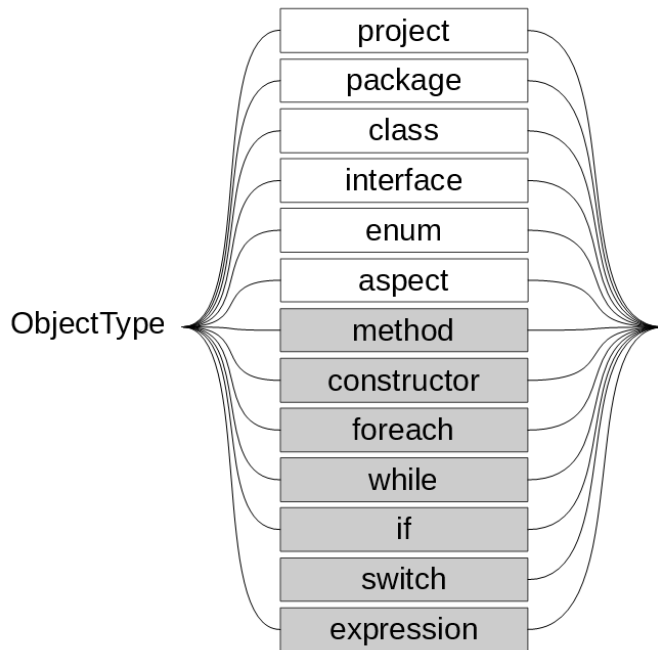


Figura 3.1 – Grafo de sintaxe do elemento `ObjectType`

A modificação deste elemento teve pelo menos duas motivações. A primeira, expandir a linguagem AQL com o intuito de realizar buscas por oportunidade de refatoração que não eram possíveis usando a AQL original, como por exemplo as refatorações relacionadas às expressões lambda de Java 8 (TEIXEIRA JÚNIOR, 2014) e realizar buscas por refatorações para aplicação de padrões de projeto (KERIEVSKY, 2008; PAULI, 2014).

A segunda motivação é fornecer recursos necessários para manipular as unidades de compilação do metamodelo. Comumente, nos catálogos de refatorações, os autores descrevem uma sequência de passos detalhados para aplicar determinada refatoração em um sistema. Esses passos são chamados de mecânica (FOWLER et al., 1999). Por exemplo, a mecânica da refatoração *Convert Enhanced For to Lambda Enhanced For* (TEIXEIRA JÚNIOR, 2014) é descrita por quatro passos:

1. Substitua a construção `for each` pela chamada do método `forEach` da coleção;

2. Implemente a interface funcional `Consumer` requerida pelo método `forEach`, movendo o conteúdo da estrutura de repetição para o método da interface `Consumer`;

3. Aplique a refatoração *Convert Functional Interface Instance to Lambda Expression* para a instância da interface `Consumer`;

4. Compile o código e teste.

A Listagem 3.2 mostra um exemplo de aplicação da refatoração *Convert Enhanced For to Lambda Enhanced For* em um código Java. As linhas 2, 3 e 4 mostram o código antes de aplicar a refatoração. Ele utiliza um laço de repetição `for each` para percorrer uma lista de objetos do tipo `Person` e mostrar na saída padrão do sistema o atributo `name` de cada objeto da lista. As linhas 6 e 7 mostram o código após aplicar a refatoração. Ele utiliza o método `forEach` da coleção e uma expressão lambda para percorrer a lista e mostrar o atributo `name` (TEIXEIRA JÚNIOR, 2014).

```
1  // Antes
2  Collection<Person> people = ...
3  for (Person person : people)
4      System.out.println(person.getName());
5  // Depois
6  Collection<Person> people = ...
7  people.forEach(person -> System.out.println(person.getName()));
```

Listagem 3.2: Exemplo de aplicação da refatoração *Convert Enhanced For to Lambda Enhanced For*

Observa-se a aplicação da refatoração *Convert Enhanced For to Lambda Enhanced For* utilizando as novas funcionalidades da AQL na Listagem 3.3. Inicialmente foi escrita uma consulta `delete` para remover a estrutura `for each` (linha 1). Posteriormente, foi escrita uma consulta `insert` para inserir a implementação do método `forEach` da coleção e da expressão lambda (linha 2).

```
1  delete foreach f where f.id = 2
2  insert expression e (owner, codeBody) values (object method 1, 'people.forEach(
       person -> System.out.println(person.getName()))')
```

Listagem 3.3: Utilizando as novas funcionalidades da AQL para aplicar a refatoração *Convert Enhanced For to Lambda Enhanced For*

3.1.1 Adicionando novos elementos de busca

O compilador AQL utiliza uma gramática EBNF para gerar os analisadores léxicos e sintáticos (PARR, 2013). Para adicionar os novos elementos de buscas, foram realizadas modificações na gramática do compilador.

Dentre os elementos da gramática, o elemento `BidingObject` é responsável por associar o tipo do elemento de alta ordem a um ou mais pseudônimos. Esses tipos são definidos pelo elemento `ObjectType`. A Listagem 3.4 mostra a regra do elemento `ObjectType`. Ela é descrita por uma lista de palavras reservadas que indicam os tipos dos objetos utilizados nas principais cláusulas de AQL. Nesta regra, foram adicionados os elementos `method`, `constructor`, `foreach`, `while`, `if`, `switch` e `expression`. Eles podem ser observados entre as linhas 8 e 14 da Listagem 3.4. Entre as linhas 2 e 7 são mostradas as palavras reservadas para os objetos de maior ordem que fazem parte da implementação original da gramática da linguagem AQL.

```
1  ObjectType:
2      {Project}            value='project'
3      | {Package}          value='package'
4      | {Class}            value='class'
5      | {Aspect}           value='aspect'
6      | {Interface}        value='interface'
7      | {Enum}             value='enum'
8      | {Method}           value='method'
9      | {Constructor}      value='constructor'
10     | {Foreach}          value='foreach'
11     | {While}            value='while'
12     | {If}               value='if'
13     | {Switch}           value='switch'
14     | {Expr}             value='expression'
15 ;
```

Listagem 3.4: Definição de sub-regras para o elemento ObjectType

O compilador AQL utiliza regas de transformação entre modelo baseadas em pré-condições, pós-condições e invariantes para transformar o código AQL em HQL. Estas regras dividem-se em transformações diretas (1 para 1) e múltiplas (1 para N). As transformações diretas convertem os padrões do modelo AQL para os padrões do modelo HQL. As transformações múltiplas resolvem os nomes qualificados. Como foram adicionados objetos de alta ordem na AQL, as respectivas transformações foram implementadas no compilador, de acordo com o metamodelo de referência.

A Listagem 3.5 mostra as modificações realizadas no arquivo de mapeamento de módulos de transformação. Foram adicionadas regras referentes aos elementos de alta ordem (linhas 5 a 11), as quais associam os tipos dos elementos da consulta de acordo com as classes do metamodelo e os nomes qualificados aos módulos de transformação (linhas 12 e 13). As classes de transformações implementadas encontram-se no Anexo D desta dissertação.

```
1  <aojql-mapping>
2      <bindings>
```

```
3         (...)
4         <bind in="method" out="AOJMethodDeclaration"/>
5         <bind in="constructor" out="AOJConstructorDeclaration"/>
6         <bind in="foreach" out="AOJEnhancedForStatement"/>
7         <bind in="while" out="AOJWhileStatement"/>
8         <bind in="if" out="AOJIfStatement"/>
9         <bind in="switch" out="AOJSwitchStatement"/>
10        <bind in="expression" out="AOJExpressionStatement"/>
11        (...)
12        <bind out="br.ufsm.hql.transformation.HqlOwnerFunction" in="
              AOJEnhancedForStatement.owner"/>
13        <bind out="br.ufsm.hql.transformation.HqlStatmentsFunction" in="
              AOJEnhancedForStatement.statments"/>
14        (...)
15    </bindings>
16 </aojql-mapping>
```

Listagem 3.5: Fragmento do arquivo de mapeamento de módulos de transformação

A consulta da Listagem 3.6 mostra um exemplo de busca utilizando os novos tipos de elementos de AQL. Esta consulta seleciona todos os elementos `foreach` que estão contidos dentro de blocos condicionais `if`. O elemento do tipo `foreach` é resolvido para a classe `AOJEnhancedForStatement` do metamodelo, de acordo com a regra de transformação mostrada na linha 7 da Listagem 3.5. Os atributos `in` e `out` são responsáveis por associar os elementos as classes correspondentes do metamodelo. Por fim, na linha 14, o nome qualificado `AOJEnhancedForStatement.owner` é resolvido para a função de transformação `br.ufsm.hql.transformation.HqlOwnerFunction` (Anexo D), a qual transforma o nome qualificado em um ponto junção (`join`). Os atributos `in` e `out` também são responsáveis por associar os nomes qualificados as funções de transformação.

```
1 find foreach f where f.owner.metaName = 'if' returns f
```

Listagem 3.6: Exemplo de consulta AQL utilizando `foreach`

A Listagem 3.7 mostra a consulta em HQL da Listagem 3.6 gerada pelo compilador AQL. Observa-se na linha 1 a transformação do elemento `foreach` para a classe `AOJEnhancedForStatement` que representa os laços de repetição `for each` dos programas no metamodelo. E na linha 2 a adição do ponto de junção que relaciona a classe `AOJEnhancedForStatement` com seu proprietário (`owner`).

```
1 SELECT f FROM AOJEnhancedForStatement f
2     LEFT JOIN f.owner aojowner_1 WHERE aojowner_1.metaName = 'if'
```

Listagem 3.7: Transformação da consulta AQL da Listagem 3.6 em HQL

3.2 NOVAS CLÁUSULAS

A implementação original da linguagem AQL não possuía foco na definição e manipulação de dados. Seu uso era restrito a consultas de elementos e relações (FAVERI, 2013). Um dos objetivos deste trabalho é fornecer recursos para manipular os elementos do metamodelo, o qual representa os sistemas de software analisados. Uma das formas de manipular tais informações é utilizar uma linguagem para manipulação de dados (DML - *Data Manipulation Language*), contendo cláusulas que possibilitem a inserção, a atualização e a remoção dos elementos.

Para permitir a realização das consultas, o grafo de sintaxe das expressões AQL foi modificado. A Figura 3.2 apresenta o grafo do elemento `QueryStatement` da especificação original de AQL. Nele, as cláusulas `find` e `returns` são obrigatórias, enquanto `where`, `order by` e `group by` são opcionais.

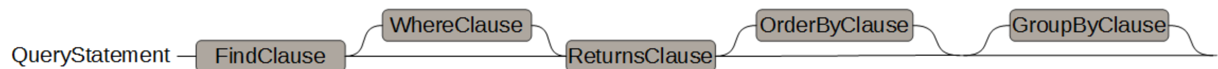


Figura 3.2 – Grafo de sintaxe das cláusulas principais da especificação original de AQL

A Figura 3.3 apresenta o grafo do elemento `QueryStatment` da extensão de AQL. Nele foram adicionadas as cláusulas de inserção (`insert`), atualização (`update`) e remoção (`delete`) de elementos. Para as cláusulas de atualização (`update`) e remoção (`delete`) a cláusula `where` é opcional.

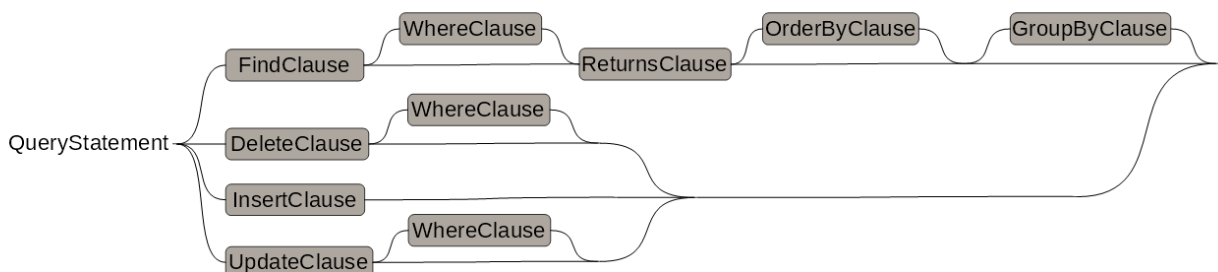


Figura 3.3 – Grafo de sintaxe das cláusulas principais da extensão de AQL

Para cada sentença AQL apenas uma cláusula de maior ordem é permitida, ou seja, executa-se uma cláusula de busca (`find`), remoção (`delete`), inserção (`insert`) ou atualização (`update`). Estas validações são gerenciadas pelo Xtext através dos analisadores léxicos e sintáticos gerados a partir da gramática EBNF.

Para as cláusulas de manipulação de dados, apenas um objeto de alta ordem é permitido. Por exemplo, é permitido remover os elementos que representam o condicional `if`, mas não é permitido remover, numa mesma consulta, os elementos que representam condicionais `if` e laços de repetição `for each`. Apesar de permitir a manipulação dos elementos, a atual implementação não realiza a validação dos códigos adicionados, remo-

vidos ou atualizados. As Seções 3.2.2, 3.2.3 e 3.2.4 mostram consultas válidas utilizando cláusulas `delete`, `insert` e `update`, respectivamente.

3.2.1 Adicionando cláusulas para manipulação de dados

Para adicionar as cláusulas para manipulação de dados foi necessário modificar a gramática AQL e adicionar novas regras de transformação e de emissão no compilador AQL.

A Listagem 3.8 mostra o elemento `QueryStatement` da gramática AQL. Ele representa o início das instruções de consulta em AQL, enquanto o elemento `Query` representa o início da AST. Dessa forma, as cláusula de manipulação, `delete` (`DeleteClause`), `insert` (`InsertClause`) e `update` (`UpdateClause`), foram adicionadas ao elemento `QueryStatement`, como mostrado na Listagem 3.8 linhas 3, 4 e 5. Para os elementos `delete` e `update` a cláusula `where` é opcional.

```
1  QueryStatement:
2      (find=FindClause) (where=WhereClause)? (return=ReturnsClause) (orderby=
           OrderByClause)? (groupby=GroupByClause)?
3      | (delete=DeleteClause) (where=WhereClause)?
4      | (insert=InsertClause)
5      | (update=UpdateClause) (where=WhereClause)?
6  ;
```

Listagem 3.8: Elemento `QueryStatment` da gramática AQL

O compilador AQL utiliza dois modelos Ecore[3]: um deles é gerenciado pelo Xtext, a partir da gramática AQL, e o outro é um modelo HQL implementado no *Eclipse Modeling Framework* (GRONBACK, 2009). Dessa forma, para adicionar as novas cláusulas, foram adicionados novos elementos no modelo HQL, os quais são utilizados para realizar as transformações uniformes.

Observa-se na Figura 3.4 os elementos `DeleteClause`, `InsertClause`, `UpdateClause` e `DMLObject` que foram inseridos no modelo HQL. Eles foram referenciados no elemento `QueryStatement`, o qual possui as principais cláusulas da consulta.

Depois de modificar o modelo HQL, as transformações entre os modelos foram implementadas. A classe `HqlTransformationPass1Impl` (Anexo D) é responsável por realizar as transformações uniformes (1 para 1) e criar a instância da AST HQL. As cláusulas de manipulação possuem equivalência semântica direta no modelo destino HQL e foram resolvidas na primeira fase de transformação. Dessa forma, o método `compile(br.ufsm.aql.Query aqlQuery)` foi modificado para realizar a criação dos elementos HQL das consultas de manipulação. As seções seguintes mostram a definição e a implementação das

[3]http://www.eclipse.org/modeling/emf/

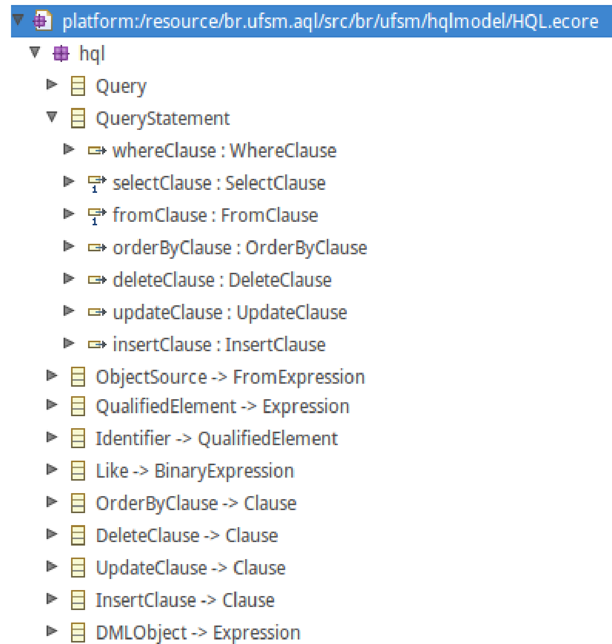


Figura 3.4 – Modelo HQL implementado no *Eclipse Modeling Framework*

transformações e emissões das cláusulas de manipulação.

3.2.2 A cláusula `delete`

A cláusula `delete` é utilizada para remover elementos em programas, como por exemplo, classes, métodos, laços de repetição e condicionais. Sua estrutura assemelha-se às cláusulas `delete` encontradas nas principais linguagens de consulta tais como HQL (HIBERNATE, 2017), XQuery (WALMSLEY, 2007) e SQL (SQL, 2017).

A Figura 3.5 mostra o grafo de sintaxe da cláusula `delete`. A cláusula inicia utilizando a palavra-chave `delete`, a qual é responsável pela sua identificação. Posteriormente, o elemento de alta ordem, que será excluído, deve ser informado e associado a um pseudônimo (*alias*). Para isso utiliza-se a sub-regra `BindingObject`, a qual permite relacionar um tipo de objeto (`ObjectType`) a um identificador (`ID`). Este poderá ser utilizado na cláusula `where`. Apenas um tipo de elemento pode ser excluído e o uso do pseudônimo é obrigatório. Por fim, a cláusula `where` pode ser utilizada para filtrar os elementos que devem ser removidos. Ela é opcional, enquanto os outros elementos são obrigatórios.

Os exemplos da Listagem 3.9 ilustram códigos válidos em AQL que utilizam a cláusula `delete`. A primeira cláusula (linha 1) mostra a exclusão de todas as classes, associadas a um pseudônimo `c`, contidas no metamodelo, as quais contenham o nome igual a `'Foo'`. A segunda cláusula (linha 2) mostra a exclusão de todos os elementos que representam expressões `if`, associadas a um pseudônimo `i`, os quais possuem identificador igual a 12. A terceira cláusula (linha 3) mostra a exclusão de todos os métodos, associados

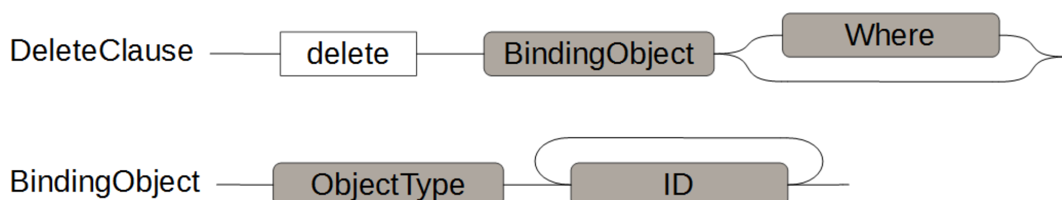


Figura 3.5 – Grafo de sintaxe da cláusula `delete`

a um pseudônimo `m`, que possuem o nome igual a ‘`getName`’ e estão contidos em uma classe que possui o nome igual a ‘`Client`’.

```
1  delete class c where c.name = 'Foo'
2  delete if i where i.id = 12
3  delete method m where m.name = 'getName' and m.owner.metaName like 'Class' and m.
       owner.name = 'Client'
```

Listagem 3.9: Exemplos de cláslulas `delete` em AQL

3.2.2.1 *Implementando a cláusula `delete`*

Para implementar a cláusula `delete` a regra `DeleteClause` foi adicionada à gramática no Xtext. Observa-se tal regra na linha 1 da Listagem 3.10. Ela descreve a palavra reservada `delete` e um elemento retornado da sub-regra `BindingObject`. A regra `BindingObject` utiliza a sub-regra `ObjectType` seguida de identificadores, como detalhado na Seção 3.2.2. Ela possui as palavras reservadas que indicam os elementos de alta ordem da consulta, como mostra a Listagem 3.4.

```
1  DeleteClause:
2      clause='delete' bindingObject+=BindingObject
3  ;
```

Listagem 3.10: Definição da regra para a cláusula `delete` em AQL

Após a adição da regra na gramática, a transformação do modelo AQL para HQL foi implementada na classe `HqlTransformationPass1Impl`, como mostra a Listagem 3.11. Inicialmente, na linha 2, verifica-se a existência da cláusula `from`, caso não exista ela será instanciada. A cláusula `from` é responsável por instanciar os elementos de alta ordem na consulta. Depois disso, na linha 4, o elemento `delete` é criado a partir do modelo da HQL. Por fim, na linha 6, o método `compile` é chamado para resolver as transformações da regra `BindingObject`.

```
1  def dispatch void compile(br.ufsm.aql.DeleteClause clause) {
2      if (hqlASTRoot.queryStatement.getFromClause == null)
```

```
3         hqlASTRoot.queryStatement.setFromClause(createFromClause)
4     hqlASTRoot.queryStatement.setDeleteClause(createDeleteClause)
5     currentClause = hqlASTRoot.queryStatement.deleteClause
6     clause.bindingObject.compile
7 }
```

Listagem 3.11: Transformação do modelo AQL para HQL da cláusula `delete`

A emissão da cláusula `Delete` é o último processo a ser realizado pelo compilador. O método `compile(hql.DeleteClause delete, hql.QueryStatement query)` da classe `AQLGenerator` é responsável pela emissão do modelo HQL, que foi gerado pelas regras transformações AQL, para a linguagem alvo. Observa-se na Listagem 3.12 a implementação deste método. Para cada cláusula `delete` em AQL, duas cláusulas HQL são geradas. Uma delas realiza a busca pelos elementos que serão removidos e retorna seus identificadores (linha 7) e a segunda consulta realiza a remoção dos elementos do metamodelo (linhas 3 a 8). Encontram-se no Anexo D as classes `AQLGenerator` e `HqlTransformationPass1Impl`.

```
1 def dispatch compile(DeleteClause delete, QueryStatement query) ''
2     «val hql.ObjectSource elementDelete = toHqlObjectSource(query)»
3     updateHQL("«delete.clause.toUpperCase»
4         FROM «elementDelete.className» «elementDelete.alias»
5         WHERE «elementDelete.alias».id IN (
6             "+getAllIds(session.createQuery("
7         SELECT «elementDelete.alias» «dmlSubQuery(elementDelete, query)»").list()
8           )+")");
9 ''
```

Listagem 3.12: Regra de emissão da cláusla `delete`

A Listagem 3.13 mostra uma instrução `delete` em AQL. Ela remove todas as classes do metamodelo que possuem o nome igual a `'Foo'`.

```
1 delete class c where c.name = 'Foo'
```

Listagem 3.13: Exemplo de consulta de remoção em AQL

A Listagem 3.14 mostra as duas consultas geradas em HQL para realizar a remoção dos elementos. Na linha 2 é executada a consulta que retorna a lista de objetos a serem removidos. O método `getAllIds` foi implementado para retornar todos os identificadores da consulta. Por fim, na linha 1, a consulta `delete` é executada.

```
1 updateHQL("DELETE FROM AOJClass c WHERE c.id IN ("+ getAllIds(
2     session.createQuery("SELECT c FROM AOJClass WHERE c.name = 'Foo'" ).list()
3 )+")");
```

Listagem 3.14: Transformação da consulta AQL da Listagem 3.13 para a linguagem alvo

3.2.3 A cláusula insert

A cláusula insert permite a inserção de novos elementos em programas. Ela assemelha-se às cláusulas de inserção encontradas nas linguagens SQL (SQL, 2017) e HQL (HIBERNATE, 2017), com algumas modificações. Inicialmente a cláusula de inserção foi baseada na cláusula insert da HQL, a linguagem de consulta do Hibernate. Porém, esta cláusula possui limitações, tais como não dar suporte à inserção de novos elementos se não tiver como base outro elemento. A Listagem 3.15 mostra um exemplo de inserção utilizando HQL. Nela, um objeto DelinquentAccount é inserido com base no objeto Customer.

```
1  insert into DelinquentAccount (id, name) select c.id, c.name from Customer c where
       c.id = 10;
```

Listagem 3.15: Exemplo inserção utilizando HQL (HIBERNATE, 2017)

A fim de não herdar essas limitações para linguagem AQL, foi utilizada como base a cláusula de inserção de SQL. A Figura 3.6 mostra o grafo de sintaxe da cláusula insert. Ele é composto (i) pela palavra-chave insert, (ii) pelo tipo de elemento de maior ordem que será inserido, (iii) por uma sequência de expressões que devem estar entre parênteses, separadas por vírgula e representar os atributos das classes implementados no metamodelo e, por fim, (iv) pelos valores que serão atribuídos aos atributos do objeto inserido.

Para informar tais valores, utiliza-se o elemento values. Observa-se a sintaxe deste elemento na Figura 3.6. Ele inicia pela palavra-chave values, seguida por uma sequência de expressões (Expression) ou referências a objetos do metamodelo (DMLObject). Os elementos desta sequência deve estar entre parênteses, separados por vírgula e representar os valores a serem atribuídos nos campos informados na cláusula insert. Tais valores podem ser do tipo String, ponto flutuante, inteiro, boleano ou a referência de um objeto do metamodelo.

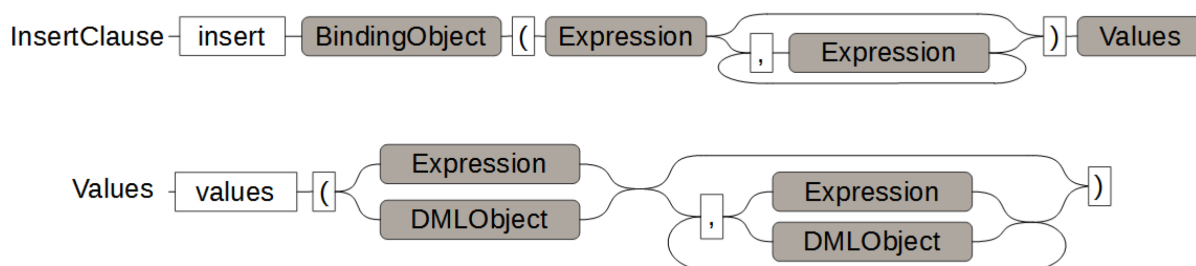


Figura 3.6 – Grafo de sintaxe da cláusula insert

A atual implementação da linguagem AQL não fornece suporte a subconsultas. Este recurso pode facilitar o uso das cláusulas de inserção e atualização. Na sua atual implementação, um elemento somente é referenciado através do seu identificador. No lugar

deste identificador, uma busca poderia ser realizada, facilitando a referência do elemento que será associado. Por exemplo, a Listagem 3.16 mostra a inserção do método `criarEmprestimo` que possui a assinatura (`signature`) igual a `'public void criarEmprestimo(double valor, Date vencimento)'` (linha 2) e como proprietário (`onwed`) a classe `'Cliente'`, a qual é selecionada através de uma subconsulta (linha 3).

```
1  insert method d (signature, onwed)
2      values ('public void criarEmprestimo(double valor, Date vencimento)',
3      find class c where c.name = 'Cliente' return c)
```

Listagem 3.16: Exemplos de cláusula `insert` utilizando subconsultas

A Listagem 3.17 mostra exemplos de consultas válidas em AQL utilizando a cláusula `insert`. A primeira cláusula (linha 1) mostra a inserção de uma expressão lambda no código. Para esta expressão foram informados, respectivamente, o corpo da expressão (`codeBody`), o elemento pai (`onwed`), que neste caso é um método que possui o identificador igual a 28, e a ordem deste elemento no método (`codeOrder`). A segunda cláusula (linha 2), mostra a inserção de um método, para ele foram informados, respectivamente, a sua assinatura (`signature`) e seu objeto pai (`onwed`), que neste caso é uma classe que possui o identificador igual a 5.

```
1  insert expression e (codeBody, onwed, codeOrder) values ('cliente.stream().filter(
       cliente -> true).forEach(cliente -> System.out.println(cliente.getName()));',
       object method 28, 30)
2  insert method d (signature, onwed) values ('public void criaEmprestimo(double valor
       , Date vencimento)', object class 5)
```

Listagem 3.17: Exemplos de cláslulas `insert` em AQL

Para cada inserção, é obrigatório informar seu elemento pai que é representado pela propriedade `onwed` dos elementos do metamodelo. Note que, caso houvesse suporte à subconsulta, o uso de identificadores deixaria de ser necessário.

Quando os tipos primitivos de dados são utilizados, como inteiro, texto e ponto flutuante, deve-se apenas informar os valores para as respectivas propriedades. Porém, quando um objeto é utilizado, precisa-se informar sua referência. Como a linguagem AQL ainda não fornece recursos para realizar subconsultas, foi desenvolvido o elemento *DMLObject* para auxiliar a busca por objetos no metamodelo, os quais serão referenciados nas consultas de manipulação de dados. A Figura 3.7 mostra o grafo de sintaxe do elemento *DMLObject*. Ele inicia pela palavra-chave `object`, seguido pela opção de adicionar ou remover os objetos, caso eles pertençam a uma lista (opcional), o tipo de objeto a ser buscado e o identificador do objeto.

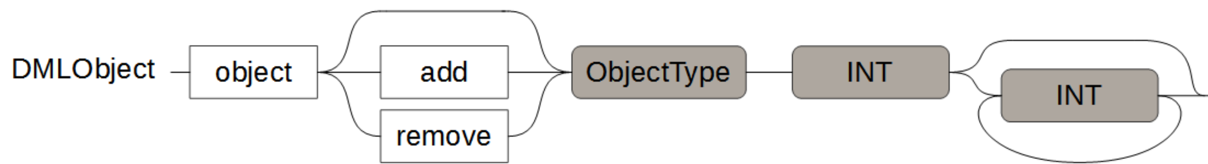


Figura 3.7 – Grafo de sintaxe para referenciar um ou mais objetos

3.2.3.1 *Implementando a cláusula `insert`*

Para implementar a inserção de elementos a regra `InsertClause` foi adicionada à gramática AQL. A Listagem 3.18 mostra a regra da cláusula de inserção. Tal regra descreve a palavra reservada `insert`, um objeto da sub-regra `BindingObject`, uma lista de objetos da sub-regra `Expression` e um objeto da sub-regra `Values` (linha 2). A regra `Values` (linha 3) representa os valores a serem atribuídos aos campos do elemento que será inserido no metamodelo. Tal regra descreve a palavra reservada `values` e uma lista de objetos retornados da sub-regra `Expression` ou `DMLObject`. A sub-regra `Expression` é utilizada para atribuir tipos primitivos, enquanto a sub-regra `DMLObject` é utilizada para referenciar objetos existentes no metamodelo. Para cada elemento da regra `expressions`, um respectivo valor deve ser inserido na cláusula `Values`.

```
1  InsertClause:
2      clause='insert' bindingObject+=BindingObject '(' expressions+=Expression (','
          expressions+=Expression)* ')' values=Values
3  ;
4  Values:
5      clause='values' '(' expressions+=(Expression | DMLObject) (',' expressions+=(
          Expression | DMLObject))* ')'
6  ;
```

Listagem 3.18: Regra da cláusula `insert` em AQL

A Listagem 3.19 mostra a regra de transformação desenvolvida para a cláusula `insert`, que converte a consulta de inserção AQL para HQL. Inicialmente, nas linhas 2 e 3, verifica-se a existência da cláusula `from` no modelo HQL e, caso necessário, cria-se o elemento. Posteriormente, na linha 4, o elemento `insert` do modelo HQL é criado através do método `createInsertClause`. Na linha 5, o elemento `BindingObject` é resolvido através das regras de transformações implementadas no método `clause.bindingObject.compile`. Nas linhas 6 e 7 os elementos `expressions` são resolvidas através das regras de transformações implementadas no método `expressionCompiler.compile(e)` da classe `HqlTransformationPass1Impl` e adicionadas ao elemento `expressions` da cláusula `insertClause` do modelo HQL. Por fim, na linha 8, as regras de transformação do elemento `values` são resolvidas através do método `clause.values.compile`.

```
1  def dispatch void compile(br.ufsm.aql.InsertClause clause) {
```

```
2     if (hqlASTRoot.queryStatement.getFromClause == null)
3         hqlASTRoot.queryStatement.setFromClause(createFromClause)
4     hqlASTRoot.queryStatement.setInsertClause(createInsertClause)
5     clause.bindingObject.compile
6     for (br.ufsm.aql.Expression e : clause.expressions)
7         hqlASTRoot.queryStatement.insertClause.expressions.add(expressionCompiler.
              compile(e))
8     clause.values.compile
9 }
```

Listagem 3.19: Transformação do modelo AQL para HQL da cláusula `insert`

A emissão da cláusula é o último passo a ser realizado. Para isso foi implementado o método `compile(InsertClause insert, QueryStatement query)` na classe responsável pela emissão do modelo HQL gerado para a linguagem alvo, a `AQLGenerator` (Anexo D). A Listagem 3.20 mostra o método que realiza a emissão da cláusula `insert`.

```
1 def dispatch compile(InsertClause insert, QueryStatement query) ''
2     «insert.values.searchObjects»
3     «val ObjectSource elementInsert = toHqlObjectSource(query)»
4     «var onwed = insert.findOnwed»
5     «elementInsert.className» element = new «elementInsert.className»(null, «
          getObjectNames(onwed)»);
6     «setAttributes(insert)»
7     element.loadByAQL();
8     session.save(element);
9 ''
```

Listagem 3.20: Regra de emissão da cláusula `insert`

A emissão da cláusula utiliza instruções HQL para buscar os objetos de referência persistidos no metamodelo e instruções Java e Hibernate para inserir o elemento. A primeira etapa do processo de emissão é criar uma lista dos objetos de referência utilizados na consulta. Esta etapa é realizada através do método `searchObjects` (linha 2). Posteriormente, na linha 3, o elemento a ser inserido é localizado na consulta e atribuído à variável `elementInsert` a fim de facilitar sua manipulação. Na linha 4, o elemento proprietário é buscado na lista de objetos criada no método `searchObjects` e referenciado na variável `owned`. Na linha 5, é realizada a instância do objeto a ser inserido passando como construtor o proprietário. Posteriormente, as propriedades do objeto são preenchidas de acordo com os valores informados na consulta, linha 6. Por fim, o método `loadByAQL` é chamado (linha 7) e o objeto é persistido (linha 8).

A Listagem 3.21 mostra uma instrução `insert` em AQL que adiciona uma expressão lambda no programa. O proprietário desta expressão é o método que possui identificador igual a 28. O corpo da expressão será igual a `cliente.stream().filter(cliente -> true).forEach(cliente -> System.out.println(cliente.getName()))`. A ordem que

o elemento será inserido dentro do método é igual a 30.

```
1 insert expression (owner, codeBody, codeOrder) values (object method 28, "cliente.
      stream().filter(cliente -> true).forEach(cliente -> System.out.println(cliente.
      getName()));", 30)
```

Listagem 3.21: Exemplo de consulta de inserção em AQL

A Listagem 3.22 mostra as instruções geradas para realizar a inserção do elemento no metamodelo. Nesta cláusula, apenas um objeto é referenciado e buscado do metamodelo, que será o proprietário do elemento a ser inserido (linha 1). Posteriormente, o objeto é instanciado (linha 2) e suas propriedades são atribuídas de acordo com as informações da consulta (linhas 3 e 4). Depois disso, é feita a referência do objeto proprietário (linhas 5 e 6), as regras de inserção são carregadas (linha 7) e o objeto é persistido no modelo de dados (linha 8).

```
1 AOJProgramElement obj0 = (AOJProgramElement) session.createQuery("FROM
      AOJMethodDeclaration m WHERE id in (28)").list().iterator().next();
2 AOJExpressionStatement element = new AOJExpressionStatement(null, obj0, 1);
3 element.setCodeBody("cliente.stream().filter(cliente -> true).forEach(cliente ->
      System.out.println(cliente.getName()));");
4 element.setCodeOrder(30);
5 AOJMethodDeclaration owner0 = (AOJMethodDeclaration) obj0;
6 owner0.addStatements(element);
7 element.loadByAQL();
8 session.save(element);
```

Listagem 3.22: Transformação da consulta AQL da Listagem 3.21 para a linguagem alvo

3.2.4 A cláusula `update`

A cláusula `update` permite modificar os atributos e as referências dos elementos contidos no metamodelo. Sua estrutura foi projetada utilizando a estrutura da cláusula `insert`, mostrada na Seção 3.2.3. Em HQL, a cláusula `update` também possui limitações, tais como não dar suporte a associações explicitas (HIBERNATE, 2017).

A Figura 3.8 mostra o grafo de sintaxe da cláusula `update`. Ele é composto (i) pela palavra-chave `update`, (ii) seguida pelo objeto de maior ordem que será atualizado, (iii) uma sequência de expressões que referenciam as propriedades do objeto, elas devem estar entre parenteses e separadas por vírgula, (iv) seguido pelo elemento `values`, o qual deverá informar os valores que serão atualizados, e, por fim, (v) a cláusula `where` poderá ser utilizada para filtrar os elementos. A cláusula `where` é opcional, enquanto os outros elementos são obrigatórios.

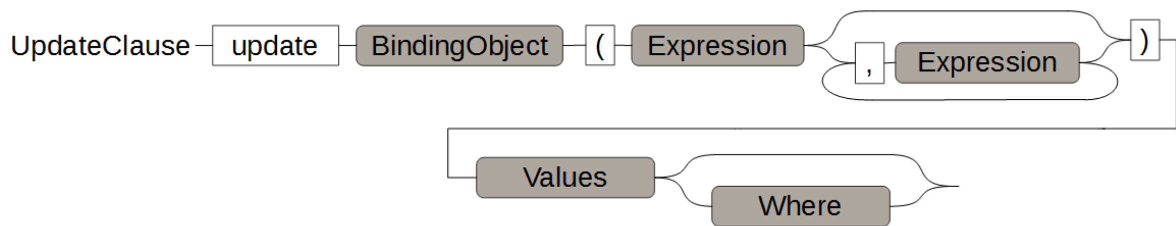


Figura 3.8 – Grafo de sintaxe da cláusula `update`

A Listagem 3.23 mostra exemplos válidos de código AQL que utilizam a cláusula `update`. A primeira cláusula (linha 1) mostra a alteração do nome da classe `"Felino"` para `"Gato"`. A segunda cláusula (linha 2) mostra a alteração do nome do método `"novoMetodo"` para `"getFelino"`.

```
1  update class c (name) values ('Gato') where c.name = 'Felino'
2  update method m (name) values ('getFelino') where m.name = 'novoMetodo'
```

Listagem 3.23: Exemplos de cláslulas `update` em AQL

3.2.4.1 *Implementando a cláusula `update`*

A regra da cláusula `update`, denominada `UpdateClause` (linha 1), é idêntica à regra `InsertClause`, tendo como diferença a palavra reservada `update` e a possibilidade de utilizar a cláusula `where`. A Listagem 3.25 mostra a regra da cláusula `update`.

```
1  UpdateClause :
2      clause='update' bindingObject+=BindingObject '(' expressions+=Expression (','
           expressions+=Expression)* ')' values=Values
3  ;
```

Listagem 3.24: Regra da cláusula `update` em AQL

A Listagem 3.25 mostra a implementação da emissão da cláusula `update`. Ela é semelhante a cláusula `InsertClause`. Sua diferença está na linha 4, a qual realiza a busca pelos elementos que serão modificados e nas linhas 5 a 8, que realizam a modificação dos valores dos atributos informados na consulta.

```
1  def dispatch compile(UpdateClause update, QueryStatement query) ''
2      «val ObjectSource elementUpdate = toHqlObjectSource(query)»
3      «update.values.searchObjects»
4      List<«elementUpdate.className»> elements = session.createQuery(''«dmlSubQuery(
           elementUpdate, query).toString»'').list();
5      for («elementUpdate.className» element : elements) {
6          «setUpdateElement(update, query)»
7          session.update(element);
```

```
8     }
9 ''
```

Listagem 3.25: Regra de emissão da cláusula `update`

A Listagem 3.26 mostra uma instrução de atualização em AQL, que atualizará o atributo `metaName` de todos os objetos do tipo `Expression` cujo o atributo `metaName` é igual `'Expression'` para `'Lambda Expression'`.

```
1 update expression e (metaName) values ('Lambda Expression') where e.metaName = '
      Expression'
```

Listagem 3.26: Exemplo de consulta de atualização em AQL

A Listagem 3.27 mostra as instruções geradas, a partir da Listagem 3.26, para realizar as atualizações nos elementos do metamodelo.

```
1 List<AOJExpressionStatement> elements = session.createQuery("SELECT e FROM
      AOJExpressionStatement e WHERE e.metaName = 'Expression'").list();
2 for (AOJExpressionStatement element: elements) {
3     element.setMetaName("Lambda Expression");
4     session.update(element);
5 }
```

Listagem 3.27: Transformação da consulta AQL da Listagem 3.26 para a linguagem alvo

3.3 APRIMORANDO A CLÁUSULA `ORDER BY`

Na original versão da cláusula `order by` em AQL, somente era possível indicar uma ordem para todos os elementos declarados na cláusula `order by`, ou seja, os elementos são ordenados em ordem ascendente ou descendente. Esta cláusula foi modificada a fim de possibilitar a ordenação individual de cada elemento da consulta. A Figura 3.8 mostra o grafo modificado da sintaxe da cláusula `order by`.

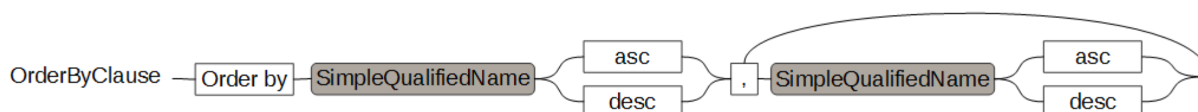


Figura 3.9 – Grafo de sintaxe da cláusula `order by`

A Listagem 3.28 mostra um exemplo utilizando as modificações realizadas na cláusula. Esta consulta seleciona todas as classes e as ordena pelo seu nome (name) em ordem alfabética e pelo seu número de linhas (nol) em ordem descendente.

```
1  find class c returns c order by c.name asc, c.nol desc
```

Listagem 3.28: Exemplo de consulta `order by` em AQL

3.3.1 Implementado melhorias na Cláusula `Order By`

Para permitir a ordenação independente para cada elemento da cláusula `OrderBy-Clause` sua regra na gramática AQL foi modificada, como mostra a Listagem 3.29. Para cada elemento, é permitido utilizar uma expressão e indicar a ordem em que ele será ordenado.

```
1  OrderByClause :
2      clause='order by' expressions+=OrderByExpression (',' expressions+=
          OrderByExpression)* ('having' havingExpression=Expression)?
3  ;
4  OrderByExpression :
5      expr=SimpleQualifiedName option=('asc' | 'desc')?
6  ;
```

Listagem 3.29: Modificação realizada na cláusula `order by` em AQL

A Listagem 3.30 mostra o uso da cláusula `order by`, em AQL, a qual ordena ascendentemente pelo atributo `metaName` e descendente pelo atributo `codeOrder`. A Listagem 3.31 mostra a cláusula HQL gerada pelo compilador AQL a partir da cláusula da Listagem 3.30.

```
1  find foreach f return f.metaName, f.codeOrder
2      order by f.metaName asc, f.codeOrder desc
```

Listagem 3.30: Exemplo de consulta AQL utilizando as melhorias da cláusula `order by`

```
1  SELECT f.metaName, f.codeOrder FROM AOJEnhancedForStatement f
2      ORDER BY f.metaName asc, f.codeOrder desc
```

Listagem 3.31: Transformação da consulta AQL da Listagem 3.30 em HQL

3.4 MELHORANDO VALIDAÇÕES

A implementação de referência da linguagem AQL utiliza a metalinguagem fornecida pelo Xtext, baseada em regras EBNF, para definir sua gramática. O Xtext gera os analisadores que realizam as análises léxicas e sintáticas a partir da gramática EBNF

formalmente descrita. Posteriormente, se nenhuma inconsistência for identificada, é realizada a análise semântica. Esta análise é parte da estrutura de validações customizadas do Xtext. Por padrão, o Xtext instancia uma classe que estende a classe `AbstractDeclarativeValidator` do pacote `org.eclipse.xtext.validation`, a qual permite implementar validações customizadas.

A implementação original da linguagem AQL possuía algumas validações porém elas não estavam sendo executadas ou mostravam apenas exceções no console da IDE, dificultando a interação do usuário com a linguagem. Tais validações foram corrigidas na extensão da linguagem.

As cláusulas `delete` e `update`, adicionadas no elemento `QueryStatement` (Seção 3.1), fazem o uso da cláusula `where` quando houver a necessidade de filtrar os resultados da consulta. Desta forma, os elementos não declarados na cláusula principal, `delete` e `update`, não podem ser referenciados na cláusula `where`. Esta validação foi implementada utilizando as validações customizadas.

A Listagem 3.32 mostra dois exemplos de código AQL inválidos. A linha 1 mostra uma instrução de remoção de uma classe que está identificada pelo pseudônimo `c` e está sendo referenciada na cláusula `where` utilizando o pseudônimo `d`. A linha 2 mostra uma consulta que atualiza o nome da classe `Gato` para `Felino`, porém, a classe é referenciada através do pseudônimo `c` na cláusula `update` e é utilizada através do pseudônimo `d` na cláusula `where`.

```
1  delete class c where d.id = 10
2  update class c (name) values ('Gato') where d.name = 'Felino'
```

Listagem 3.32: Cláusulas inválidas em AQL

As consultas inválidas são tratadas pela extensão da implementação de referência da linguagem.

3.4.1 Implementando Validações

Algumas validações customizadas foram implementadas no compilador da linguagem AQL. Elas são executadas após realizar as análises léxica e sintática da linguagem, as quais são geradas pelo Xtext com base nas regras da gramática EBNF. O Xtext instancia uma classe que estende a classe `AbstractDeclarativeValidator`[4]. Ela é responsável por executar as validações. No compilador AQL é denominada `AQLValidator` (Anexo D).

A Listagem 3.33 mostra algumas das validações implementadas. A primeira validação, linhas 2 a 7, verifica se existe pelo menos uma das cláusulas principais da consulta.

[4]http://download.eclipse.org/modeling/tmf/xtext/javadoc/2.3/org/eclipse/xtext/validation/AbstractDeclarativeValidator.html

São elas: *find*, *update*, *insert* ou *delete*. E a segunda validação, linhas 8 a 16, verifica a existência de pseudônimos duplicados. Nota-se que todas as validações devem utilizar a notação `@Check` do pacote `org.eclipse.xtext.validation.Check`.

```
1  class AQLValidator extends AbstractAQLValidator {
2      @Check
3      def checkEmptyClauses(QueryStatement query){
4          if (query.find == null && query.update == null && query.insert == null &&
               query.delete == null)
5              error ('You must have a main clause.', query.eContainer(), query.
                   eContainingFeature(), -1);
6          return;
7      }
8      @Check
9      def checkDuplicatedAlias (FindClause find) {
10         for (BindingObject bind : find.getBindingObject()) {
11             Set<String> set = new HashSet<String>(bind.getAlias());
12             if (set.size() < bind.getAlias().size())
13                 error ("Duplicated binding alias definition on find clause", query.
                       eContainer(), query.eContainingFeature(), -1);
14             return;
15         }
16     }
17     (...)
18 }
```

Listagem 3.33: Validaçoes customizadas em AQL

A Figura 3.10 mostra como os erros são exibidos na extensão da linguagem AQL.

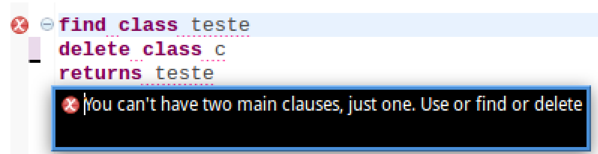


Figura 3.10 – Informando um erro na consulta para o usuário

3.5 MODIFICAÇÕES REALIZADAS NO *FRAMEWORK* AOPJUNGLE

O *framework* AOPJungle é responsável por fornecer informações sobre o código fonte de sistemas orientados a objetos e aspectos (FAVERI, 2013). Para isso, ele utiliza as APIs do Eclipse que fornecem a lista de projetos do *workspace*, e a AJDT, a qual fornece informações sobre as unidades de compilação das ASTs dos projetos. As informações coletadas são armazenadas e organizadas em um metamodelo orientado a objetos, o qual

é utilizado para realizar as instruções geradas pelo compilador AQL. Como a linguagem foi estendida, possibilitando a manipulação de outras unidades de compilação, houve a necessidade de modificar o metamodelo, adicionando a ele novas informações sobre os sistemas.

Para instanciar o metamodelo, a classe `AOJungleVisitor`, do *Framework* AOP-Jungle, estende a classe `AjASTVisitor` do AJDT e implementa o padrão *Visitor*, como mostrado na Seção 2.5.1. Através desta implementação, é possível percorrer as unidades de compilação dos sistemas de software analisados e extrair as informações necessárias de cada nó. Cada método `visit` permite o acesso a um nó específico da AST. Para cada nova classe adicionada no metamodelo foi sobrescrito o método *visit* correspondente. A Listagem 3.34, por exemplo, mostra o método que visita os nós `IfStatement` das ASTs. As classes do AOPJungle encontram-se no Anexo E deste trabalho.

```
1  @Override
2  public boolean visit(IfStatement node) {
3      AOJIfStatement aojNode = new AOJIfStatement(node, (AOJProgramElement)
          getLastMemberFromStack());
4      addStatments(aojNode);
5      getElementStack().push(aojNode);
6      return super.visit(node);
7  }
```

Listagem 3.34: Sobrescrita do método visit da unicade de compilaçao IfStatement

A Figura 3.11 mostra a estrutura base do metamodelo. Tais classes herdam da classe abstrata `AOJMember`, a qual herda da classe `ASTElement`, que, por sua vez herda da classe `AOJProgramElement`.

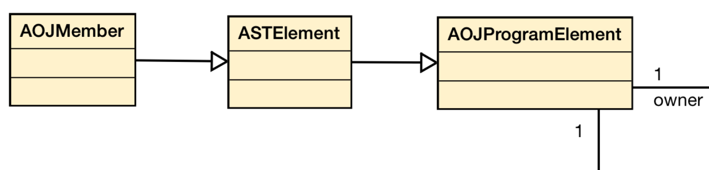


Figura 3.11 – Estrutura básica do metamodelo do AOPJungle (TEIXEIRA JÚNIOR, 2014)

Até o momento não havia interesse em adicionar novos elementos no modelo e o atributo identificador era utilizado para manter a ordem dos elementos. Como os identificadores são gerenciados pelo banco de dados, não há como garantir a ordem correta dos elementos inseridos através dos identificadores. Para isso, o atributo `codeOrder` foi adicionado na classe no `AOJMember` do AOPJungle.

A primeira versão do metamodelo não possuía classes que representassem instruções de código contidas em métodos e construtores, pois não eram relevantes para a pesquisa original (FAVERI, 2013). Uma segunda versão deste metamodelo foi desenvolvida para dar suporte a expressões lambda (TEIXEIRA JÚNIOR, 2014). Esta versão

adicionou as classes `AOJEnhancedForStatement` e `AOJIfStatement` que representam, respectivamente, laços de repetição `for each` e estruturas condicionais. Porém, algumas modificações precisaram ser feitas para adaptar a extensão do *framework* (TEIXEIRA JÚNIOR, 2014), pois, para sua pesquisa, não houve a necessidade de utilizar o Hibernate para realizar a persistência dos dados.

Primeiramente, adicionamos as classes `AOJSwitchStatement`, `AOJWhileStatment` e `AOJEpressionStatment`, as quais representam, respectivamente, unidades de código do tipo `switch`, `while` e `expression`. A Figura 3.12 mostra a estrutura proposta para o armazenamento das informações no metamodelo. Todo elemento `Statement` da AST herda da super classe `AOJStatement`. Ela armazena informações comuns entre tais elementos, como a lista dos elementos filhos.

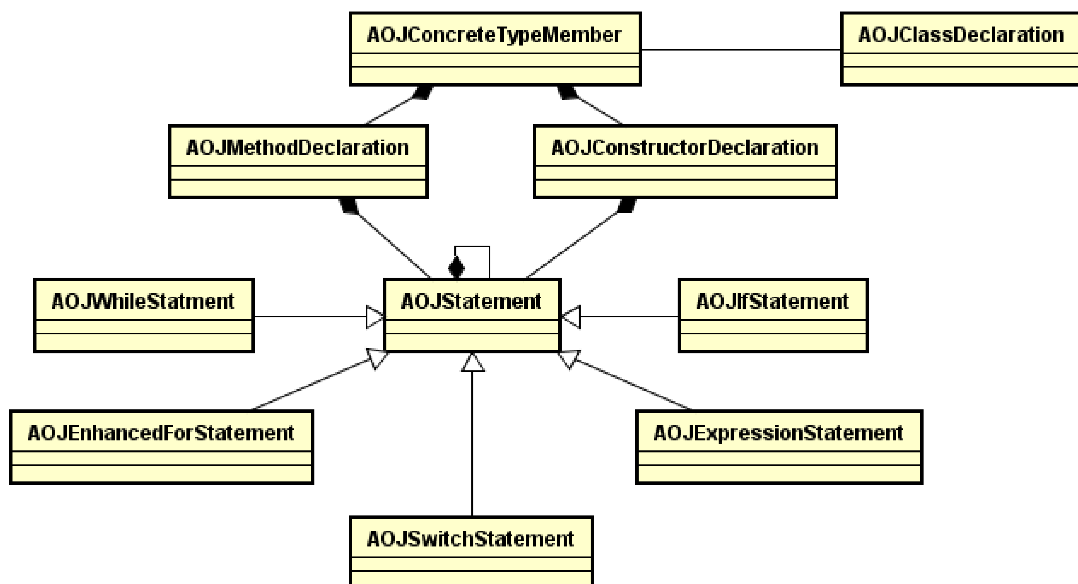


Figura 3.12 – Novos elementos adicionados no metamodelo do AOPJungle

As classes `AOJMethodDeclaration` e `AOJConstructDeclaration`, contidas no pacote `br.ufsm.aopjungle.metamodel.commons`, foram modificadas para receber as referências dos `statements`, elementos que representam estrutura de código contida dentro de métodos e construtores.

Para cada subclasse da classe `AOJStatmen` foram adicionadas informações referentes à sua unidade de compilação. Por exemplo, a classe `AOJEnhancedForStatement` não possuía informações sobre a expressão e o parâmetro utilizado.

Tais informações foram coletadas da AST e adicionadas no elemento. Durante o desenvolvimento, verificou-se que o analisador implementado pelo AJDT para AspectJ[5] possui limitações quanto à resolução de ligações. Na documentação oficial do AspectJ, em sua versão 1.7, tal suporte ainda é uma questão pendente. Algumas destas informações foram mapeadas para o metamodelo utilizando como referência os elementos da AST,

[5] `org.aspectj.org.eclipse.jdt.core.dom.ASTParser`

como por exemplo o tipo do parâmetro utilizado no `EnhancedFor Statement`.

Por fim, uma classe para a geração de código, a partir do metamodelo, foi implementada no `framework` AOPJungle. A partir de um elemento selecionado do metamodelo é possível gerar seu respectivo código fonte. Esta classe encontra-se no Anexo E deste trabalho.

3.6 CONSIDERAÇÕES FINAIS

Este capítulo apresentou os novos recursos adicionados à linguagem AQL e os detalhes das modificações realizadas na implementação de referência da linguagem AQL e no *framework* AOPJungle. Tais modificações forneceram os recursos necessários para realizar consultas utilizando granularidade mais fina e realizar modificações no metamodelo através das cláusulas DML, as quais permitem inserir, alterar e excluir elementos dos programas. Para isso, foram inseridas novas regras na gramática do compilador AQL, foram implementadas as respectivas regras de transformação, foi modificado o metamodelo de referência HQL e foi adaptado o módulo de emissão do compilador para emitir as cláusulas utilizando instruções HQL, Hibernate e Java. Este capítulo também apresentou as melhorias realizadas nas cláusulas existentes da linguagem AQL e as validações que foram adicionadas na implementação de referência. No *framework* AOPJungle foram adicionados os novos elementos no metamodelo, foram implementadas as funções auxiliares para executar as instruções de manipulação e foi implementado um gerador de código que reproduz a estrutura do metamodelo para código Java. O próximo capítulo apresenta um estudo de caso utilizando as novas funcionalidades da linguagem AQL para identificar oportunidades de refatoração em código OO e a utilização das cláusulas de manipulação para aplicar as refatorações no metamodelo.

4 ESTUDO DE CASO

Este capítulo apresenta um estudo de caso envolvendo o uso da extensão da linguagem AQL como apoio nas atividades de busca por oportunidades de refatoração e a aplicação da refatoração em código orientado a objetos. Para realizar as buscas e modificações nos sistemas analisados, foi utilizado o *framework* AOPJungle.

Neste estudo de caso, foram realizadas buscas por algumas refatorações relacionadas a expressões lambda definidas para Java 8 (TEIXEIRA JÚNIOR, 2014) e por uma refatoração para padrões de projeto (KERIEVSKY, 2008; PAULI, 2014).

Para cada uma das refatorações selecionadas, foram executados os passos para aplicar a refatoração na oportunidade encontrada. De forma a mostrar a aplicabilidade da abordagem foram utilizadas algumas classes exemplos. As refatorações são:

- *Convert Enhanced For to Lambda Enhanced For*.
- *Convert Collections.sort to sort*.
- *Convert Enhanced For with If to Lambda Filter*.
- *Convert Abstract Interface Method to Default Method*.
- *Chain Constructors*.

As seções seguintes (4.1 a 4.5) apresentam as instruções em AQL utilizadas para realizar as buscas pelas oportunidades de refatoração, as respectivas instruções para a aplicação da refatoração e as classes de exemplo que possuem as oportunidades de refatoração. O Anexo C contém as transformações geradas pelo compilador AQL para cada consulta.

4.1 APLICANDO *CONVERT ENHANCED FOR TO LAMBDA ENHANCED FOR*

Aconselha-se utilizar a refatoração *Convert Enhanced For to Lambda Enhanced For* (TEIXEIRA JÚNIOR, 2014) quando, ao percorrer uma coleção, o desenvolvedor deseja adicionar outros comportamentos, por exemplo, filtro (`filter`) ou uma ordenação (`sort`). Esta refatoração pode ser aplicada quando se utiliza um laço de repetição para percorrer uma coleção.

Para mostrar a aplicação desta refatoração a classe `Pessoas` foi utilizada. Ela é mostrada na Listagem 4.1. Esta classe possui dois métodos que percorrem a coleção de nomes e os mostram na saída padrão do sistema. O método `mostraNomes`, linhas 3 a 6,

utiliza o laço de repetição `for each` e o método `mostraNomesW`, linhas 7 a 13, utiliza o laço de repetição `while`.

```
1  public class Pessoas {
2      private Collection<String> nomes = new ArrayList<>();
3      public void mostraNomes() {
4          for (String nome : nomes)
5              System.out.println(nome);
6      }
7      public void mostraNomesW() {
8          Iterator<String> itNomes = nomes.iterator();
9          while (itNomes.hasNext()) {
10             String nome = itNomes.next();
11             System.out.println(nome);
12         }
13     }
14     public void inserir(String nome){
15         nomes.add(nome);
16     }
17     public void remover(String nome){
18         if (nomes.contains(nome))
19             nomes.remove(nome);
20     }
21     public void ordenaNomes() {
22         Collections.sort((List<String>) nomes, new Comparator<String>() {
23             public int compare(String p1, String p2) {
24                 return p1.compareTo(p2);
25             }
26         });
27     }
28     public void mostrarNomesIniciandoPor(String inicio) {
29         for (String nome : nomes)
30             if (nome.startsWith(inicio))
31                 System.out.println(nome);
32     }
33 }
```

Listagem 4.1: Classe `Pessoas`

Para localizar essas estruturas foram escritas duas consultas em AQL. Uma para selecionar e retornar os laços de repetição `for each` que percorrem uma coleção e outra para os laços de repetição `while`. As Listagens 4.2 e 4.3 mostram as instruções AQL utilizadas para buscar por oportunidades da refatoração *Convert Enhanced For to Lambda Enhanced For* em laços `for each` e `while`, respectivamente.

```
1  find foreach f where f.isCollection = true returns f
```

Listagem 4.2: Buscando pelos laços de repetição `for each` que percorrem uma coleção

```
1  find while w where w.isCollection = true returns w
```

Listagem 4.3: Buscando pelos laços de repetição `while` que percorrem uma coleção

Estas consultas retornam os objetos buscados de acordo com seu tipo, ou seja, a consulta da Listagem 4.2 retorna os objetos do tipo `AOJEnhancedForStatement` que representam os laços de repetição `for each` e a Listagem 4.3 retorna os objetos do tipo `AOJWhileStatement` que representam os laços de repetição `while`.

Para aplicar a refatoração *Convert Enhanced For to Lambda Enhanced For* nos elementos identificados, algumas informações são necessárias, tais como:

- O identificador do laço de repetição, para ser utilizado nas consultas de manipulação;
- O corpo do elemento que será refatorado, para verificar se é possível realizar a refatoração e como ela deve ser realizada;
- A ordem do elemento, pois a expressão lambda será inserida após o laço de repetição;
- O identificador do elemento proprietário, para associar a ele a expressão lambda inserida;
- O tipo do elemento.

A Figura 4.1 mostra as informações do elemento `for each` do método `mostraNomes` da classe `Pessoas`, o qual será refatorado. O procedimento utilizado para aplicar a refatoração no laço de repetição `while` é idêntico ao `for each`, sendo assim, apenas o `for each` será mostrado.

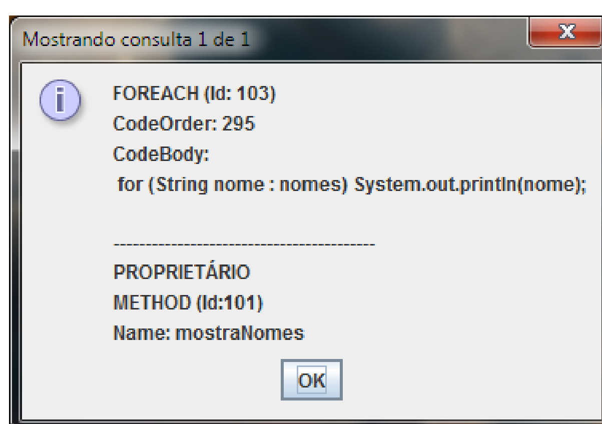


Figura 4.1 – Mostrando informações do elemento `for each`

Após localizar as oportunidades de refatoração e identificar as informações dos elementos a serem refatorados, deve-se conhecer a mecânica da refatoração. A mecânica da refatoração *Convert Enhanced For to Lambda Enhanced For* (TEIXEIRA JÚNIOR, 2014) é descrita pelos seguintes passos:

1. Substitua a construção `for each` pela chamada ao método `forEach` da coleção;
2. Implemente a interface funcional *Consumer* requerida pelo método `forEach`, movendo o conteúdo da estrutura de repetição para o método da interface *Consumer*;
3. Aplique a refatoração *Convert Functional Interface Instance to Lambda Expression* para a instância da interface *Consumer*.

Uma expressão lambda é representada no metamodelo do AOPJungle pela classe `AOJExpressionStatement` e na AQL pelo tipo `expression`. Sendo assim, deve-se adicionar um elemento `expression` após a estrutura `for each` do método `mostraNomes`. Esta expressão deve utilizar o método `forEach` da coleção e implementar a interface funcional requerida. Dessa forma, foi escrita uma consulta para inserir este elemento. Observa-se tal consulta em AQL na Listagem 4.4.

```
1  insert expression e (owner, codeOrder, codeBody) values (object method 101, 296, '
       nomes.forEach(nome -> System.out.println(nome))')
```

Listagem 4.4: Consulta `insert` para adicionar a expressão lambda no método `mostraNomes`

Nesta consulta, o elemento proprietário (método `mostraNomes`) é associado a expressão através do seu identificador (101). Posteriormente, são informadas a implementação da expressão (`'nomes.forEach(nome -> System.out.println(nome))'`) e sua ordem (296).

A atual implementação da linguagem AQL não fornece suporte a subconsultas, sendo assim, os elementos do metamodelo são referenciados através do atributo identificador. Observa-se na Listagem 4.4 que o método `mostraNomes` é referenciado através do seu identificador (101).

Observa-se na Figura 4.1 que a ordem do elemento `for each`, o qual será removido, é igual a 295. Como a expressão lambda deve ser inserida após este elemento, sua ordem deve ser maior que a ordem do elemento `for each`.

Após executar a consulta AQL, o método `mostraNomes` da classe `Pessoas` possui as duas implementações que iteram sobre a coleção `nomes`: a do laço de repetição `for each` e a da expressão lambda. Observa-se na Listagem 4.5 as duas implementações, nas linhas 3 e 4 o `for each` e na linha 5 a expressão lambda.

```
1      //...
2      public void mostraNomes(){
3          for (String nome : nomes)
4              System.out.println(nome);
5          nomes.forEach(nome -> System.out.println(nome));
6      }
7      //...
```

Listagem 4.5: Método `mostraNomes` da classe `Pessoas`

O próximo passo é remover a implementação `for each` do método `mostraNomes`. Para isso, foi escrita uma consulta de remoção em AQL. Observa-se na Listagem 4.6 que, para remover este elemento, foi informado seu tipo, `foreach`, e seu identificador, 103. Essas informações foram mostradas na Figura 4.1. As instruções geradas pelo compilador AQL foram utilizadas para remover o elemento através de uma classe de testes do AOP-Jungle.

```
1  delete foreach f where f.id = 103
```

Listagem 4.6: Consulta em AQL para remover o elemento `for each`

A Listagem 4.7 mostra a classe `Pessoas` gerada no AOPJungle após as refatorações serem aplicadas. Observa-se que os métodos `mostraNomes`, na linha 3, e `mostraNomesW`, na linha 6, implementam as expressões lambdas ao invés dos laços de repetição. Estes métodos ficaram menores e reduziram o número de linhas da classe.

```
1  public class Pessoas {
2      private Collection<String> nomes = new ArrayList<>();
3      public void mostraNomes() {
4          nomes.forEach(nome -> System.out.println(nome));
5      }
6      public void mostraNomesW() {
7          nomes.forEach(nome -> System.out.println(nome));
8      }
9      public void inserir(String nome){
10         nomes.add(nome);
11     }
12     public void remover(String nome){
13         if (nomes.contains(nome))
14             nomes.remove(nome);
15     }
16     public void ordenaNomes() {
17         Collections.sort((List<String>) nomes, new Comparator<String>() {
18             public int compare(String p1, String p2) {
19                 return p1.compareTo(p2);
20             }
21         });
22     }
23     public void mostrarNomesIniciandoPor(String inicio) {
24         for (String nome : nomes)
25             if (nome.startsWith(inicio))
26                 System.out.println(nome);
27     }
28 }
```

Listagem 4.7: Classe `Pessoas` após a execuçao das refatorações

As instruções geradas pelo compilador de AQL para as consultas desta refatoração (assim como para as demais) podem ser vistas no Anexo C.

4.2 APLICANDO *CONVERT COLLECTIONS.SORT TO SORT*

Sugere-se utilizar a refatoração *Convert Collections.sort to sort* (TEIXEIRA JÚNIOR, 2014) quando as chamadas para o método `Collections.sort` podem ser substituídas pelo método `sort` da própria coleção. Em Java 8, a interface `Collection` passou a fornecer uma nova funcionalidade que permite a ordenação da própria coleção. O processo de ordenação é feito utilizando a implementação da interface funcional `Comparator`.

A fim de mostrar a aplicação desta refatoração, a classe `Pessoas` foi utilizada. Observa-se na linha 21 da Listagem 4.1 o método `ordenaNomes` que realiza uma chamada para o método `Collections.sort`, o qual pode ser substituído. A Listagem 4.8 mostra apenas o método `ordenaNomes`.

```
1  public class Pessoas {
2      //...
3      public void ordenaNomes() {
4          Collections.sort((List<String>) nomes, new Comparator<String>() {
5              public int compare(String p1, String p2) { return p1.compareTo(p2); }
6          });
7      }//...
8  }
```

Listagem 4.8: Código exemplo para aplicar a refatoração `Convert Collections.sort to sort`

Para localizar as possíveis oportunidades da refatoração *Convert Collections.sort to sort*, foi escrita uma consulta em AQL que seleciona as expressões que chamam o método `sort` da classe `Collections`. Uma das formas de buscar por estes elementos é verificar se no corpo da expressão existem chamadas para o método `sort`. Observa-se na Listagem 4.9 a consulta escrita em AQL para buscar esses elementos.

```
1  find expression e where e.codeBody like 'Collections.sort%' returns e
```

Listagem 4.9: Busca por expressões que chamam o método `Collections.sort`

A consulta da Listagem 4.9 retorna os objetos do tipo `AOJExpressionStatement`. Eles também representam as expressões em programas que realizam chamadas aos métodos. As informações necessárias para aplicar esta refatoração são:

- O identificador da expressão;
- Sua implementação (corpo da expressão);

- A ordem da expressão no método;
- O identificador do elemento proprietário, neste cado o identificador do método ordenaNomes;
- O tipo do proprietário.

Elas são utilizadas para verificar se é possível aplicar a refatoração neste elemento e escrever as consultas de inserção e remoção. A Figura 4.2 mostra as informações da chamada do método Collections.sort do método ordenaNomes que será refatorado.

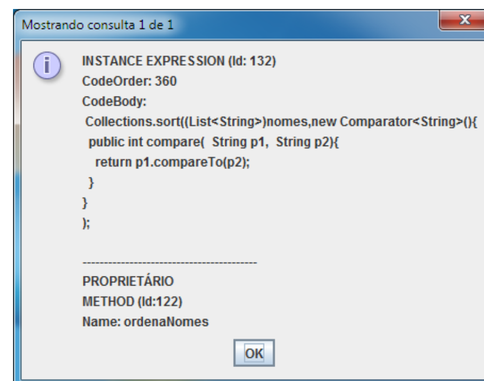


Figura 4.2 – Mostrando informações da chamada ao método Collections.sort

Com as oportunidades de refatoração e as informações identificadas, deve-se conhecer sua mecânica para aplicá-la. A mecânica da refatoração *Convert Collections.sort to sort* (TEIXEIRA JÚNIOR, 2014) é descrita pelos seguintes passos:

1. Utilize o método sort da própria coleção, ao invés do método sort da classe Collections;
2. Mova a implementação da classe anônima para o método sort da instância da coleção;
3. Aplique a refatoração *Convert Functional Interface Instance to Lambda Expression*;
4. Exclua a implementação antiga.

Como mostrado anteriormente, uma expressão lambda é representada pelo elemento expression em AQL. Então, deve-se adicionar o elemento expression após a chamada do método Collections.sort. Esta expressão deve utilizar o método sort da coleção e mover a implementação da classe anônima para o método sort da coleção.

A Listagem 4.10 mostra a instrução insert escrita em AQL. Esta consulta insere um elemento expression que possui como proprietário o método ordenaNomes, identificado pelo id 122, ordem 361, indicando que ele aparecerá após a instância do método Collections.sort, e a implementação do método sort da coleção, neste caso ((List<String>) nomes).sort((p1, p2) -> p1.compareTo(p2));.

```
1  insert expression e (owner, codeOrder, codeBody) values ( object method 122, 361, '
       ((List<String>) nomes).sort((p1, p2) -> p1.compareTo(p2));')
```

Listagem 4.10: Consulta em AQL para inserir a expressão lambda no método `ordenaNomes`

Após executar as instruções geradas pelo compilador AQL, o método `ordenaNomes` possui duas implementações, a do método `Collection.sort` e a do método `sort` da coleção. A Listagem 4.5 mostra o código gerado, a partir do metamodelo, após a execução da instrução de inserção. Observa-se a implementação do método `Collection.sort` nas linhas de 3 a 5 e a implementação utilizando expressão lambda na linha 6.

```
1  //...
2  public void ordenaNomes(){
3      Collections.sort((List<String>)nomes,new Comparator<String>(){
4          public int compare(String p1, String p2){ return p1.compareTo(p2); }
5      });
6      ((List<String>) nomes).sort((p1, p2) -> p1.compareTo(p2));
7  }
8  //...
```

Listagem 4.11: Método `ordenaNomes` após a inserção da expressão lambda

O próximo passo é excluir a implementação do método `Collection.sort`. Para isso, foi escrita uma expressão de remoção em AQL, como mostra a Listagem 4.12. A partir desta instrução, o compilador gerou as instruções de exclusão que foram executadas no AOPJungle. Este elemento é excluído a partir do seu tipo e identificador, neste caso, `expression` e 132, respectivamente. Essas informações são observadas na Figura 4.2.

```
1  delete expression e where e.id = 132
```

Listagem 4.12: Removendo a antiga implementação do método `sort` da classe `Collections`

A Listagem 4.13 mostra o código gerado, a partir do metamodelo, após utilizar as instruções de manipulação em AQL para aplicar a refatoração *Convert Collections.sort to sort*. Observa-se na Listagem 4.13 que o método `ordenaNomes` (linhas 16 a 18) foi reduzido e simplificado.

```
1  public class Pessoas {
2      private Collection<String> nomes = new ArrayList<>();
3      public void mostraNomes() {
4          nomes.forEach(nome -> System.out.println(nome));
5      }
6      public void mostraNomesW() {
7          nomes.forEach(nome -> System.out.println(nome));
8      }
9      public void inserir(String nome){
```

```
10          nomes.add(nome);
11      }
12      public void remover(String nome){
13          if (nomes.contains(nome))
14              nomes.remove(nome);
15      }
16      public void ordenaNomes(){
17          ((List<String>) nomes).sort((p1, p2) -> p1.compareTo(p2));
18      }
19      public void mostrarNomesIniciandoPor(String inicio) {
20          for (String nome : nomes)
21              if (nome.startsWith(inicio))
22                  System.out.println(nome);
23      }
24  }
```

Listagem 4.13: Classe `Pessoas` após aplicar a refatoração *Convert Collections.sort to sort*

4.3 APLICANDO *CONVERT ENHANCED FOR WITH IF TO LAMBDA FILTER*

Recomenda-se utilizar a refatoração *Convert Enhanced For with If to Lambda Filter* (TEIXEIRA JÚNIOR, 2014) quando um controle de seleção (`if`) é utilizado para implementar um filtro sobre uma coleção em Java. Sendo assim, deve-se substituir o controle de seleção pelo método `filter` das expressões lambda.

A fim de exemplificar a aplicação da refatoração *Convert Enhanced For with If to Lambda Filter*, o método `mostrarNomesIniciadosPor` da classe `Pessoas` foi utilizado. Ele possui a implementação de um laço de repetição `for each` que itera sobre a coleção de nomes da classe. Observa-se que a estrutura condicional `if` (linha 30) realiza um filtro na coleção e mostra apenas os nomes que iniciam por uma `String`, a qual é informada por parâmetro. A Listagem 4.14 mostra o método `mostrarNomesIniciadosPor` da classe `Pessoas`.

```
1  public class Pessoas {
2  //...
3      public void mostrarNomesIniciandoPor(String inicio) {
4          for (String nome : nomes)
5              if (nome.startsWith(inicio))
6                  System.out.println(nome);
7      }
```

```
8 }
```

Listagem 4.14: Código exemplo para aplicar a refatoração *Convert Enhanced For with If to Lambda Filter*

Para localizar as oportunidades da refatoração *Convert Enhanced For with If to Lambda Filter*, buscou-se por estruturas condicionais que estão dentro do escopo de laços de repetição `for each` e iteram sob uma coleção. A Listagem 4.15 mostra a consulta escrita em AQL. Para realizar as buscas, as instruções HQL geradas pelo compilador foram executadas em uma classe de testes do AOPJungle.

```
1 find if i where i.owner.metaName = 'Foreach' and i.owner.isCollection = true
    returns i.owner
```

Listagem 4.15: Busca por laços de repetição `for each` que implementam um filtro através de um condicional

A busca da Listagem 4.15 retorna objetos do tipo `AOJEnhancedForStatement` que são os proprietários dos condicionais buscados. A Figura 4.3 mostra as informações do elemento `for each` que possui uma estrutura condicional `if`. As informações mostradas são:

- O identificador do elemento `for each` (164);
- A implementação do `for each` (`CodeBody`);
- A ordem do elemento `for each` (455);
- E o identificador do elemento proprietário (160).

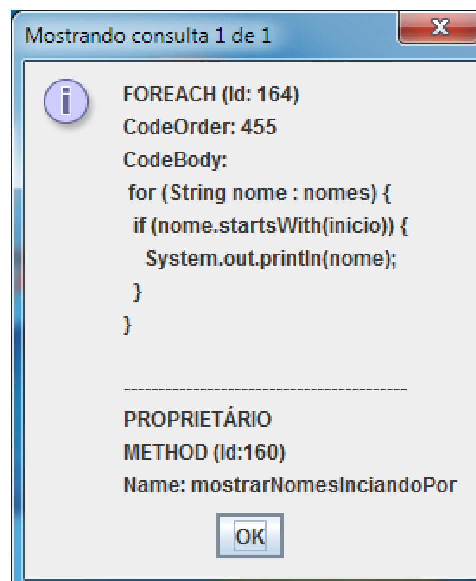


Figura 4.3 – Mostrando informações de elementos `for each` que possuam condicionais

Após identificar as oportunidades de refatoração e as informações necessárias, deve-se conhecer sua mecânica para aplicá-la. A mecânica da refatoração *Convert Enhanced For with If to Lambda Filter* (TEIXEIRA JÚNIOR, 2014) é descrita pelos seguintes passos:

1. Utilize o método `stream` e, em sequência, o método `filter` sobre a coleção;
2. Implemente a interface `Predicate` requerida pelo método `filter`, utilizando a mesma comparação da estrutura de seleção;
3. Adicione a chamada do método `forEach`;
4. Implemente a interface `Consumer` do método `forEach`;
5. Aplique a refatoração *Convert Functional Interface Instance to Lambda Expression*;
6. Apague a estrutura `for each` antiga.

Como visto anteriormente, uma expressão lambda é representada pelo elemento `expression` em AQL e pela classe `AOJExpressionStatement` no metamodelo do AOPJungle. Sendo assim, deve-se adicionar um elemento `expression` após a estrutura `for each` do método `mostrarNomesIniciadosPor`. Esta expressão deve usar o método `stream` da coleção seguido pelos métodos `filter` e `foreach`. Para isso, uma consulta `insert` foi escrita em AQL.

A consulta é mostrada na Listagem 4.16. Nesta consulta foram informados: o elemento proprietário da instrução, neste caso o método `mostrarNomesIniciadosPor` que possui identificação igual a 160, sua ordem, que deve maior que a estrutura `for each`, neste caso 456, e a implementação da expressão lambda, neste caso `nomes.stream().filter(nome -> nome.startsWith(inicio)).forEach(nome -> System.out.println(nome));`.

```
1  insert expression e (owner, codeOrder, codeBody) values (object method 160, 456, '
       nomes.stream().filter(nome -> nome.startsWith(inicio)).forEach(nome -> System.
       out.println(nome));')
```

Listagem 4.16: Inserirndo a expressão lambda que utiliza um filtro

Após inserir a expressão lambda, o método `mostrarNomesIniciandoPor` possui as duas implementações. A estrutura `for each` e a expressão lambda. A Listagem 4.17 mostra o método `mostrarNomesIniciandoPor` gerado a partir do metamodelo após a execução da instrução de inserção. Observa-se a antiga implementação nas linhas 4 a 6 e a expressão lambda na linha 7.

```
1  public class Pessoas {
2  //...
```

```
3       public void mostrarNomesIniciandoPor(String inicio) {
4           for (String nome : nomes)
5               if (nome.startsWith(inicio))
6                   System.out.println(nome);
7           nomes.stream().filter(nome -> nome.startsWith(inicio)).forEach(nome ->
                System.out.println(nome));
8       }
9   }
```

Listagem 4.17: Método `mostrarNomesIniciandoPor` após a inserção da expressão lambda

Por fim, deve-se remover a antiga implementação `for each`. Para isso, foi escrita uma consulta de remoção em AQL, como mostra a Listagem 4.18. A partir desta consulta, o compilador AQL gerou as instruções para remover o elemento `expression` identificado pelo `id` igual a 164. As informações do elemento `for each` são observadas na Figura 4.3.

```
1   delete foreach f where f.id = 164
```

Listagem 4.18: Removendo o laço de repetição `for each` do metamodelo

A Listagem 4.19 mostra a classe `Pessoa` gerada a partir do metamodelo após a aplicação das refatorações *Convert Enhanced For to Lambda Enhanced For*, *Convert Collections.sort to sort* e *Convert Enhanced For with If to Lambda Filter* (TEIXEIRA JÚNIOR, 2014). Observa-se que, após aplicar as três refatorações, houve uma redução de 11 linhas na classe `Pessoa`.

```
1   public class Pessoas {
2       private Collection<String> nomes = new ArrayList<>();
3       public void mostraNomes() {
4           nomes.forEach(nome -> System.out.println(nome));
5       }
6       public void mostraNomesW() {
7           nomes.forEach(nome -> System.out.println(nome));
8       }
9       public void inserir(String nome){
10          nomes.add(nome);
11      }
12      public void remover(String nome){
13          if (nomes.contains(nome))
14              nomes.remove(nome);
15      }
16      public void ordenaNomes(){
17          ((List<String>) nomes).sort((p1, p2) -> p1.compareTo(p2));
18      }
19      public void mostrarNomesInciandoPor(String inicio){
20          nomes.stream().filter(nome -> nome.startsWith(inicio)).forEach(nome ->
                System.out.println(nome));
```

```
21      }
22  }
```

Listagem 4.19: Código gerado pela AOPJungle após aplicar a refatoração *Convert Enhanced For with If to Lambda Filter*

4.4 APLICANDO *CONVERT ABSTRACT INTERFACE METHOD TO DEFAULT METHOD*

Aconselha-se utilizar a refatoração *Convert Abstract Interface Method to Default Method* (TEIXEIRA JÚNIOR, 2014) quando os métodos de uma interface possuem implementações vazias nos métodos das classes que as implementam. Para aplicar essa refatoração, recomenda-se a implementação de métodos `default` na interface. Nas versões anteriores de Java 8, as interfaces não permitiam a implementação de métodos concretos. Em Java 8, os métodos da interface podem ser definidos como `default` e possuírem uma implementação padrão.

Para exemplificar a aplicação desta refatoração foi utilizada a interface `Mostrar` e a classe `Cliente`. Observa-se na Listagem 4.20 a interface, linhas 1 a 4, e a classe, linhas 6 a 18. Os métodos `getters` e `setters` da classe `Cliente` foram ocultados, pois não são significativos para este exemplo. Nota-se que a classe `Cliente` implementa a interface `Mostrar` (linha 6). A interface `Mostrar` obriga a classe a implementar dois métodos: `mostrar` e `contar`. Observa-se na linha 16 da Listagem 4.20 que o método `contar` não possui uma lógica, ele apenas retorna um valor qualquer de forma a satisfazer a interface.

```
1   public interface Mostrar{
2       public void mostrar();
3       public int contar();
4   }
5
6   public class Cliente implements Mostrar{
7       private String nome;
8       private String cpf;
9       @Override
10      public void mostrar() {
11          System.out.println("Cliente:");
12          System.out.println("Nome: " + nome);
13          System.out.println("CPF: " + cpf);
14      }
15      @Override
16      public int contar() { return 0; }
17      //...
18  }
```

Listagem 4.20: Classe `Cliente` e Interface `Mostrar` antes de aplicar a refatoração

Para localizar as oportunidades da refatoração *Convert Abstract Interface Method to Default Method*, buscou-se por métodos vazios nas classes que implementam pelo menos uma interface. Um método foi considerado vazio quando: seu retorno for do tipo `void` e não possuir instruções ou quando seu retorno for diferente de `void` e possuir apenas a instrução de retorno do tipo especificado (TEIXEIRA JÚNIOR, 2014). A Listagem 4.21 mostra a consulta escrita em AQL que busca por métodos vazios. O código gerado pelo compilador foi executado em uma classe de testes do AOPJungle, a qual retornou os resultados da busca.

```
1  find method m
2  where m.owner.metaName = 'Class' and m.owner.implementsInterface = true
3      and ((m.statementsBody < 1)
4          or (m.statementsBody <= 1 and m.returnTypeName <> 'void'))
5      and m.hasAnnotation = true
6  returns m
```

Listagem 4.21: Buscando por métodos vazios em classes que implementam pelo menos uma interface

A consulta da Listagem 4.21 retorna os métodos vazios das classes. Os métodos são representados pela classe `AOJMethodDeclaration` do metamodelo. Neste caso, o método `contar` da classe `Cliente` foi selecionado. A Figura 4.4 mostra algumas informações do método, tais como:

- O nome da classe que ele pertence (`Cliente`);
- As interfaces que a classe implementa (`Mostrar`);
- O identificador do método (16);
- A implementação do método.

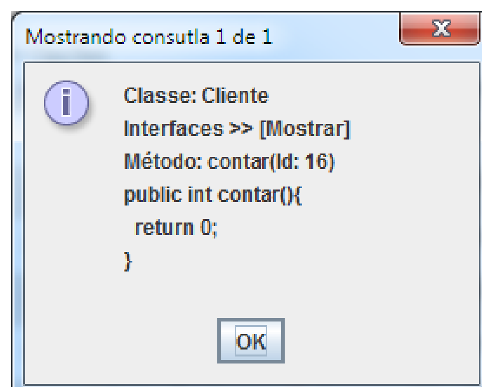


Figura 4.4 – Mostrando informações do método `contar` da classe `Cliente`

Porém, para aplicar essa refatoração, além das informações do método são necessárias algumas informações da interface. Dessa forma, outra consulta foi escrita em AQL a

fim de selecionar o método `contar` da interface `Mostrar`. Observa-se a consulta de busca na Listagem 4.22.

```
1  find method m where m.owner.name = 'Mostrar' and m.name = 'contar' returns m
```

Listagem 4.22: Buscando pelo método `contar` na interface `Mostrar`

A consulta da Listagem 4.22 retorna o método `contar` da interface `Mostrar`. A Figura 4.4 mostra algumas informações que foram usadas nas consultas de manipulação, tais como:

- O nome da interface (`Mostrar`);
- O nome do método (`contar`);
- O identificador do método (174);
- A assinatura do método (`public int contar()`).

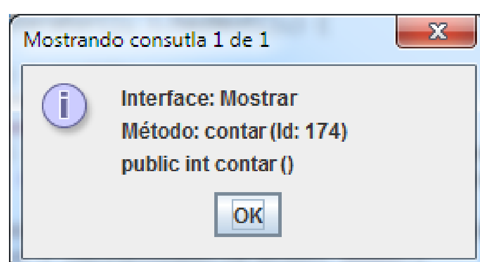


Figura 4.5 – Mostrando informações do método `contar` da interface `Mostrar`

Após identificar a oportunidade de refatoração e conhecer algumas informações, deve-se conhecer sua mecânica para aplicá-la. A mecânica da refatoração *Convert Abstract Interface Method to Default Method* (TEIXEIRA JÚNIOR, 2014) é descrita pelos seguintes passos:

1. Adicione o modificador `default` no método da interface;
2. Forneça um código ao método `default`, se desejar. Ou deixe o corpo do método vazio, caso contrário;
3. Remova os métodos vazios das classes que implementem a interface.

Primeiramente, deve-se adicionar o modificador `default` no método `contar`. Para isso, a propriedade `signature` do método `contar` da interface `Mostrar` foi modificada. Ela representa a assinatura do método. A fim de modificar essa propriedade, uma instrução de atualização foi escrita em AQL e executada no AOPJungle. Observa-se na Listagem 4.23 a instrução `update`, que modifica a assinatura do método que possui identificador igual a 174 para `public int default contar()`. Na atual implementação não é possível adicionar

apenas o modificador `default` no método, dessa forma, é necessário informar toda sua assinatura.

```
1 update method m (signature) values ('public int default contar()') where m.id = 174
```

Listagem 4.23: Instruções em AQL para atualizar a assinatura do método `contar` da interface `Mostrar`

O próximo passo é informar uma implementação padrão para o método da interface. Dessa forma, foi adicionada uma instrução de retorno ao método `contar`. As expressões de retorno são representadas pelo elemento `expression` na AQL e pela classe `AOJExpressionStatement` no metamodelo. Observa-se na Listagem 4.24 a consulta, em AQL, para inserir a expressão de retorno no método `contar`. Essa instrução é associada ao método através da propriedade `owner`, que representa o proprietário da expressão. O método é referenciado através do seu identificador (174). A expressão possui ordem igual a 1 e implementação igual a `return 0;`.

```
1 insert expression e (owner, codeOrder, codeBody) values (object method 174, 1, '
      return 0;')
```

Listagem 4.24: Instrução em AQL para adicionar uma implementação `default` para o método `contar`

Por fim, deve-se remover os métodos vazios das classes que implementam a interface `Mostrar`. Neste caso, o método `contar` deve ser removido da classe `Cliente`. Utilizando as informações mostradas na Figura 4.4, foi escrita uma consulta em AQL para remover o método `contar` que possui o identificador igual a 16. Observa-se na Listagem 4.25 tal instrução. As instruções geradas pelo compilador AQL a partir da consulta foram executadas em uma classe de testes no AOPJungle que removeu o método `contar` da classe `Cliente`.

```
1 delete method m where m.id = 16
```

Listagem 4.25: Removendo o método `contar` da classe `Cliente`

Após aplicar a refatoração, foi utilizado o gerador de código do `AOPJungle` para mostrar a interface `Mostrar` e a classe `Cliente` após executar as instruções de manipulação. Observa-se nas linhas de 1 a 6 da Listagem 4.26 a interface e nas linhas de 8 a 18 a classe. Neste caso, a classe `Cliente` foi simplificada, enquanto a complexidade da interface `Mostrar` aumentou.

```
1 public interface Mostrar{
2     public void mostrar();
3     public int default contar(){
4         return 0;
5     }
```

```
6  }
7
8  public class Cliente implements Mostrar{
9      private String nome;
10     private String cpf;
11
12     public void mostrar(){
13         System.out.println("Cliente:");
14         System.out.println("Nome: " + nome);
15         System.out.println("CPF: " + cpf);
16     }
17     //...
18 }
```

Listagem 4.26: Classe `Cliente` e interface `Mostrar` após aplicar a refatoração

4.5 APLICANDO *CHAIN CONSTRUCTORS*

Recomenda-se utilizar a refatoração *Chain Constructors* (KERIEVSKY, 2008; PAULI, 2014) quando existem métodos construtores que possuem código duplicado. Para isso, deve-se realizar a chamada de um método construtor mais genérico para um construtor mais específico. E assim encadear os construtores até que o construtor principal da cadeia seja chamado. Normalmente o construtor principal é aquele que possui o maior número de parâmetros. Essa refatoração elimina código duplicado do sistema e evita que um desenvolvedor se esqueça de replicar determinado comportamento a uma nova instância de um construtor.

A fim de exemplificar aplicação desta refatoração, a classe `Pagamento` foi utilizada (Listagem 4.27). A classe `Pagamento` possui três métodos construtores que compartilham código duplicado. Tais métodos são mostrados nas linhas de 8 a 15, de 16 a 23 e de 24 a 31. Para esta refatoração é importante identificar o construtor mais genérico da classe. Em ordem, nota-se que o construtor mais genérico é o da linha 8, seguido pelo da linha 24 e, por fim, o da linha 16. Dessa forma, para aplicar essa refatoração, o construtor da linha 16 deve chamar o da linha 24, que por sua vez chamará o da linha 8, que realizará a atribuição dos valores passados por parâmetros aos atributos da classe. Os métodos `getters` e `setters` da classe foram ocultados, pois não são significativos para este exemplo.

```
1  public class Pagamento{
2      private String descricao;
3      private double valorTotal;
4      private double valorPago;
5      private int quantidade;
6      private Date inicio;
```

```
7       private Date fim
8       public Pagamento(String descricao, double valorTotal, double valorPago, int
            quantidade, Date inicio, Date fim){
9           this.descricao=descricao;
10          this.valorTotal=valorTotal;
11          this.valorPago=valorPago;
12          this.quantidade=quantidade;
13          this.inicio=inicio;
14          this.fim=fim;
15      }
16      public Pagamento(String descricao, double valorTotal, Date inicio){
17          this.descricao=descricao;
18          this.valorTotal=valorTotal;
19          this.valorPago=0;
20          this.quantidade=0;
21          this.inicio=inicio;
22          this.fim=null;
23      }
24      public Pagamento(String descricao, double valorTotal, double valorPago, Date
            inicio){
25          this.descricao=descricao;
26          this.valorTotal=valorTotal;
27          this.valorPago=valorPago;
28          this.quantidade=0;
29          this.inicio=inicio;
30          this.fim=null;
31      } //...
32  }
```

Listagem 4.27: Classe `Pagamento`

Para localizar as classes que possuem oportunidades da refatoração *Chain Constructors* foi escrita uma consulta em AQL que seleciona todas as classes que possuem mais de dois métodos construtores com duas ou mais instruções. Observa-se na Listagem 4.28 a consulta em AQL. Ela foi escrita no compilador AQL e executada em uma classe de testes do AOPJungle.

```
1  find constructor c where c.statementsBody > 2 returns c.owner group by c.owner.id
       having count(c.owner.id) > 2
```

Listagem 4.28: Buscando por oportunidades da refatoração *Chain Constructors*

A consulta da Listagem 4.28 retorna as classes que devem ser analisadas e, posteriormente, refatoradas. A classe `AOJClassDeclaration` representa as classes de programas no metamodelo. Através dela é possível visualizar as informações relevantes para a análise e aplicação da refatoração, são elas:

- O identificador da classe, 203;

- O nome da classe, `Pagamento`;
- O identificador dos construtores, 203, 222 e 235;
- E a implementação dos construtores.

A Figura 4.6 mostra as informações da classe `Pagamento` e seus respectivos construtores. Analisando os construtores, observa-se que todos eles possuem código duplicado. Por exemplo, as instruções `this.descricao = descricao`, `this.valorTotal = valorTotal` e `this.inicio = inicio` estão duplicadas nos três métodos.

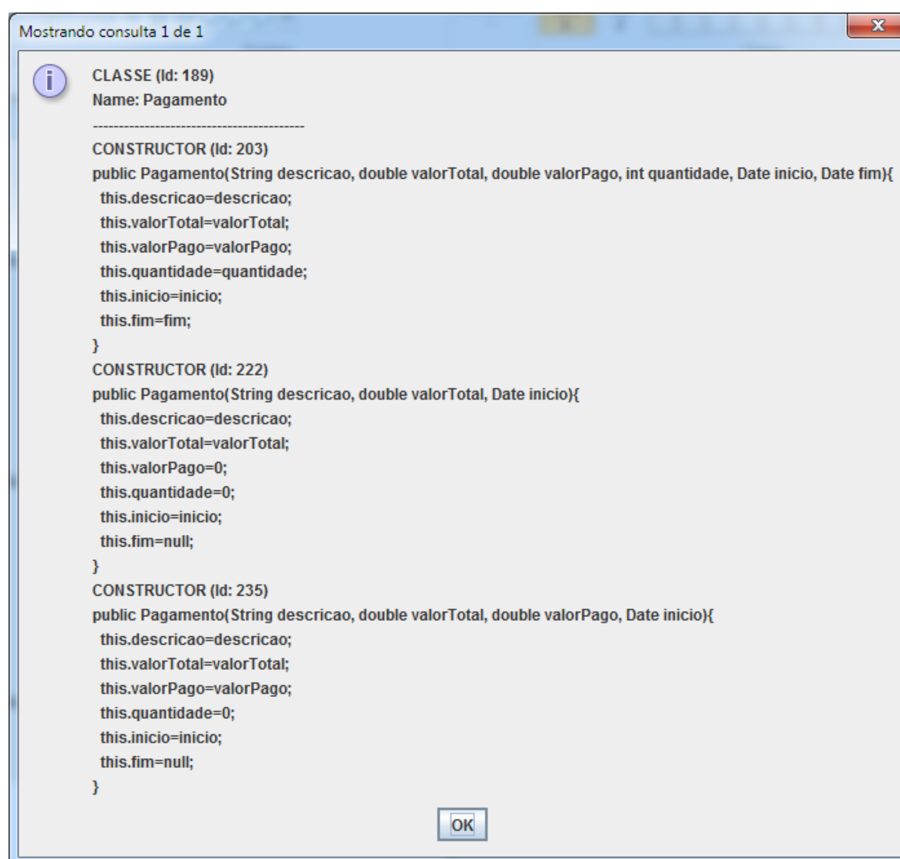


Figura 4.6 – Mostrando informações da classe `Pagamento`

Tendo pelo menos uma oportunidade de refatoração e as informações dos elementos, deve-se conhecer sua mecânica para aplicá-la. A mecânica da refatoração *Chain Constructors* (KERIEVSKY, 2008) é descrita pelos seguintes passos:

1. Encontre dois construtores que contenham código duplicado e determine se um pode chamar o outro de forma que o código duplicado possa ser apagado de um destes construtores. Faça com que um construtor chame o outro construtor de forma que o código duplicado seja reduzido ou eliminado;
2. Repita o 1º passo para todos os construtores da classe;

3. Modifique a visibilidade dos construtores caso necessário.

O primeiro passo a ser executado é remover as instruções do construtor mais específico. Neste caso, o construtor da linha 16 da Listagem 4.27. Observa-se que as instruções deste método são apenas atribuições. As atribuições são definidas como expressões. Em AQL elas são representadas pelo elemento `expression` e no metamodelo pela classe `AOJExpressionStatement`. Sendo assim, foi escrita uma instrução AQL para remover todas as expressões que possuem como proprietário o construtor que possui o identificador 222, como mostra a linha 1 da Listagem 4.29.

Observa-se na linha 2 da Listagem 4.29 uma instrução escrita em AQL para adicionar a chamada ao construtor mais genérico dentro do construtor mais específico. Esta instrução insere uma expressão (`expression`) que possui o construtor de identificador 222 como proprietário (`owner`) , ordem (`codeOrder`) igual a 5 e implementação (codeBody) igual a `this(descricao, valorTotal, 0, inicio)`.

```
1  delete expression e where e.owner.id = 222
2  insert expression e (owner, codeOrder, codeBody) values (object constructor 222, 5,
       'this(descricao, valorTotal, 0, inicio);')
```

Listagem 4.29: Removendo as instruções do construtor e adicionando chamada ao construtor mais genérico

Após executar as instruções de remoção e inserção, a classe `Pagamento` pode ser observada na Listagem 4.30. Nota-se que o construtor da linha 16 apenas realiza uma chamada para o próximo construtor, o da linha 19.

```
1  public class Pagamento{
2      private String descricao;
3      private double valorTotal;
4      private double valorPago;
5      private int quantidade;
6      private Date inicio;
7      private Date fim;
8      public Pagamento(String descricao, double valorTotal, double valorPago, int
           quantidade, Date inicio, Date fim){
9          this.descricao=descricao;
10         this.valorTotal=valorTotal;
11         this.valorPago=valorPago;
12         this.quantidade=quantidade;
13         this.inicio=inicio;
14         this.fim=fim;
15     }
16     public Pagamento(String descricao, double valorTotal, Date inicio){
17         this(descricao, valorTotal, 0, inicio);
18     }
```

```
19      public Pagamento(String descricao, double valorTotal, double valorPago, Date
            inicio){
20          this.descricao=descricao;
21          this.valorTotal=valorTotal;
22          this.valorPago=valorPago;
23          this.quantidade=0;
24          this.inicio=inicio;
25          this.fim=null;
26      } //...
27  }
```

Listagem 4.30: Classe `Pagamento` após aplicar as instruções AQL

O próximo passo descrito pela mecânica da refatoração é repetir esse procedimento para os demais construtores que possuem código duplicado. Nota-se que os construtores das linhas 8 e 19 da Listagem 4.30 ainda possuem código duplicado. Entre eles, o construtor mais específico é o da linha 19 que possui o identificador igual a 235, como mostrado na Figura 4.6. Sendo assim, os passos anteriores foram repetidos. Ou seja, duas consultas em AQL foram executadas. Uma para remover as expressões do construtor e uma para adicionar a chamada ao construtor principal. Observa-se a consulta de remoção na linha 1 da Listagem 4.29. Ela remove todas as expressões que possuem como proprietário o construtor identificado pelo `id` 235. E a consulta de inserção é mostrada na linha 2 da Listagem 4.29. Esta consulta adiciona uma expressão que realiza a chamada para o construtor principal da classe. Esta expressão é associada ao proprietário. Neste caso o construtor mais específico que possui identificador igual a 235, ordem igual a 5 e a chamada ao construtor principal `this(descricao, valorTotal, valorPago, 0, inicio, null);`.

```
1  delete expression e where e.owner.id = 235
2  insert expression e (owner, codeOrder, codeBody) values (object constructor 235, 5,
       'this(descricao, valorTotal, valorPago, 0, inicio, null);')
```

Listagem 4.31: Removendo as instruções do construtor e adicionando chamada ao construtor principal

A Listagem 4.32 mostra a classe `Pagamento` após aplicar a refatoração *Chain Constructors*. Ela foi gerada a partir do metamodelo modificado. Observam-se os construtores modificados nas linhas 16 e 19. Comparando as duas implementações, observa-se a redução e a eliminação de código duplicado entre as Listagens 4.27 e 4.32.

```
1  public class Pagamento{
2      private String descricao;
3      private double valorTotal;
4      private double valorPago;
5      private int quantidade;
6      private Date inicio;
7      private Date fim;
```

```
8       public Pagamento(String descricao, double valorTotal, double valorPago, int
            quantidade, Date inicio, Date fim){
9           this.descricao=descricao;
10          this.valorTotal=valorTotal;
11          this.valorPago=valorPago;
12          this.quantidade=quantidade;
13          this.inicio=inicio;
14          this.fim=fim;
15      }
16      public Pagamento(String descricao, double valorTotal, Date inicio){
17          this(descricao, valorTotal, 0, inicio);
18      }
19      public Pagamento(String descricao, double valorTotal, double valorPago, Date
            inicio){
20          this(descricao, valorTotal, valorPago, 0, inicio, null);
21      }
22      //...
23  }
```

Listagem 4.32: Classe `Pagamento` após aplicar a refatoração

4.6 CONSIDERAÇÕES FINAIS

Este estudo de caso apresentou o uso dos novos recursos adicionados na implementação de referência da linguagem AQL. Para isso foram utilizadas cinco refatorações, das quais três realizam transformações de estruturas "convencionais" para uma expressão lambda, uma realiza a implementação de métodos `default` em interface e uma remove a duplicação de código nos construtores da classe (TEIXEIRA JÚNIOR, 2014; KERIEVSKY, 2008); e as classes exemplos `Pessoa`, `Cliente` e `Pagamento` e a interface `Mostrar` que possuem as oportunidades de refatorações buscadas.

Dos cinco cenários propostos, foi possível identificar e aplicar as refatorações utilizando consultas AQL para buscar, inserir, atualizar e remover os elementos em programas. Para isso, foi utilizado o compilador AQL que gerou as instruções para a linguagem alvo e foram executadas no AOPJungle. Para cada um dos cenários foi criada uma classe de testes para executar as instruções de busca e manipulação. Por fim, as classes refatoradas foram geradas utilizando o gerador de código desenvolvido no AOPJungle, o qual gera o código a partir do metamodelo. O próximo capítulo apresenta as considerações finais e referências a trabalhos futuros.

5 CONCLUSÃO

Esta dissertação apresentou a especificação e implementação de um conjunto de melhorias para a linguagem AQL (FAVERI, 2013). A atual implementação de referência da linguagem AQL permite realizar consultas utilizando granularidade mais fina, permite o uso de cláusulas para manipulação de programas e fornece melhorias na clausula *order by* e nas regras de validação da linguagem.

Através destas melhorias, é possível buscar por oportunidades de refatoração dentro de métodos e construtores, como por exemplo as refatorações relacionadas a expressões lambda definidas para Java 8 (TEIXEIRA JÚNIOR, 2014) e as refatorações para padrões de projeto (KERIEVSKY, 2008; PAULI, 2014). Além disso, é possível aplicar a refatoração utilizando as novas cláusulas de manipulação de dados (`insert`, `update` e `delete`).

Para validar a extensão proposta, algumas modificações foram realizadas no compilador de AQL e no *framework* AOPJungle. A principal ferramenta utilizada na extensão do compilador AQL foi o *workbench* Xtext, o qual gerencia todo o ciclo de implementação da linguagem. A implementação original do compilador AQL implementava regras de transformações para a linguagem HQL (*Hibernate Query Language*). Porém, durante a implementação das cláusulas de manipulação, verificou-se que, até o momento, as instruções de manipulação de dados de HQL são limitadas (HIBERNATE, 2017). Desta forma, optou-se por utilizar o Hibernate e Java para realizar as transformações das consultas de manipulação de dados, além de HQL.

O *framework* AOPJungle fornece métricas e informações a respeito de sistemas implementados em Java e AspectJ. Ele foi melhorado para dar suporte aos novos tipos de elementos inseridos na atual implementação de referência da linguagem AQL, tais como: `method`, `constructor`, `foreach`, `if` e `expression`.

Para avaliar o uso da extensão da linguagem AQL como apoio nas atividades de busca por oportunidades de refatoração e aplicação de refatoração em código orientado a objetos, foi realizado um estudo de caso utilizando cinco refatorações. São elas: *Convert Enhanced For to Lambda Enhanced For*, *Convert Collections.sort to sort*, *Convert Enhanced For with If to Lambda Filter*, *Convert Abstract Interface Method to Default Method* e *Chain Constructors*. De forma a mostrar a aplicabilidade da abordagem, foram utilizadas algumas classes exemplos que possuem oportunidades para aplicar estas refatorações. A principal motivação para a escolha destas refatorações foi demonstrar o uso dos novos recursos adicionados na AQL.

5.1 TRABALHOS FUTUROS

A seguir são apresentadas algumas sugestões para trabalhos futuros.

- **Adicionar suporte a subconsultas.** A atual implementação da linguagem AQL não fornece suporte a subconsultas. Este recurso pode facilitar o uso das cláusulas de remoção, inserção e atualização. Na atual implementação, um elemento somente é referenciado através do seu identificador. No lugar deste identificador, uma consulta de busca poderia se utilizada.

- **Separação dos módulos orientado a objetos e orientado a aspectos.** A atual implementação da linguagem AQL possui apenas um núcleo que contém os módulos orientado a objetos (OO) e orientado a aspectos (AO). A fim de desenvolver recursos específicos para cada paradigma, facilitar a evolução da linguagem e facilitar a evolução do modelo de referência, o núcleo da linguagem pode ser dividido em dois: um núcleo OO e outro AO.

- **Aplicar as modificações em programas.** Na atual implementação de referência da linguagem AQL as modificações são realizadas no metamodelo instanciado pelo AOPJungle. Porém, elas não são refletidas nos sistemas, apenas é possível gerar código a partir de um elemento do metamodelo. Este elemento pode ser uma classe, um método ou uma interface. O código é gerado no console do AOPJungle. Tais modificações poderiam ser refletidas no código fonte do sistema.

- **Atualizar o módulo de consulta.** Na original implementação de referência da linguagem AQL o módulo de consulta desenvolvido no AOPJungle dá suporte a consultas HQL. Seria interessante modificá-lo para receber qualquer consulta em AQL, ou seja, receber instruções HQL, Hibernate e Java.

- **Validar os códigos manipulados.** Na atual implementação da linguagem AQL as modificações realizadas no metamodelo não são validadas, ou seja, o uso de instruções de manipulação de dados podem resultar em código inválido. Seria interessante validar esse código antes de realizar uma operação remoção, atualização ou inserção. Quando houver uma inconsistência, o sistema deve alertar o usuário e fornecer opções para permitir a execução da consulta deixando o código inválido, não permitir a execução da consulta ou realizar a execução da consulta em cascata.

REFERÊNCIAS BIBLIOGRÁFICAS

AJDT. **Project AJDT**. 2017. Disponível em: <http://www.eclipse.org/ajdt/>.

BAJRACHARYA, J. O. S.; LOPES, C. Sourcerer: An infrastructure for large-scale collection and analysis of open-source code. **Science of Computer Programming, Vol. 79**, p. 241–259, 2014.

BAUER, C.; KING, G. **Hibernate in Action**. [S.l.]: Manning Publications, 2004. ISBN 193239415X.

BENTLEY, J. L. Little languages for pictures in AWK. **AT&T Technical Journal, Vol. 68**, p. 21–32, 1989.

BETTINI, L. **Implementing Domain-Specific Languages with Xtext and Xtend**. [S.l.]: Packt Publishing, 2013. ISBN 1782160302.

BISON. **GNU Bison**. 2017. Disponível em: <https://www.gnu.org/software/bison/>.

BOEHM, B.; IN, H. Identifying quality-requirement conflicts. **IEEE Software**, p. 25–35, 2009.

BOIS, B. D. **A Study of Quality Improvements by Refactoring**. 2006. Tese (Doutorado) — Universiteit Antwerpen (Belgium), 2006. AAI3251211.

BOIS, B. D.; MENS, T. Describing the impact of refactoring on internal program quality. **International Workshop on Evolution of Large-Scale Industrial Software Applications**, p. 37–48, 2003.

CANVIZ. **Canviz**. 2017. Disponível em: <https://code.google.com/archive/p/canviz/>.

CLANG. **clang: a C language family frontend for LLVM**. 2017. Disponível em: <https://clang.llvm.org/>.

COELHO, W.; MURPHY, G. C. Presenting crosscutting structure with active models. **AOSD ’06 Proceedings of the 5th International Conference on Aspect-Oriented Software Development**, p. 158–168, março 2005.

COHEN, T.; GIL, J. Y.; MAMAN, I. The java tools language. **ACM SIGPLAN Conference on Object-Oriented Programming Sysytems, Languages, and Applications**, p. 89–108, 2006.

CPPDEPEND. **CppDepend Features**. 2017. Disponível em: <http://www.cppdepend.com/>.

CQLINQ. **CQLinq Syntax**. 2017. Disponível em: <http://www.ndepend.com/docs/cqlinq-syntax>.

CSS. **W3C: Cascading Style Sheets**. 2017. Disponível em: <https://www.w3.org/Style/CSS/>.

DAMAS, L. **SQL. Structured Query Language**. [S.l.]: LTC, 2007. ISBN 8521615582.

DEURSEN, A. van; KLINT, P.; VISSER, J. Domain-specific languages: an annotated bibliography. **ACM SIGPLAN**, p. 26–39, 2000.

DOT. **The DOT Language**. 2017. Disponível em: <http://www.graphviz.org/content/dot-language>.

EYSHOLDT, M.; BEHRENS, H. Xtext: implement your language faster than the quick and dirty way. **Proceedings of the ACM international conference companion**, p. 307–309, 2010.

FAVERI, C. **Uma Linguagem Específica de Domínio para Consulta em Código Orientado a Aspectos**. 2013. Dissertação (Mestrado) — Universidade Federal de Santa Maria-RS, Brasil, 2013.

FIT. **FIT: Framework for Integrated Test**. 2017. Disponível em: <http://fit.c2.com/>.

FOWLER, M. et al. **Refactoring: Improving the Design of Existing Code**. [S.l.]: Addison-Wesley Professional, 1999. ISBN 0201485672.

FOWLER, M.; PARSONS, R. **Domain-Specific Languages (Addison-Wesley Signature Series (Fowler))**. [S.l.]: Addison-Wesley Professional, 2010. ISBN 0321712943.

GAMMA, E. et al. **Padrões de Projetos: Soluções Reutilizáveis de Software Orientados a Objetos**. [S.l.]: Bookman, 2000. ISBN 8573076100.

GRAPHVIZ. **Graphviz - Graph Visualization Software**. 2017. Disponível em: <http://www.graphviz.org/>.

GRONBACK, R. C. **Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit (Eclipse Series)**. [S.l.]: Addison-Wesley Professional, 2009.

GROUP, A. C. **Krugle home page**. 2017. Disponível em: <http://www.krugle.com/>.

HIBERNATE. **Hibernate: Community Documentation**. 2017. Disponível em: <https://docs.jboss.org/hibernate/orm/3.5/reference/pt-BR/html/queryhql.html>.

HUB, B. D. O. **Sourcerer: An infrastructure for large-scale collection and analysis of open-source code**. 2017. Disponível em: <http://code.openhub.net/>.

ISO, I. Iec 9126-1: Software engineering - product quality - part 1: Quality model. **ICSC ´08 Proceedings of the 2008 IEEE International Conference on Semantic Computing**, June 2001.

JACKPOT. **Jackpot**. 2017. Disponível em: <http://wiki.netbeans.org/Jackpot>.

JETBRAINS. **JetBrains Meta Programming System.** 2017. Disponível em: <https://www.jetbrains.com/mps/>.

JMOCK. **jMock API Reference Documentation**. 2017. Disponível em: <http://www.jmock.org/javadoc.html>.

JOHNSON, S. C. **Yacc: Yet Another Compiler-Compiler**. 2017. Disponível em: <http://dinosaur.compilertools.net/yacc/>.

KATAOKA, Y. et al. Automated support for program refactoring using invariants. **ICSM ´01 IEEE International Conference on Software Maintenance (ICSM01)**, p. 736–744, 2001.

KERIEVSKY, J. **Refatoração Para Padrçõs**. [S.l.]: BOOKMAN - GRUPO A, 2008. ISBN 8577802442.

KERSTEN, M.; MURPHY, G. C. Mylar: a degree-of-interest model for ides. **In Proceedings of the 4th International Conference on Aspect-Oriented Software Development**, p. 159–168, 2005.

KLASSEN, L.; WAGNER, R. Emorf - a tool for model transformations. **Electronic Communications of the EASST**, v. 54, 2012.

KUHN, T.; THOMANN, O. **Abstract Syntax Tree**. 2017. Disponível em: <http://www.eclipse.org/articles/Article-JavaCodeManipulation{_}A>.

KULLBACH, B.; WINTER, A. Querying as an enabling technology in software reengineering. **Procedings of the 3rd Euromicro Conference on Software Maintenance and Reengineering**, p. 42–50, 1999.

LATEX. **LaTeX – A document preparation system**. 2017. Disponível em: <https://www.latex-project.org/>.

LESS. **A linguagem dinâmica de estilos**. 2017. Disponível em: <http://lesscss.loopinfinito.com.br/>.

LINQ. **LINQ: .NET Language-Integrated Query**. 2017. Disponível em: <https://msdn.microsoft.com/en-us/library/bb308959.aspx>.

LISP. **Common Lisp Documentation**. 2017. Disponível em: <http://www.lispworks.com/documentation/common-lisp.html>.

MAKE. **GNU Make**. 2017. Disponível em: <http://www.gnu.org/software/make/>.

MCCORMICK, E.; VOLDER, K. D. Jquery: finding your way through tangled code. **Companion tho the 19th ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications Companion**, p. 9–10, 2004.

MENS, T.; TOURWE, T. A survey of software refactoring. **IEEE Transactions on Software Engineering**, v. 20, p. 126–139, February 2004.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and how to develop domain-specific languages. **ACM Computing Surveys**, p. 316–344, 2005.

OPDYKE, W. **Refactoring Object-oriented Frameworks**. 1992. Tese (Doutorado) — University of Illinois, 1992.

PARR, T. **The Definitive ANTLR 4 Reference**. [S.l.]: Pragmatic Bookshelf, 2013. ISBN 1934356999.

PAULI, G. B. **Busca por Oportunidades de Refatoração para Aplicação de Padroes de Projeto**. 2014. Dissertação (Mestrado) — Universidade Federal de Santa Maria-RS, Brasil, 2014.

PFEIFFER, J. H.; SARDOS, A.; GURD, J. R. Complex code querying and navigation for AspectJ. **OOPSLA Workshop on Eclipse Technology Exchange, 2005**, p. 60–54, 2005.

PIVETA, E. K. **Improving the Search for Refactoring Opportunities on Object-oriented and Aspect-oriented Software**. 2009. Tese (Doutorado) — Universidade Federal do Rio

Grande do Sul. Instituto de Informática. Programa de Pós-Graduação em Computação., 2009.

PMD. **PMD - Don't shoot the messenger**. 2017. Disponível em: <https://pmd.github.io/>.

PRESSMAN, R. S. **Engenharia de Software - uma Abordagem Profissional**. [S.l.]: Mc Graw Hill, 2011. ISBN 8563308335.

PYDOT. **Pydot**. 2017. Disponível em: <https://pypi.python.org/pypi/pydot>.

RAILS, R. on. **Ruby on Rails Guides**. 2017. Disponível em: <http://guides.rubyonrails.org/>.

REGEXR. **RegExr Documentation**. 2017. Disponível em: <http://regexr.com/>.

SASS. **Sass Documentation**. 2017. Disponível em: <https://sass-lang.com/>.

SEEMLE. **Seemle QL: Code Exploration**. 2017. Disponível em: <https://semmle.com/products/semmle-ql/>.

SILLITO, J.; MURPHY, G. C.; VOLDER, K. D. Questions programmers ask during software evolution tasks. **Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering**, p. 23–34, 2006.

SOMMERVILLE, I. **Engenharia de Software**. [S.l.]: Pearson, 2011. ISBN 8579361087.

SQL. **ISO/IEC 9075-1:2016 Preview - Information technology**. 2017. Disponível em: <https://www.iso.org/standard/63555.html>.

TEIXEIRA JÚNIOR, J. E. **Um Catálogo de Refatorações Envolvendo Expressões Lambda em Java**. 2014. Dissertação (Mestrado) — Universidade Federal de Santa Maria-RS, Brasil, 2014.

URMA, R.-G.; MYCROFT, A. Programming language evolution via source code query languages. **Proceedings of the ACM 4th annual workshop on Evaluation and usability of programming languages and tools**, p. 35–38, 2012.

URMA, R. G.; MYCROFT, A. Source-code queries with graph databases - with application to programming language usage and evolution. **Science of Computer Programming, Vol. 97**, p. 127–134, 2015.

VERBAERE, M.; HAJIYEV, E.; MOOR, D. Improve software quality with semmlecode: an eclipse plugin for semantic code search. **Companion to the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Sysytems, Languages, and Applications**, p. 89–108, 2007.

VOLDER, K. D. **Type-Oriented Logic Meta Programming**. 1998. Tese (Doutorado) — Department of Computer Science, Vrije Universiteit Brussel, Belgium, 1998.

W3C. **Comunidade Word Wide Web**. 2017. Disponível em: <https://www.w3.org/standards/webdesign/htmlcss>.

WALMSLEY, P. **XQuery**. [S.l.]: O'Reilly Media, 2007. ISBN 0596006349.

XAML. **Documentação XAML**. 2017. Disponível em: <https://msdn.microsoft.com/pt-br/library/ms752059(v=vs.110).aspx>.

XML. **W3C: Extensible Markup Language (XML)**. 2017. Disponível em: <https://www.w3.org/XML/>.

XTEND. **Xtend Documentation**. 2017. Https://eclipse.org/xtend/documentation/.

ZLOOF, M. M. Query by example. **AFIPS ’75 Proceedings of the May 19-22, 1975, national computer conference and exposition**, p. 35–38, 1975.

ANEXO A – SINTAXE ORIGINAL DE AQL EM XTEXT

```
1  Query hidden (WS, SL_COMMENT, ML_COMMENT) : query=QueryStatement ;
2  QueryStatement : (find=FindClause) (where=WhereClause)? (return=ReturnsClause) (
       orderby=OrderByClause)? (groupby=GroupByClause)? ;
3  FindClause : clause='find' bindingObject+=BindingObject (',' bindingObject+=
       BindingObject)* ;
4  BindingObject : type=ObjectType (alias+=ID)+ ;
5  ObjectType : {Project} value='project' | {Package} value='package' | {Class} value=
       'class' | {Aspect} value='aspect' | {Interface} value='interface' | {Enum} value
       ='enum' ;
6  WhereClause : clause='where' expression=Expression ;
7  ReturnsClause : clause='returns' expressions+=Expression ('as' resultAlias+=ID)? (
       ',' expressions+=Expression ('as' resultAlias+=ID)?)* ;
8  GroupByClause : clause='group by' expressions+=SimpleQualifiedName (',' expressions
       +=SimpleQualifiedName)* ;
9  OrderByClause : clause='order by' expressions+=SimpleQualifiedName (',' expressions
       +=SimpleQualifiedName)* option=('asc' | 'desc')? ('having' havingExpression=
       Expression)? ;
10 SimpleQualifiedName returns QualifiedName : qualifiers+=SimpleQualified ('.'
       qualifiers+=SimpleQualified)* ;
11 SimpleQualified returns QualifiedElement : Identifier ;
12 Expression hidden (WS, SL_COMMENT, ML_COMMENT) : Or ;
13 Or returns Expression : And ( ('or'{Or.left=current}) right=And)* ;
14 And returns Expression : Relation ( ('and'{And.left=current} right=Relation))* ;
15 Relation returns Expression : Add ( ( ('='{Equals.left=current}) | ('>'{Greater.
       left=current}) | ('>='{GreaterEqual.left=current}) | ('<'{Less.left=current}) |
       ('<='{LessEqual.left=current}) | ('like'{Like.left=current}) | ('!='{NotEquals.
       left=current} )) right=Add ) * ;
16 Add returns Expression : Mult ( ( ('+'{Add.left=current} ) | ('-'{Minus.left=
       current} )) right=Mult) * ;
17 Mult returns Expression : In ( ( ('*'{Mult.left=current}) | ('/'{Div.left=current})
        | ('%'{Mod.left=current})) right=In) * ;
18 In returns Expression : Unary (('in' {In.left=current}) right=Unary) * ;
19 Unary returns Expression : Exponential | ('not' {Not} exp=Unary) | ('-' {UnaryMinus
       } expr=Unary) ;
20 Exponential returns Expression : Atom ('^' {Exponential.left=current} right=
       Exponential)? ;
21 Atom returns Expression : Literal | QualifiedName | ParsExpression ;
22 ParsExpression returns Expression : {ParsExpression} '(' expr+=Expression (',' expr
       +=Expression)* ')' ;
23 Literal : {StringLiteral} value=STRING | {FloatLiteral} value=FLOAT | {
       IntegerLiteral} value=INT | {Null} value='null' | {True} value='true' | {False}
       value='false' ;
24 QualifiedName returns QualifiedName : granular=('distinct'|'all')? qualifiers+=
```

```
    Qualified ('.' qualifiers+=Qualified)* ;
25  Qualified returns QualifiedElement : QualifiedElement ;
26  QualifiedElement returns QualifiedElement : Identifier | Function ;
27  Function returns Function : name=ID '(' params+=Expression (',' params+=Expression)
    * ')' ;
28  Identifier returns Identifier : id=ID ;
29  terminal ID returns ecore::EString : ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'
    ..'9'|'_')* ;
30  terminal INT returns ecore::EInt : '0'..'9'+ ;
31  terminal FLOAT returns ecore::EDouble : ( ('0'..'9')+ ( '.' ('0'..'9')+ ( ('E'|'e')
     ('-')? ('0'..'9')+ )? )? ) ;
32  terminal STRING : '"' ( '\\' ('b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\') | !('\\'|'"') )
    * '"' | "'" ( '\\' ('b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\') | !('\\'|"'") )* "'" ;
33  terminal ML_COMMENT : '/*' -> '*/' ;
34  terminal SL_COMMENT : '//' !('\n'|'\r')* ('\r'? '\n')? ;
35  terminal WS : (' '|'\t'|'\r'|'\n')+ ;
```

ANEXO B – SINTAXE ATUAL DE AQL EM XTEXT

```
1  Query hidden (WS, SL_COMMENT, ML_COMMENT) : query=QueryStatement ;
2  QueryStatement: (find=FindClause) (where=WhereClause)? (return=ReturnsClause) (
       orderby=OrderByClause)? (groupby=GroupByClause)? | (delete=DeleteClause) (where=
       WhereClause)? | (insert=InsertClause) | (update=UpdateClause) (where=WhereClause
       )? ;
3  FindClause : clause='find' bindingObject+=BindingObject (',' bindingObject+=
       BindingObject)* ;
4  BindingObject : type=ObjectType (alias+=ID)+ ;
5  ObjectType : {Project} value='project' | {Package} value='package' | {Class} value=
       'class' | {Aspect} value='aspect' | {Interface} value='interface' | {Enum} value
       ='enum' | {Method} value='method' | {Constructor} value='constructor' | {Foreach
       } value='foreach' | {While} value='while' | {If} value='if' | {Switch} value='
       switch' | {Expr} value='expression';
6  DeleteClause : clause='delete' bindingObject+=BindingObject (',' bindingObject+=
       BindingObject)* ;
7  UpdateClause : clause='update' bindingObject+=BindingObject '(' expressions+=
       Expression (',' expressions+=Expression)* ')' values=Values ;
8  InsertClause : clause='insert' bindingObject+=BindingObject '(' expressions+=
       Expression (',' expressions+=Expression)* ')' values=Values ;
9  Values : clause='values' '(' expressions+=(Expression | DMLObject) (',' expressions
       +=(Expression | DMLObject))* ')' ;
10 DMLObject : clause='object' option=('remove' | 'add')? bindingObject+=ObjectType
       ids+=(INT)+ ;
11 WhereClause : clause='where' expression=Expression ;
12 ReturnsClause : clause='returns' expressions+=Expression ('as' resultAlias+=ID)? (
       ',' expressions+=Expression ('as' resultAlias+=ID)?)* ;
13 GroupByClause : clause='group by' expressions+=SimpleQualifiedName (',' expressions
       +=SimpleQualifiedName)* ;
14 OrderByClause : clause='order by' expressions+=OrderByExpression (',' expressions+=
       OrderByExpression)* ('having' havingExpression=Expression)? ;
15 OrderByExpression : expr=SimpleQualifiedName option=('asc' | 'desc')? ;
16 SimpleQualifiedName returns QualifiedName : qualifiers+=SimpleQualified ('.'
       qualifiers+=SimpleQualified)* ;
17 SimpleQualified returns QualifiedElement : Identifier ;
18 Expression hidden (WS, SL_COMMENT, ML_COMMENT) : Or ;
19 Or returns Expression : And ( ('or'{Or.left=current}) right=And)* ;
20 And returns Expression : Relation ( ('and'{And.left=current} right=Relation))*;
21 Relation returns Expression : Add ( ( ('='{Equals.left=current}) | ('>'{Greater.
       left=current}) | ('>='{GreaterEqual.left=current}) | ('<'{Less.left=current}) |
       ('<='{LessEqual.left=current}) | ('like'{Like.left=current}) | ('!='{NotEquals.
       left=current} )) right=Add ) * ;
22 Add returns Expression : Mult ( ( ('+'{Add.left=current} ) | ('-'{Minus.left=
       current} )) right=Mult) * ;
```

```
23  Mult returns Expression : In ( ( ('*'{Mult.left=current}) | ('/'{Div.left=current})
      | ('%'{Mod.left=current})) right=In) * ;
24  In returns Expression : Unary (('in' {In.left=current}) right=Unary) * ;
25  Unary returns Expression : Exponential | ('not' {Not} exp=Unary) | ('-' {UnaryMinus
      } expr=Unary) ;
26  Exponential returns Expression : Atom ('^' {Exponential.left=current} right=
      Exponential)? ;
27  Atom returns Expression : Literal | QualifiedName | ParsExpression ;
28  ParsExpression returns Expression : {ParsExpression} '(' expr+=Expression (',' expr
      +=Expression)* ')' ;
29  Literal : {StringLiteral} value=STRING | {FloatLiteral} value=FLOAT | {
      IntegerLiteral} value=INT | {Null} value='null' | {True} value='true' | {False}
      value='false' ;
30  QualifiedName returns QualifiedName : granular=('distinct'|'all')? qualifiers+=
      Qualified ('.' qualifiers+=Qualified)* ;
31  Qualified returns QualifiedElement : QualifiedElement ;
32  QualifiedElement returns QualifiedElement : Identifier | Function ;
33  Function returns Function : name=ID '(' params+=Expression (',' params+=Expression)
      * ')' ;
34  Identifier returns Identifier : id=ID ;
35  terminal ID returns ecore::EString : ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'
      ..'9'|'_')* ;
36  terminal INT returns ecore::EInt : '0'..'9'+ ;
37  terminal FLOAT returns ecore::EDouble : ( ('0'..'9')+ ( '.' ('0'..'9')+ ( ('E'|'e')
      ('-')? ('0'..'9')+ )? )? ) ;
38  terminal STRING : '"' ( '\\' ('b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\') | !('\\'|'"') )
      * '"' | "'" ( '\\' ('b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\') | !('\\'|"'") )* "'" ;
39  terminal ML_COMMENT : '/*' -> '*/' ;
40  terminal SL_COMMENT : '//' !('\n'|'\r')* ('\r'? '\n')? ;
41  terminal WS : (' '|'\t'|'\r'|'\n')+ ;
```

ANEXO C – CLÁUSULAS GERADAS PELO COMPILADOR AQL

Consultas usadas na refatoração *Convert Enhanced For to Lambda Enhanced For*.

```
1  SELECT f FROM AOJEnhancedForStatement f WHERE f.isCollection = true
```

Listagem C.1: Consulta HQL que busca por laços de repetição `for each` que percorrem uma coleção

```
1  SELECT w FROM AOJWhileStatement w WHERE w.isCollection = true
```

Listagem C.2: Consulta em HQL que busca por laços de repetição `while` que percorrem uma coleção

```
1  AOJProgramElement obj0 = (AOJProgramElement) session.createQuery("FROM
       AOJMethodDeclaration m WHERE id in (101)").list().iterator().next();
2  AOJExpressionStatement element = new AOJExpressionStatement(null, obj0, 1);
3  element.setCodeBody("nomes.forEach(nome -> System.out.println(nome));");
4  element.setCodeOrder(296);
5  AOJMethodDeclaration owner0 = (AOJMethodDeclaration) obj0;
6  owner0.addStatements(element);
7  element.loadByAQL();
8  session.save(element);
```

Listagem C.3: Instruções Java e Hibernate que adicionam uma expressão lambda

```
1  updateHQL("DELETE FROM AOJEnhancedForStatement f WHERE f.id IN ("+getAllIds(session
       .createQuery("SELECT f FROM AOJEnhancedForStatement f WHERE f.id = 103").list())
       +")");
```

Listagem C.4: Instruções Java e Hibernate que removem o laço de repetição `for each`

Consultas usadas na refatoração *Convert Collections.sort to sort*.

```
1  SELECT e FROM AOJExpressionStatement e WHERE e.codeBody like 'Collections.sort%'
```

Listagem C.5: Consulta HQL que busca por expressões que chamam o método `Collections.sort`

```
1  AOJProgramElement obj0 = (AOJProgramElement) session.createQuery("FROM
       AOJMethodDeclaration m WHERE id in (122)").list().iterator().next();
2  AOJExpressionStatement element = new AOJExpressionStatement(null, obj0, 1);
3  element.setCodeBody("((List<String>) nomes).sort((p1, p2) -> p1.compareTo(p2));");
4  element.setCodeOrder(361);
5  AOJMethodDeclaration owner0 = (AOJMethodDeclaration) obj0;
6  owner0.addStatements(element);
7  element.loadByAQL();
```

```
8  session.save(element);
```

Listagem C.6: Instruções Java e Hibernate que adicionam uma expressão lambda no método `ordenaNomes`

```
1  updateHQL("DELETE FROM AOJExpressionStatement e WHERE e.id IN (" + getAllIds(
       session.createQuery("SELECT e FROM AOJExpressionStatement e WHERE e.id = 132").
       list()) + ")");
```

Listagem C.7: Instruções Java e Hibernate que removem a antiga implementação do método `Collections.sort`

Consultas usadas na refatoração *Convert Enhanced For with If to Lambda Filter*.

```
1  SELECT aojowner_1 FROM AOJIfStatement e LEFT JOIN e.owner aojowner_1 WHERE
       aojowner_1.metaName = 'Foreach' AND aojowner_1.isCollection = true
```

Listagem C.8: Consulta HQL que busca por laços de repetição `for each` que implementam um filtro através de um condicional

```
1  AOJProgramElement obj0 = (AOJProgramElement) session.createQuery("FROM
       AOJMethodDeclaration m WHERE id in (160)").list().iterator().next();
2  AOJExpressionStatement element = new AOJExpressionStatement(null, obj0, 1);
3  element.setCodeBody("nomes.stream().filter(nome -> nome.startsWith(inicio)).forEach
       (nome -> System.out.println(nome));");
4  element.setCodeOrder(456);
5  AOJMethodDeclaration owner0 = (AOJMethodDeclaration) obj0;
6  owner0.addStatements(element);
7  element.loadByAQL();
8  session.save(element);
```

Listagem C.9: Instruções Java e Hibernate que adicionam uma expressão lambda utilizando um filtro

```
1  updateHQL("DELETE FROM AOJEnhancedForStatement f WHERE f.id IN (" +getAllIds(
       session.createQuery("SELECT f FROM AOJEnhancedForStatement f WHERE f.id = 164").
       list()) + ")");
```

Listagem C.10: Instruções Java e Hibernate que o laço de repetição `for each`

Consultas usadas na refatoração *Convert Abstract Interface Method to Default Method*.

```
1  SELECT m FROM AOJMethodDeclaration m LEFT JOIN m.owner aojowner_1 WHERE aojowner_1.
       metaName = 'Class' and aojowner_1.implementsInterface = true AND ((m.
       statementsBody < 1) OR (m.statementsBody <= 1 and m.returnTypeName <> 'void'))
       AND m.hasAnnotation = true
```

Listagem C.11: Consulta HQL que busca por métodos vazios em classes que implementam pelo menos uma interface

```
1 SELECT m FROM AOJMethodDeclaration m LEFT JOIN m.owner aojowner_1 WHERE aojowner_1.
    name = 'Mostrar' and m.name = 'contar'
```

Listagem C.12: Consulta HQL que busca pelo método `contar` na interface `Mostrar`

```
1 List<AOJMethodDeclaration> elements = session.createQuery("SELECT e FROM
    AOJMethodDeclaration e WHERE e.id = 174").list();
2 for (AOJMethodDeclaration element: elements) {
3     element.setSignature("public int default contar()");
4     session.update(element);
5 }
```

Listagem C.13: Instruções Java e Hibernate que atualizam a assinatura do método `contar` da interface `Mostrar`

```
1 AOJProgramElement obj0 = (AOJProgramElement) session.createQuery("FROM
    AOJMethodDeclaration m WHERE id in (174)").list().iterator().next();
2 AOJExpressionStatement element = new AOJExpressionStatement(null, obj0, 1);
3 element.setCodeBody("return 0;");
4 element.setCodeOrder(1);
5 AOJMethodDeclaration owner0 = (AOJMethodDeclaration) obj0;
6 owner0.addStatements(element);
7 element.loadByAQL();
8 session.save(element);
```

Listagem C.14: Instruções Java e Hibernate que adiciona uma implementação `default` para o método `contar`

```
1 updateHQL("DELETE FROM AOJMethodDeclaration m WHERE m.id IN ("
2 +getAllIds(session.createQuery("SELECT m FROM AOJMethodDeclaration m WHERE m.id =
    16").list()) +
3 ")");
```

Listagem C.15: Instruções Java e Hibernate que removem o método `contar` da classe `Cliente`

Consultas usadas na refatoração *Chain Constructors*.

```
1 SELECT aojowner_1 FROM AOJConstructorDeclaration c LEFT JOIN c.owner aojowner_1
    WHERE c.statementsBody > 2 GROUP BY aojowner_1.id HAVING count(aojowner_1.id) >
    2
```

Listagem C.16: Consulta HQL que busca por oportunidades da refatoração *Chain Constructors*

```
1 updateHQL("DELETE FROM AOJExpressionStatement e WHERE e.id IN (" + getAllIds(
    session.createQuery("SELECT e FROM AOJExpressionStatement e LEFT JOIN e.owner
```

```
    owner_1 WHERE owner_1.id = 222").list()) + ")");
```

Listagem C.17: Instruções Java e Hibernate que removem as instruções do método construtor

```
1  AOJProgramElement obj0 = (AOJProgramElement) session.createQuery("FROM
       AOJConstructorDeclaration c WHERE id in (222)").list().iterator().next();
2  AOJExpressionStatement element = new AOJExpressionStatement(null, obj0, 1);
3  element.setCodeBody("this(descricao, valorTotal, 0, inicio);");
4  element.setCodeOrder(5);
5  AOJConstructorDeclaration onwed0 = (AOJConstructorDeclaration) obj0;
6  onwed0.addStatements(element);
7  element.loadByAQL();
8  session.save(element);
```

Listagem C.18: Instruções Java e Hibernate que adicionam a chamada ao construtor mais genérico

```
1  updateHQL("DELETE FROM AOJExpressionStatement e WHERE e.id IN (" +
2  getAllIds(session.createQuery("SELECT e FROM AOJExpressionStatement e LEFT JOIN e.
       owner aojowner_1 WHERE aojowner_1.id = 235").list())
3  + ")");
```

Listagem C.19: Instruções Java e Hibernate que removem as instruções do método construtor

```
1  AOJProgramElement obj0 = (AOJProgramElement) session.createQuery("FROM
       AOJConstructorDeclaration c WHERE id in (235)").list().iterator().next();
2  AOJExpressionStatement element = new AOJExpressionStatement(null, obj0, 1);
3  element.setCodeBody("this(descricao, valorTotal, valorPago, 0, inicio, null);");
4  element.setCodeOrder(5);
5  AOJConstructorDeclaration owner0 = (AOJConstructorDeclaration) obj0;
6  owner0.addStatements(element);
7  element.loadByAQL();
8  session.save(element);
```

Listagem C.20: Instruções Java e Hibernate que adicionam a chamada ao construtor principal

ANEXO D – CLASSES MODIFICADAS DO COMPILADOR AQL

Classe `HqlOwnerFunction`.

```
1  class HqlOwnerFunction extends HqlCommonTransformation {
2     private extension HqlASTTool hqlASTTool = HqlTransformationFactory::eINSTANCE.
          createHqlASTTool
3     override void internalTransform (hql.QualifiedElement function, Qualified
          resolved, hql.Query ast) {
4        val hql.ObjectSource rightSide = HqlFactory::eINSTANCE.createObjectSource
5        rightSide.setClassName(Qualified::qualify(objectSource.alias).add(
            PROPERTYHQLOWNER).toFullString)
6        rightSide.setAlias(ALIASOWNER.getAlias)
7        rightSide.setResolvedPath(resolved.toFullString)
8        rightSide.addToAliasList
9        var hql.FromExpression leftSide = ast.queryStatement.fromClause.
            getLastFromExpression
10       val hql.LeftJoin join = hqlASTTool.createLeftJoin(leftSide, rightSide)
11       val String oldAlias = Qualified::qualify(qualifiedObject).add(
            FUNCTIONOWNERQUALIFIER).toFullString
12       ast.replaceAlias(oldAlias, rightSide.alias)
13    }
14 }
```

Classe `HqlStatmentsFunction`.

```
1  class HqlStatmentsFunction extends HqlCommonTransformation {
2     private extension HqlASTTool hqlASTTool = HqlTransformationFactory::eINSTANCE.
          createHqlASTTool
3     override void internalTransform (hql.QualifiedElement function, Qualified
          resolved, hql.Query ast) {
4        val hql.ObjectSource rightSide = HqlFactory::eINSTANCE.createObjectSource
5        rightSide.setClassName(Qualified::qualify(objectSource.alias).add(
            PROPERTYHQLSTATMENTS).toFullString)
6        rightSide.setAlias(ALIASSTATMENTS.getAlias)
7        rightSide.setResolvedPath(resolved.toFullString)
8        rightSide.addToAliasList
9        var hql.FromExpression leftSide = ast.queryStatement.fromClause.
            getLastFromExpression
10       val hql.LeftJoin join = hqlASTTool.createLeftJoin(leftSide, rightSide)
11       val String oldAlias = Qualified::qualify(qualifiedObject).add(
            FUNCTIONSTATMENTSQUALIFIER).toFullString
12       ast.replaceAlias(oldAlias, rightSide.alias)
13    }
14 }
```

Classe `HqlTransformationPass1Impl`.

```
1  class HqlTransformationPass1Impl extends CommonTransformationImpl implements
      HqlTransformation1 {
2     @Property private hql.Query hqlASTRoot
3     @Property private hql.Clause currentClause
4     private extension ASTPrinter printer
5     private extension HqlParamFunctionResolver paramResolver
6     private Setting settings = Setting::getINSTANCE(Setting::defaulURI)
7     @Property private HqlExpressionCompiler expressionCompiler;
8     override registerTransformation (Resource resource, IFileSystemAccess fsa) {
9        fromResource = resource
10       this.fsa = fsa
11       checkResources
12    }
13    new() {
14       expressionCompiler = new HqlExpressionCompilerImpl();
15       paramResolver = new HqlParamFunctionResolver();
16    }
17    def void checkResources () {
18       if ( (null == fromResource) || (null == fsa) )
19       throw new IllegalStateException ("No Source resource or File System Access
            was found ! Null state found")
20    }
21    override run() {
22       preProcessing
23       checkResources
24       val aqlAST = fromResource.contents
25       aqlASTRoot = aqlAST.get(0) as br.ufsm.aql.Query
26       aqlASTRoot.compile
27       phase2
28       postProcessing
29    }
30
31    def dispatch void compile (br.ufsm.aql.Query aqlQuery) {
32       print("\nCompiling aql query - Phase 1...")
33       createHqlASTRootNode
34       hqlASTRoot.setQueryStatement(createHqlQueryStatement)
35       if (null != aqlASTRoot.query.find)
36       aqlASTRoot.query.find.compile
37       if (null != aqlASTRoot.query.delete)
38       aqlASTRoot.query.delete.compile
39       if (null != aqlASTRoot.query.update)
40       aqlASTRoot.query.update.compile
41       if (null != aqlASTRoot.query.insert)
42       aqlASTRoot.query.insert.compile
43       if (null != aqlASTRoot.query.where)
44       aqlASTRoot.query.where.compile
45       if (null != aqlASTRoot.query.^return)
```

```
46          aqlASTRoot.query.^return.compile
47          if (null != aqlASTRoot.query.orderby)
48          aqlASTRoot.query.orderby.compile
49          print("\nAST - HQL - Fase 1")
50          checkNodesNull; println;
51          hqlASTRoot.queryStatement.selectClause.printAST('hql'); println;
52          hqlASTRoot.queryStatement.deleteClause.printAST('hql'); println;
53          hqlASTRoot.queryStatement.updateClause.printAST('hql'); println;
54          hqlASTRoot.queryStatement.insertClause.printAST('hql'); println;
55          hqlASTRoot.queryStatement.fromClause.printAST('hql'); println;
56          hqlASTRoot.queryStatement.whereClause.printAST('hql');
57      }
58      def void phase2() {
59          val HqlTransformation2 hqlT2 = HqlTransformationFactory::eINSTANCE.
                createHqlTransformation2
60          hqlT2.registerTransformation(hqlASTRoot, aqlASTRoot)
61          hqlT2.run
62          hqlASTRoot = hqlT2.hqlASTRoot
63      }
64      def void preProcessing() {
65          HqlFromAlias::sourceList.clear
66          HqlAliasHandler::functionCounterMap.clear
67      }
68      def void postProcessing() {
69          cleanup
70      }
71      def void cleanup() {
72          cleanupEmptyWhere
73          debug()
74      }
75      def void debug() {
76          println("\nAST - AQL")
77          aqlASTRoot.query.printAST ('aql')
78          print("\nAST - HQL")
79          checkNodesNull
80          println
81          hqlASTRoot.queryStatement.selectClause.printAST('hql')
82          println
83          hqlASTRoot.queryStatement.deleteClause.printAST('hql')
84          println
85          hqlASTRoot.queryStatement.updateClause.printAST('hql')
86          println
87          hqlASTRoot.queryStatement.insertClause.printAST('hql')
88          println
89          hqlASTRoot.queryStatement.fromClause.printAST('hql')
90          println
91          hqlASTRoot.queryStatement.whereClause.printAST('hql')
```

```
92      }
93      def void checkNodesNull () {
94          hqlASTRoot.queryStatement.selectClause.printNodesNull
95          hqlASTRoot.queryStatement.deleteClause.printNodesNull
96          hqlASTRoot.queryStatement.updateClause.printNodesNull
97          hqlASTRoot.queryStatement.insertClause.printNodesNull
98          hqlASTRoot.queryStatement.fromClause.printNodesNull
99          hqlASTRoot.queryStatement.whereClause.printNodesNull
100     }
101     def void cleanupEmptyWhere() {
102         if (null != hqlASTRoot.queryStatement.whereClause)
103             if (hqlASTRoot.queryStatement.whereClause.eContents.empty)
104                 hqlASTRoot.queryStatement.whereClause = null
105     }
106     def dispatch void compile(br.ufsm.aql.ReturnsClause clause) {
107         hqlASTRoot.queryStatement.setSelectClause(createSelectClause)
108         for (br.ufsm.aql.Expression e : clause.expressions)
109             hqlASTRoot.queryStatement.selectClause.expressions.add(expressionCompiler
                    .compile(e))
110         for (String alias : clause.resultAlias)
111             hqlASTRoot.queryStatement.selectClause.alias.add(alias)
112     }
113     def dispatch void compile(br.ufsm.aql.WhereClause clause) {
114         hqlASTRoot.queryStatement.setWhereClause(createWhereClause)
115         hqlASTRoot.queryStatement.whereClause.setExpression(expressionCompiler.
                compile(clause.expression))
116     }
117     def dispatch void compile(br.ufsm.aql.FindClause clause) {
118         hqlASTRoot.queryStatement.setFromClause(createFromClause)
119         currentClause = hqlASTRoot.queryStatement.fromClause
120         clause.bindingObject.compile
121     }
122     def dispatch void compile(br.ufsm.aql.DeleteClause clause) {
123         if (hqlASTRoot.queryStatement.getFromClause == null)
124             hqlASTRoot.queryStatement.setFromClause(createFromClause)
125         hqlASTRoot.queryStatement.setDeleteClause(createDeleteClause)
126         currentClause = hqlASTRoot.queryStatement.deleteClause
127         clause.bindingObject.compile
128     }
129     def dispatch void compile(br.ufsm.aql.UpdateClause clause) {
130         if (hqlASTRoot.queryStatement.getFromClause == null)
131             hqlASTRoot.queryStatement.setFromClause(createFromClause)
132         hqlASTRoot.queryStatement.setUpdateClause(createUpdateClause)
133         currentClause = hqlASTRoot.queryStatement.updateClause
134         clause.bindingObject.compile
135         for (br.ufsm.aql.Expression e : clause.expressions)
136             hqlASTRoot.queryStatement.updateClause.expressions.add(expressionCompiler
```

```
                .compile(e))
137         clause.values.compile
138     }
139     def dispatch void compile(br.ufsm.aql.InsertClause clause) {
140         if (hqlASTRoot.queryStatement.getFromClause == null)
141             hqlASTRoot.queryStatement.setFromClause(createFromClause)
142         hqlASTRoot.queryStatement.setInsertClause(createInsertClause)
143         currentClause = hqlASTRoot.queryStatement.insertClause
144         clause.bindingObject.compile
145         for (br.ufsm.aql.Expression e : clause.expressions)
146             hqlASTRoot.queryStatement.insertClause.expressions.add(expressionCompiler
                .compile(e))
147         clause.values.compile
148     }
149     def dispatch void compile(br.ufsm.aql.Values clause) {
150         for (var i = 0; i < clause.expressions.length; i++) {
151             var e = clause.expressions.get(i);
152             if (e instanceof br.ufsm.aql.Expression){
153                 if (null != hqlASTRoot.queryStatement.insertClause)
154                     hqlASTRoot.queryStatement.insertClause.values.add(
                        expressionCompiler.compile(e))
155                 else if (null != hqlASTRoot.queryStatement.updateClause)
156                     hqlASTRoot.queryStatement.updateClause.values.add(
                        expressionCompiler.compile(e))
157             }
158             if(e instanceof br.ufsm.aql.DMLObject){
159                 var dmlObject = HqlFactory::eINSTANCE.createDMLObject
160                 dmlObject.type = "object"
161                 val String _type = settings.resolve(e.bindingObject.get(0).value)
162                 if (null == _type)
163                     throw new IllegalArgumentException ("Illegal Bind Object type " +
                         e.bindingObject.get(0).eClass)
164                 dmlObject.object = _type
165                 if(null != e.option)
166                 dmlObject.option = e.option
167                 else
168                 dmlObject.option = ''
169                 dmlObject.ids = e.ids.toString.replace('[', '').replace(']', '')
170                 if (null != hqlASTRoot.queryStatement.insertClause)
171                     hqlASTRoot.queryStatement.insertClause.values.add(dmlObject)
172                 else if (null != hqlASTRoot.queryStatement.updateClause)
173                     hqlASTRoot.queryStatement.updateClause.values.add(dmlObject)
174             }
175         }
176     }
177     def dispatch void compile(br.ufsm.aql.OrderByClause clause) {
178         hqlASTRoot.queryStatement.setOrderByClause(createOrderByClause)
```

```
179         for (br.ufsm.aql.OrderByExpression e : clause.expressions) {
180             var orderByExpression = HqlFactory::eINSTANCE.createOrderByExpression
181             val char lOption = e.option.optionAsChar
182             orderByExpression.setOption(lOption)
183             orderByExpression.expressions.add(compile(e.expr))
184             hqlASTRoot.queryStatement.orderByClause.expressions.add(orderByExpression
                    )
185         }
186 
187     }
188     def char optionAsChar (String arg) {
189         if (null != arg)
190             return arg.charAt(0)
191         return ' '.charAt(0)
192     }
193     def dispatch void compile (EList<br.ufsm.aql.BindingObject> bindingObject) {
194         for (br.ufsm.aql.BindingObject bind: bindingObject)
195             for (alias : bind.alias)
196                 hqlASTRoot.queryStatement.fromClause.objectList.add(createFromObject
                        (bind.type, alias))
197         if (hqlASTRoot.queryStatement.fromClause.objectList.size > 1) {
198             val hql.ThetaJoin join = hqlASTRoot.queryStatement.fromClause.objectList.
                    transformToThetaJoin
199             hqlASTRoot.queryStatement.fromClause.objectList.clear
200             hqlASTRoot.queryStatement.fromClause.binaryObjectList.add(join)
201         }
202     }
203     def hql.ThetaJoin transformToThetaJoin (EList<hql.FromExpression> expressions) {
204         val hql.ThetaJoin join = HqlFactory::eINSTANCE.createThetaJoin
205         join.setLeft(expressions.get(0))
206         join.setRight(join.createThetaChildren(expressions, 0))
207         return join
208     }
209     def hql.FromExpression createThetaChildren (hql.ThetaJoin parent, EList<hql.
            FromExpression> expressions, int index) {
210         if (index > expressions.size)
211             throw new IndexOutOfBoundsException ("Index parameters is out of
                    expression.size when creating Theta Join")
212         if (index == expressions.size - 1)
213             return expressions.get(index)
214         val hql.ThetaJoin join = HqlFactory::eINSTANCE.createThetaJoin
215         parent.setLeft(expressions.get(index))
216         parent.setRight(join.createThetaChildren(expressions, 0))
217         return join
218     }
219     def hql.ObjectSource createFromObject(br.ufsm.aql.ObjectType type, String alias)
             {
```

```
220        val hql.ObjectSource objectSource = HqlFactory::eINSTANCE.createObjectSource
221        objectSource.className = type.translateTypeName
222        objectSource.alias = alias
223        objectSource.setResolvedPath(objectSource.className)
224        objectSource.addAtSymbolsIfDoesNotExist
225        return objectSource
226    }
227    def String translateTypeName (br.ufsm.aql.ObjectType type) {
228        val String _type = settings.resolve(type.value)
229        if (null == _type)
230        throw new IllegalArgumentException ("Illegal Bind Object type " + type.
               eClass)
231        return _type
232    }
233    def createSelectClause() {
234        val hql.SelectClause hqlSelect = HqlFactory::eINSTANCE.createSelectClause
235        hqlSelect.clause = "Select"
236        return hqlSelect
237    }
238    def createDeleteClause() {
239        val hql.DeleteClause hqlDelete = HqlFactory::eINSTANCE.createDeleteClause
240        hqlDelete.clause = "Delete"
241        return hqlDelete
242    }
243    def createUpdateClause() {
244        val hql.UpdateClause hqlUpdate = HqlFactory::eINSTANCE.createUpdateClause
245        hqlUpdate.clause = "Update"
246        return hqlUpdate
247    }
248    def createInsertClause() {
249        val hql.InsertClause hqlInsert = HqlFactory::eINSTANCE.createInsertClause
250        hqlInsert.clause = "Insert into"
251        return hqlInsert
252    }
253    def createFromClause() {
254        val hql.FromClause hqlFrom = HqlFactory::eINSTANCE.createFromClause
255        hqlFrom.clause = "From"
256        return hqlFrom
257    }
258    def createWhereClause() {
259        val hql.WhereClause hqlWhere = HqlFactory::eINSTANCE.createWhereClause
260        hqlWhere.clause = "Where"
261        return hqlWhere
262    }
263    def createOrderByClause() {
264        val hql.OrderByClause hqlOrderBy = HqlFactory::eINSTANCE.createOrderByClause
265        hqlOrderBy.clause = "Order By"
```

```
266         return hqlOrderBy
267     }
268     def createDMLObject() {
269         val hql.DMLObject dml = HqlFactory::eINSTANCE.createDMLObject
270         return dml
271     }
272     def void createHqlASTRootNode() {
273         hqlASTRoot = HqlFactory::eINSTANCE.createQuery
274     }
275     def createHqlQueryStatement() {
276         return HqlFactory::eINSTANCE.createQueryStatement
277     }
278     override void setup() {
279         hqlASTRoot = null
280     }
281     override getASTRoot() {
282         return hqlASTRoot
283     }
284     def void printAST (EObject node, String ASTKind) {
285         if (null != node) {
286             printer = new ASTPrinter(ASTKind)
287             node.printTree
288         }
289     }
290     def void printNodesNull (EObject node) { }
291 }
```

Classe `AQLGenerator`.

```
1  class AQLGenerator implements IGenerator {
2      var int i = 0;
3      val log = Logger::getLogger(getClass());
4      val Map<Integer,String> myObjects = new HashMap();
5      private HqlTransformation1 hqlT1 = HqlTransformationFactory::eINSTANCE.
           createHqlTransformation1
6      override void doGenerate(Resource resource, IFileSystemAccess fsa) {
7          println("Generating hql statements...")
8          hqlT1.registerTransformation(resource, fsa)
9          hqlT1.run
10         var hqlQuery = ""
11         if (hqlT1.hqlASTRoot.queryStatement.insertClause == null && hqlT1.hqlASTRoot
               .queryStatement.updateClause == null){
12             hqlQuery = hqlT1.hqlASTRoot.queryStatement.compile.toString.replaceAll("\
                   n", " ")
13         } else
14             hqlQuery = hqlT1.hqlASTRoot.queryStatement.compile.toString
15         println("\n********************")
16         println("hql> " + hqlQuery)
```

```
17        println("********************")
18        fsa.generateFile("query.hql", hqlQuery)
19    }
20    def dispatch compile(QueryStatement query) ''
21    «IF (query.deleteClause != null)»
22        «this.compile(query.deleteClause, query)»
23    «ELSE»
24        «IF (query.updateClause != null)»
25            «this.compile(query.updateClause, query)»
26            «ELSE»
27                «IF (query.insertClause != null)»
28                    «this.compile(query.insertClause, query)»
29                «ELSE»
30                «IF (query.selectClause != null)»
31                    «query.selectClause.compile»
32                «ENDIF»
33                «IF (query.fromClause != null)»
34                    «query.fromClause.compile»
35                «ENDIF»
36                «IF (query.whereClause != null)»
37                    «query.whereClause.compile»
38                «ENDIF»
39                «IF (query.orderByClause != null)»
40                    «query.orderByClause.compile»
41                «ENDIF»
42            «ENDIF»
43        «ENDIF»
44    «ENDIF»
45    ''
46    def dispatch compile(WhereClause where) ''
47        «where.clause.toUpperCase»
48        «where.expression.compile»
49    ''
50    def ObjectSource toHqlObjectSource(QueryStatement query) {
51        var ObjectSource elementDelete = null
52        if (query.fromClause != null)
53            if (query.fromClause.binaryObjectList.size > 0)
54                elementDelete = query.fromClause.binaryObjectList.get(0).left as
                      ObjectSource
55            if (query.fromClause.objectList.size > 0)
56                elementDelete = query.fromClause.objectList.get(0) as ObjectSource
57        return elementDelete;
58    }
59    def void addObjectNames(int i){
60        myObjects.put(i, "obj" + myObjects.size);
61    }
62    def String getParamModification(UpdateClause update, int i){
```

```
63          if(update.values.get(i) instanceof DMLObject){
64              val obj = update.values.get(i) as DMLObject
65              if (!obj.option.empty)
66                  return obj.option;
67          }
68          return "set";
69      }
70      def String getObjectNames(int i){
71          return myObjects.get(i);
72      }
73      def int findOwner(InsertClause insert){
74          for ( i : 0..insert.expressions.length) {
75              if (insert.expressions.get(i) instanceof QualifiedName){
76                  var parametro = insert.expressions.get(i) as QualifiedName
77                  if ( parametro.qualifiers.get(0) instanceof Identifier){
78                      var ident = parametro.qualifiers.get(0) as Identifier
79                      if (ident.id.toLowerCase.equals("owner"))
80                          return i;
81                  }
82              }
83          }
84          return -1;
85      }
86      def dispatch compile(DeleteClause delete, QueryStatement query) ''
87          «val ObjectSource elementDelete = toHqlObjectSource(query)»
88          updateHQL("«delete.clause.toUpperCase» FROM
89          «elementDelete.className» «elementDelete.alias»
90          WHERE «elementDelete.alias».id IN (
91              "+getAllIds(session.createQuery("SELECT «elementDelete.alias» «
                    dmlSubQuery(elementDelete, query)»").list())
92          +")");
93      ''
94      def dispatch compile(UpdateClause update, QueryStatement query) ''
95          «val ObjectSource elementUpdate = toHqlObjectSource(query)»
96          «update.values.searchObjects»
97          List«<elementUpdate.className»> elements = session.createQuery("«dmlSubQuery
                (elementUpdate, query).toString»").list();
98          for («elementUpdate.className» element : elements) {
99              «setUpdateElement(update, query)»
100             session.update(element);
101         }
102     ''
103     def dispatch setUpdateElement(UpdateClause update, QueryStatement query)''
104         «FOR i : 0..<update.expressions.length»
105             «IF (update.expressions.get(i) instanceof QualifiedName)»
106             «var parametro = update.expressions.get(i) as QualifiedName»
107                 «IF (parametro.qualifiers.get(0) instanceof Identifier) »
```

```
108                 «var ident = parametro.qualifiers.get(0) as Identifier»
109                 element.«getParamModification(update, i)»«ident.id.charAt(0).
                        toString.toUpperCase.charAt(0)»«ident.id.substring(1)»(«IF (
                        update.values.get(i) instanceof DMLObject)»«getObjectNames(i)»
                        «ELSE»«update.values.get(i).compilej»«ENDIF»);
110             «ENDIF»
111         «ENDIF»
112     «ENDFOR»
113 ''
114 def dispatch searchObjects(EList<Expression> list)''
115     «FOR i : 0..<list.length»
116         «IF (list.get(i) instanceof DMLObject)»
117             «var obj = list.get(i) as DMLObject»
118             «addObjectNames(i)»
119             «IF obj.ids.contains(',') »
120                 List<AOJProgramElement> «getObjectNames(i)» = (AOJProgramElement)
                         session.createQuery("FROM «obj.object» m WHERE id IN («obj.
                        ids»)").list();
121             «ELSE»
122                 AOJProgramElement «getObjectNames(i)» = (AOJProgramElement)
                        session.createQuery("FROM «obj.object» m WHERE id IN («obj.ids
                        »)").list().iterator().next();
123             «ENDIF»
124         «ENDIF»
125     «ENDFOR»
126 ''
127 def dispatch compile(InsertClause insert, QueryStatement query) ''
128     «insert.values.searchObjects»
129     «val ObjectSource elementInsert = toHqlObjectSource(query)»
130     «var owner = insert.findOwner»
131     «elementInsert.className» element = new «elementInsert.className»(null, «
            getObjectNames(owner)»);
132     «setAttributes(insert)»
133     element.loadByAQL();
134     session.save(element);
135 ''
136 def dispatch setAttributes(InsertClause insert) ''
137     «var owner = insert.findOwner»
138     «FOR i : 0..<insert.expressions.length»
139         «IF (insert.expressions.get(i) instanceof QualifiedName)»
140             «var parametro = insert.expressions.get(i) as QualifiedName»
141             «IF ( parametro.qualifiers.get(0) instanceof Identifier) »
142                 «var ident = parametro.qualifiers.get(0) as Identifier»
143                 «IF !ident.id.toLowerCase.equals("owner") »
144                     element.set«ident.id.charAt(0).toString.toUpperCase.charAt(0)»
                            «ident.id.substring(1)»(«IF (insert.values.get(i)
                            instanceof DMLObject)»«getObjectNames(i)»«ELSE»«insert.
```

```
                          values.get(i).compilej»«ENDIF»);
145                  «ELSE»
146                     «var obj = insert.values.get(i) as DMLObject»
147                     «obj.object» onwer«i» = («obj.object») «getObjectNames(i)»;
148                     owner«i».addStatements(element);
149                  «ENDIF»
150               «ENDIF»
151            «ENDIF»
152         «ENDFOR»
153      ''
154      def dispatch compile(DMLObject dmlObject) ''
155         «dmlObject.type.toUpperCase»
156         «dmlObject.object.toString»
157         «dmlObject.ids.toString»
158      ''
159      def dispatch dmlSubQuery(ObjectSource element, QueryStatement query) ''
160         «IF (query.fromClause != null)»
161            «query.fromClause.compile»
162         «ENDIF»
163         «IF (query.whereClause != null)»
164            «query.whereClause.compile»
165         «ENDIF»
166      ''
167      def dispatch compile(OrderByClause orderBy) ''
168         «orderBy.clause.toUpperCase»
169         «FOR e : orderBy.expressions SEPARATOR ','»
170            «e.expression.compile»
171            «IF e.expression.option.toString.equals("a") »
172               ASC
173            «ELSEIF e.expression.option.toString.equals("d")»
174               DESC
175            «ENDIF»
176         «ENDFOR»
177      ''
178      def dispatch compile(SelectClause select) ''
179         «select.clause.toUpperCase»
180         «clear_i»
181         «FOR b : select.expressions SEPARATOR ','»
182            «b.compile»
183            «IF (i < select.alias.size)»
184               as «select.alias.get(i)»
185               «inc_i»
186            «ENDIF»
187         «ENDFOR»
188      ''
189      def void clear_i() { i = 0 }
190      def void inc_i() { i = i + 1 }
```

```
191    def dispatch compile(ObjectSource objectSource) '' «objectSource.className» «
           objectSource.alias» ''
192    def dispatch compile(String label) '' «label» ''
193    def dispatch compile(QualifiedName qualifiedName) ''
194        «IF qualifiedName.granular.length > 0»
195            «qualifiedName.granular »
196        «ENDIF»
197        «FOR e : qualifiedName.qualifiers SEPARATOR '.'»
198            «e.compile»
199        «ENDFOR»
200    ''
201    def dispatch compile(Identifier identifier) '' «identifier.id» ''
202    def dispatch compile(Function function) ''
203        «function.name» ( «FOR p : function.params SEPARATOR ','» «p.compile» «
               ENDFOR» )
204    ''
205    def dispatch compile(FromClause from) ''
206    «from.clause.toUpperCase»
207        «FOR b : from.objectList SEPARATOR ','»
208            «b.compile»
209        «ENDFOR»
210        «FOR b : from.binaryObjectList»
211            «b.compile»
212        «ENDFOR»
213    ''
214    def dispatch compile(LeftJoin expression) ''
215        «IF expression.left != null»
216            «expression.left.compile»
217        «ENDIF»
218        LEFT JOIN
219        «IF expression.right != null»
220            «expression.right.compile»
221        «ENDIF»
222    ''
223    def dispatch compile(InnerJoin expression) ''
224        «IF expression.left != null»
225            «expression.left.compile»
226        «ENDIF»
227        INNER JOIN
228        «IF expression.right != null»
229            «expression.right.compile»
230        «ENDIF»
231    ''
232    def dispatch compile(ThetaJoin expression) ''
233        «IF expression.left != null»
234            «expression.left.compile»
235        «ENDIF»,
```

```
236         «IF expression.right != null»
237            «expression.right.compile»
238         «ENDIF»
239      ''
240      def dispatch compile(InClause operation) '' «operation.left.compile» IN «
             operation.right.compile» ''
241      def dispatch compile(Equals operation) '' «operation.left.compile» = «operation.
             right.compile» ''
242      def dispatch compile(NotEquals operation) '' «operation.left.compile» <> «
             operation.right.compile» ''
243      def dispatch compile(Less operation) '' «operation.left.compile» < «operation.
             right.compile» ''
244      def dispatch compile(LessEqual operation) '' «operation.left.compile» <= «
             operation.right.compile» ''
245      def dispatch compile(Greater operation) '' «operation.left.compile» > «operation
             .right.compile» ''
246      def dispatch compile(GreaterEqual operation) '' «operation.left.compile» >= «
             operation.right.compile» ''
247      def dispatch compile(Like operation) '' «operation.left.compile» LIKE «operation
             .right.compile» ''
248      def dispatch compile(Add operation) '' «operation.left.compile» + «operation.
             right.compile» ''
249      def dispatch compile(Minus operation) '' «operation.left.compile» - «operation.
             right.compile» ''
250      def dispatch compile(Div operation) '' «operation.left.compile» / «operation.
             right.compile» ''
251      def dispatch compile(Mult operation) '' «operation.left.compile» * «operation.
             right.compile» ''
252      def dispatch compile(Mod operation) '' «operation.left.compile» % «operation.
             right.compile» ''
253      def dispatch compile(UnaryMinus operation) '' - «operation.expression.compile» '
             '
254      def dispatch compile(And operation) '' «operation.left.compile» AND «operation.
             right.compile» ''
255      def dispatch compile(Or operation) ''  «operation.left.compile» OR «operation.
             right.compile» ''
256      def dispatch compile(Not operation) '' NOT «operation.expression.compile» ''
257      def dispatch compile(IntegerLiteral literal) '' «literal.value» ''
258      def dispatch compile(FloatLiteral literal) ''  «literal.value» ''
259      def dispatch compile(StringLiteral literal) '' '«literal.value»' ''
260      def dispatch compile(Null literal) '' null ''
261      def dispatch compile(True literal) '' true ''
262      def dispatch compile(False literal) '' false ''
263      def dispatch compile(ParsExpression expression) ''
264      ( «FOR e : expression.expressions SEPARATOR ','» «e.compile» «ENDFOR» ) ''
265      def dispatch compile(PostProcessing expression) ''
266      <ERROR placeHolder node found on hql emitter>
```

```
267       ''
268       def dispatch compilej(QualifiedName qualifiedName) ''
269       «FOR e : qualifiedName.qualifiers SEPARATOR '.'»
270           «e.compilej»
271       «ENDFOR»
272       ''
273       def dispatch compilej(StringLiteral literal) ''
274       "«literal.value»"''
275       def dispatch boolean isPlaceHolder(BinaryFromExpression expression) {
276           var result = false
277           switch (expression.left) {
278               PostProcessing: result = true
279           }
280           switch (expression.right) {
281               PostProcessing: result = true
282           }
283           return result
284       }
285       def dispatch boolean isPlaceHolder(BinaryExpression expression) {
286           var result = false
287           switch (expression.left) {  PostProcessing: result = true }
288           switch (expression.right) { PostProcessing: result = true }
289           return result
290       }
291       def dispatch boolean isPlaceHolder(UnaryExpression expression) {
292           var result = false
293           switch (expression.expression) { PostProcessing: result = true }
294           return result
295       }
296   }
```

Classe `AQLValidator`.

```
1   class AQLValidator extends AbstractAQLValidator {
2       @Check
3       def checkEmptyClauses (QueryStatement query){
4           if (query.find == null && query.update == null && query.insert == null &&
                query.delete == null)
5               error ('You can have a main clause.', query.eContainer(), query.
                    eContainingFeature(), -1);
6           return;
7       }
8       @Check
9       def checkDuplicatedAlias (FindClause find) {
10          for (BindingObject bind : find.getBindingObject()) {
11              Set<String> set = new HashSet<String>(bind.getAlias());
12              if (set.size() < bind.getAlias().size())
13                  error ("Duplicated binding alias definition on find clause", query.
```

```
                  eContainer(), query.eContainingFeature(), -1);
14              return;
15          }
16      }
17      @Check
18      def checkSelectBindingAlias (ReturnsClause select) {
19          FindClause from = getFindClause(select);
20          if (from == null) {
21              error (ErrorList.SEMANTICANALYSIS_ERROR.getDescription(), query.
                    eContainer(), query.eContainingFeature(), -1);
22              return;
23          }
24          for (Expression selectExpression: select.getExpressions()) {
25              EList<QualifiedName> qualified = new BasicEList<QualifiedName>();
26              searchObjectIdentifier(selectExpression, qualified);
27              for (QualifiedName obj : qualified)
28              if (! existBindingAlias(from, obj) ) {
29                  String element = "";
30                  error ("Type not found", query.eContainer(), query.eContainingFeature
                        (), -1);
31                  return;
32              }
33          }
34      }
35      def EList<QualifiedName> searchObjectIdentifier (Expression expr, EList<
            QualifiedName> list) {
36          if (expr instanceof QualifiedName)
37              list.add((QualifiedName)expr);
38          else
39              for (EObject e : expr.eContents())
40                  if (e instanceof Expression)
41                      list = searchObjectIdentifier((Expression)e, list);
42          return list;
43      }
44      def String getMessageError (String message, String element, EObject object) {
45          INode node = NodeModelUtils.getNode(object);
46          String lin = Integer.toString(node.getStartLine());
47          String col = Integer.toString(node.getOffset());
48          return StringUtil.getMessage(message, element, lin, col);
49      }
50      def FindClause getFindClause (EObject clause) {
51          if (clause instanceof FindClause)
52              return (FindClause)clause;
53          EObject superClause = clause.eContainer();
54              while ( (superClause != null) && (! (superClause instanceof
                    QueryStatement)) )
55              superClause = superClause.eContainer();
```

```
56          if (superClause instanceof QueryStatement)
57              return ((QueryStatement)superClause).getFind();
58
59          return null;
60      }
61      def boolean existBindingAlias (FindClause find, QualifiedName alias) {
62          boolean result = true;
63          QualifiedElement element = alias.getQualifiers().get(0);
64          String aliasRoot = "";
65          if (element instanceof Identifier) {
66              aliasRoot = ((Identifier)element).getId();
67              result = existAliasFind(find, aliasRoot);
68          }
69          return result;
70      }
71      def boolean existAliasFind(FindClause find, String aliasRoot) {
72          for (BindingObject b : find.getBindingObject())
73              for (String a : b.getAlias())
74                  if (aliasRoot.equals(a))
75                      return true;
76          return false;
77      }
78  }
```

ANEXO E – CLASSES MODIFICADAS DO AOPJUNGLE

Classe `AOJungleVisitor`.

```
1  public class AOJungleVisitor extends AjASTVisitor {
2      @XStreamOmitField
3      private AOJCompilationUnit cUnit;
4      @XStreamOmitField
5      private static Stack<Object> elementStack = new Stack<Object>();
6      Map<String, FieldDeclaration> nodesFieldsDeclaration = new HashMap<>();
7      Map<String, VariableDeclarationStatement> nodesVariableDeclarationStatement =
           new HashMap<>();
8      Map<String, SingleVariableDeclaration> nodesSingleVariableDeclaration = new
           HashMap<>();
9      public AOJungleVisitor(AOJCompilationUnit cUnit) {
10         this.cUnit = cUnit;
11     }
12     public AOJProgramElement getCompilationUnit() {
13         return cUnit;
14     }
15     SuppressWarnings("unused")
16     private AOJungleVisitor() { }
17     @Override
18     public boolean visit(AnonymousClassDeclaration node) {
19         AOJAnonymousClassDeclaration aojNode = new AOJAnonymousClassDeclaration(node
               , cUnit);
20         AOJClassHolder classHolder = getAOJClassHolder(getLastMemberFromStack());
21         if (classHolder != null)
22         classHolder.getAnonymousClasses().add(aojNode);
23         elementStack.push(aojNode);
24         return super.visit(node);
25     }
26     @Override
27     public void endVisit(AnonymousClassDeclaration node) {
28         elementStack.pop();
29         super.endVisit(node);
30     }
31     @Override
32     public boolean visit(EnumConstantDeclaration node) {
33         return super.visit(node);
34     }
35     @Override
36     public boolean visit(EnumDeclaration node) {
37         AOJContainer aojNode = new AOJEnumDeclaration(node, cUnit);
38         cUnit.addType(aojNode);
```

```
39          getElementStack().push(aojNode);
40          return super.visit(node);
41      }
42      @Override
43      public void endVisit(EnumDeclaration node) {
44          getElementStack().pop();
45          super.endVisit(node);
46      }
47      public boolean visit(TypeDeclaration node) {
48          AOJContainer aojNode;
49          if (((AjTypeDeclaration) node).isAspect())
50          aojNode = new AOJAspectDeclaration(node, cUnit);
51          else if (((AjTypeDeclaration) node).isInterface())
52          aojNode = new AOJInterfaceDeclaration(node, cUnit);
53          else
54          aojNode = new AOJClassDeclaration(node, cUnit);
55          if (aojNode instanceof AOJClassDeclaration) {
56              if ((node.isLocalTypeDeclaration()) && (getElementStack().peek()
                    instanceof AOJClassHolder))
57              ((AOJClassHolder) getElementStack().peek()).getAnonymousClasses().add(
                    aojNode);
58              else if ((node.isMemberTypeDeclaration()) && (getElementStack().peek()
                    instanceof AOJClassHolder))
59              ((AOJClassHolder) getElementStack().peek()).getInnerClasses().add(aojNode
                    );
60              else
61              cUnit.addType(aojNode);
62          } else
63          cUnit.addType(aojNode);
64          getElementStack().push(aojNode);
65          return true;
66      }
67      @Override
68      public void endVisit(TypeDeclaration node) {
69          getElementStack().pop();
70          super.endVisit(node);
71      }
72      public boolean visit(MethodDeclaration node) {
73          if (getLastMemberFromStack() instanceof AOJContainer) {
74              if (node.isConstructor()) {
75                  AOJConstructorDeclaration aojNode = new AOJConstructorDeclaration(
                        node, (AOJProgramElement) getLastMemberFromStack());
76                  ((AOJConcreteTypeMember) ((AOJContainer) getLastMemberFromStack()).
                        getMembers()).getConstructors().add(aojNode);
77                  elementStack.push(aojNode);
78              } else {
```

```
79              AOJMethodDeclaration aojNode = new AOJMethodDeclaration(node, (
                    AOJProgramElement) getLastMemberFromStack());
80              ((AOJContainer) getLastMemberFromStack()).getMembers().getMethods().
                    add(aojNode);
81              elementStack.push(aojNode);
82          }
83      }
84      List<SingleVariableDeclaration> parameters = node.parameters();
85      for (SingleVariableDeclaration paramsMethod : parameters) {
86          nodesSingleVariableDeclaration.put(paramsMethod.getName().toString(),
                paramsMethod);
87      }
88      return super.visit(node);
89  }
90  public void endVisit(MethodDeclaration node) {
91      Object aojNode = elementStack.pop();
92      if (aojNode instanceof AOJMethodDeclaration){
93          ((AOJMethodDeclaration) aojNode).setStatementsBody(((AOJMethodDeclaration
                ) aojNode).getStatements().size());
94          if (((AOJMethodDeclaration) aojNode).getAnnotations().size() > 0)
95              ((AOJMethodDeclaration) aojNode).setHaveAnnotation(Boolean.TRUE);
96          if (null != ((AOJMethodDeclaration) aojNode).getReturnType())
97              ((AOJMethodDeclaration) aojNode).setReturnTypeName(((
                    AOJMethodDeclaration) aojNode).getReturnType().getName());
98      }
99      if (aojNode instanceof AOJConstructorDeclaration){
100         ((AOJConstructorDeclaration) aojNode).setStatementsBody(((
                AOJConstructorDeclaration) aojNode).getStatements().size());
101     }
102 }
103 @Override
104 public boolean visit(FieldDeclaration node) {
105     List<VariableDeclarationFragment> fragments = node.fragments();
106     for (VariableDeclarationFragment fragm : fragments)
107         nodesFieldsDeclaration.put(fragm.getName().toString(), node);
108     AOJAttributeDeclaration aojNode = new AOJAttributeDeclaration(node, (
            AOJProgramElement) getLastMemberFromStack());
109     ((AOJContainer) getLastMemberFromStack()).getMembers().getAttributes().add(
            aojNode);
110     elementStack.push(aojNode);
111     return super.visit(node);
112 }
113 @Override
114 public void endVisit(FieldDeclaration node) {
115     elementStack.pop();
116     super.endVisit(node);
117 }
```

```
118     @Override
119     public boolean visit(DeclarePrecedenceDeclaration node) {
120         AOJDeclarePrecedence aojNode = new AOJDeclarePrecedence(node, (
                AOJProgramElement) getLastMemberFromStack());
121         if (getLastMemberFromStack() instanceof AOJAspectDeclaration)
122         ((AOJAspectDeclaration) getLastMemberFromStack()).getMembers().
                getDeclarePrecedences().add(aojNode);
123         elementStack.push(aojNode);
124         return super.visit(node);
125     }
126     @Override
127     public void endVisit(DeclarePrecedenceDeclaration node) {
128         elementStack.pop();
129         super.endVisit(node);
130     }
131     @Override
132     public boolean visit(DeclareErrorDeclaration node) {
133         AOJDeclareCompilationEnforcement aojNode = new
                AOJDeclareCompilationEnforcement(node, (AOJProgramElement)
                getLastMemberFromStack(), node.getPointcut(), node.getMessage());
134         if (getLastMemberFromStack() instanceof AOJAspectDeclaration)
135         ((AOJAspectDeclaration) getLastMemberFromStack()).getMembers().
                getDeclareErrors().add(aojNode);
136         elementStack.push(aojNode);
137         return super.visit(node);
138     }
139     @Override
140     public void endVisit(DeclareErrorDeclaration node) {
141         elementStack.pop();
142         super.endVisit(node);
143     }
144     @Override
145     public boolean visit(DeclareWarningDeclaration node) {
146         AOJDeclareCompilationEnforcement aojNode = new
                AOJDeclareCompilationEnforcement(node, (AOJProgramElement)
                getLastMemberFromStack(), node.getPointcut(), node.getMessage());
147         if (getLastMemberFromStack() instanceof AOJAspectDeclaration)
148         ((AOJAspectDeclaration) getLastMemberFromStack()).getMembers().
                getDeclareWarnings().add(aojNode);
149         elementStack.push(aojNode);
150         return super.visit(node);
151     }
152     @Override
153     public void endVisit(DeclareWarningDeclaration node) {
154         elementStack.pop();
155         super.endVisit(node);
156     }
```

```
157     public boolean visit(DeclareParentsDeclaration node) {
158         AOJDeclareParents aojNode = new AOJDeclareParents(node, (AOJProgramElement)
                getLastMemberFromStack());
159         if (getLastMemberFromStack() instanceof AOJAspectDeclaration)
160         ((AOJAspectDeclaration) getLastMemberFromStack()).getMembers().
                getDeclareParents().add(aojNode);
161         elementStack.push(aojNode);
162         return super.visit(node);
163     }
164     @Override
165     public void endVisit(DeclareParentsDeclaration node) {
166         elementStack.pop();
167         super.endVisit(node);
168     }
169     @Override
170     public boolean visit(MarkerAnnotation node) {
171         AOJAnnotationDeclaration aojNode = new AOJAnnotationDeclaration(node, (
                AOJProgramElement) getLastMemberFromStack());
172         if (getLastMemberFromStack() instanceof AOJAnnotable)
173         ((AOJAnnotable) getLastMemberFromStack()).getAnnotations().add(aojNode);
174         elementStack.push(aojNode);
175         return super.visit(node);
176     }
177     @Override
178     public void endVisit(MarkerAnnotation node) {
179         elementStack.pop();
180         super.endVisit(node);
181     }
182     @Override
183     public boolean visit(AnnotationTypeMemberDeclaration node) {
184         return super.visit(node);
185     }
186     @Override
187     public boolean visit(AnnotationTypeDeclaration node) {
188         AOJAnnotationTypeDeclaration aojNode = new AOJAnnotationTypeDeclaration(node
                , cUnit);
189         cUnit.addType(aojNode);
190         getElementStack().push(aojNode);
191         return super.visit(node);
192     }
193     @Override
194     public void endVisit(AnnotationTypeDeclaration node) {
195         elementStack.pop();
196         super.endVisit(node);
197     }
198     @Override
199     public boolean visit(CflowPointcut node) {
```

```
200         if (getLastMemberFromStack() instanceof AOJClassDeclaration)
201         ((AOJClassDeclaration) getLastMemberFromStack()).getMembers().
                getcFlowPointcuts().add(new AOJPointcutCFlow(node, (AOJProgramElement)
                getLastMemberFromStack()));
202         return super.visit(node);
203     }
204     @Override
205     public boolean visit(InterTypeFieldDeclaration node) {
206         AOJDeclareField aojNode = new AOJDeclareField(node, (AOJProgramElement)
                getLastMemberFromStack());
207         if (getLastMemberFromStack() instanceof AOJAspectDeclaration)
208         ((AOJAspectDeclaration) getLastMemberFromStack()).getMembers().
                getDeclareFields().add(aojNode);
209         elementStack.push(aojNode);
210         return super.visit(node);
211     }
212     @Override
213     public void endVisit(InterTypeFieldDeclaration node) {
214         elementStack.pop();
215         super.endVisit(node);
216     }
217     @Override
218     public boolean visit(InterTypeMethodDeclaration node) {
219         AOJDeclareMethod aojNode = new AOJDeclareMethod(node, (AOJProgramElement)
                getLastMemberFromStack());
220         if (getLastMemberFromStack() instanceof AOJAspectDeclaration)
221         ((AOJAspectDeclaration) getLastMemberFromStack()).getMembers().
                getAllDeclareMethods().add(aojNode);
222         elementStack.push(aojNode);
223         return super.visit(node);
224     }
225     @Override
226     public void endVisit(InterTypeMethodDeclaration node) {
227         elementStack.pop();
228         super.endVisit(node);
229     }
230     @Override
231     public boolean visit(Block node) {
232         if (getLastMemberFromStack() instanceof AOJIfStatement)
233             ((AOJIfStatement) getLastMemberFromStack()).changeIfElse();
234         return super.visit(node);
235     }
236     @Override
237     public boolean visit(IfStatement node) {
238         AOJIfStatement aojNode = new AOJIfStatement(node, (AOJProgramElement)
                getLastMemberFromStack());
239         addStatements(aojNode);
```

```
240         getElementStack().push(aojNode);
241         return super.visit(node);
242     }
243     @Override
244     public void endVisit(IfStatement node) {
245         getElementStack().pop();
246         super.endVisit(node);
247     }
248     @Override
249     public boolean visit(ForStatement node) {
250         return super.visit(node);
251     }
252     @Override
253     public boolean visit(WhileStatement node) {
254         AOJWhileStatement aojNode = new AOJWhileStatement(node, (AOJProgramElement)
                getLastMemberFromStack());
255         addStatements(aojNode);
256         if (node.getExpression() instanceof MethodInvocation){
257             if (((MethodInvocation) node.getExpression()).getExpression() instanceof
                    SimpleName ){
258                 VariableDeclarationStatement variableDeclarationStatement =
                        nodesVariableDeclarationStatement.get(((MethodInvocation) node.
                        getExpression()).getExpression().toString());
259                 if (null == variableDeclarationStatement)
260                     aojNode.setCollection(false);
261                 else if (variableDeclarationStatement.getType() instanceof
                        ParameterizedType )
262                     aojNode.setCollection(true);
263                 else
264                     aojNode.setCollection(false);
265             }
266         }
267         getElementStack().push(aojNode);
268         return super.visit(node);
269     }
270     @Override
271     public void endVisit(WhileStatement node) {
272         getElementStack().pop();
273         super.endVisit(node);
274     }
275     @Override
276     public boolean visit(DoStatement node) {
277         return super.visit(node);
278     }
279     @Override
280     public boolean visit(TryStatement node) {
281         return super.visit(node);
```

```
282     }
283     @Override
284     public boolean visit(SwitchStatement node) {
285         AOJSwitchStatement aojNode = new AOJSwitchStatement(node, (AOJProgramElement
                ) getLastMemberFromStack());
286         addStatements(aojNode);
287         node.getExpression();
288         getElementStack().push(aojNode);
289         return super.visit(node);
290     }
291     @Override
292     public void endVisit(SwitchStatement node) {
293         getElementStack().pop();
294         super.endVisit(node);
295     }
296     @Override
297     public boolean visit(SynchronizedStatement node) {
298         return super.visit(node);
299     }
300     @Override
301     public boolean visit(ReturnStatement node) {
302         AOJExpressionStatement aojNode = new AOJExpressionStatement(node, (
                AOJProgramElement) getLastMemberFromStack());
303         addStatements(aojNode);
304         getElementStack().push(aojNode);
305         return super.visit(node);
306     }
307     @Override
308     public void endVisit(ReturnStatement node) {
309         getElementStack().pop();
310         super.endVisit(node);
311     }
312     @Override
313     public boolean visit(ThrowStatement node) {
314         return super.visit(node);
315     }
316     @Override
317     public boolean visit(BreakStatement node) {
318         return super.visit(node);
319     }
320     @Override
321     public boolean visit(ContinueStatement node) {
322         return super.visit(node);
323     }
324     @Override
325     public boolean visit(ExpressionStatement node) {
```

```
326         AOJExpressionStatement aojNode = new AOJExpressionStatement(node, (
                AOJProgramElement) getLastMemberFromStack());
327         addStatements(aojNode);
328         elementStack.push(aojNode);
329         return super.visit(node);
330     }
331     @Override
332     public void endVisit(ExpressionStatement node) {
333         elementStack.pop();
334         super.endVisit(node);
335     }
336     @Override
337     public boolean visit(LabeledStatement node) {
338         return super.visit(node);
339     }
340     @Override
341     public boolean visit(AssertStatement node) {
342         return super.visit(node);
343     }
344     @Override
345     public boolean visit(EnhancedForStatement node) {
346         AOJEnhancedForStatement aojNode = new AOJEnhancedForStatement(node, (
                AOJProgramElement) getLastMemberFromStack());
347         addStatements(aojNode);
348         FieldDeclaration fieldDeclaration = nodesFieldsDeclaration.get(node.
                getExpression().toString());
349         if (null == fieldDeclaration)
350             aojNode.setCollection(false);
351         else if (fieldDeclaration.getType() instanceof ParameterizedType ){
352             aojNode.setCollection(true);
353         else
354             aojNode.setCollection(false);
355         elementStack.push(aojNode);
356         return super.visit(node);
357     }
358     @Override
359     public void endVisit(EnhancedForStatement node) {
360         elementStack.pop();
361         super.endVisit(node);
362     }
363     @Override
364     public boolean visit(ConstructorInvocation node) {
365         return super.visit(node);
366     }
367     @Override
368     public boolean visit(SuperConstructorInvocation node) {
369         return super.visit(node);
```

```
370     }
371     @Override
372     public boolean visit(BeforeAdviceDeclaration node) {
373         AOJAdviceDeclaration aojNode = new AOJAdviceDeclaration(node, (
                AOJProgramElement) getLastMemberFromStack(), AOJAdviceKindEnum.BEFORE);
374         if (getLastMemberFromStack() instanceof AOJAspectDeclaration)
375         ((AOJAspectDeclaration) getLastMemberFromStack()).getMembers().getAllAdvices
                ().add(aojNode);
376         elementStack.push(aojNode);
377         return super.visit(node);
378     }
379     @Override
380     public void endVisit(BeforeAdviceDeclaration node) {
381         elementStack.pop();
382         super.endVisit(node);
383     }
384     @Override
385     public boolean visit(AfterAdviceDeclaration node) {
386         AOJAdviceDeclaration aojNode = new AOJAdviceDeclaration(node, (
                AOJProgramElement) getLastMemberFromStack(), AOJAdviceKindEnum.AFTER);
387         if (getLastMemberFromStack() instanceof AOJAspectDeclaration)
388         ((AOJAspectDeclaration) getLastMemberFromStack()).getMembers().getAllAdvices
                ().add(aojNode);
389         elementStack.push(aojNode);
390         return super.visit(node);
391     }
392     @Override
393     public void endVisit(AfterAdviceDeclaration node) {
394         elementStack.pop();
395         super.endVisit(node);
396     }
397     @Override
398     public boolean visit(AfterReturningAdviceDeclaration node) {
399         AOJAdviceDeclaration aojNode = new AOJAdviceDeclaration(node, (
                AOJProgramElement) getLastMemberFromStack(), AOJAdviceKindEnum.
                AFTERRETURNING);
400         if (getLastMemberFromStack() instanceof AOJAspectDeclaration)
401         ((AOJAspectDeclaration) getLastMemberFromStack()).getMembers().getAllAdvices
                ().add(aojNode);
402         elementStack.push(aojNode);
403         return super.visit(node);
404     }
405     @Override
406     public void endVisit(AfterReturningAdviceDeclaration node) {
407         elementStack.pop();
408         super.endVisit(node);
409     }
```

```
410     @Override
411     public boolean visit(AfterThrowingAdviceDeclaration node) {
412         AOJAdviceDeclaration aojNode = new AOJAdviceDeclaration(node, (
                AOJProgramElement) getLastMemberFromStack(), AOJAdviceKindEnum.
                AFTERTHROWING);
413         if (getLastMemberFromStack() instanceof AOJAspectDeclaration)
414         ((AOJAspectDeclaration) getLastMemberFromStack()).getMembers().getAllAdvices
                ().add(aojNode);
415         elementStack.push(aojNode);
416         return super.visit(node);
417     }
418     @Override
419     public void endVisit(AfterThrowingAdviceDeclaration node) {
420         elementStack.pop();
421         super.endVisit(node);
422     }
423     @Override
424     public boolean visit(AroundAdviceDeclaration node) {
425         AOJAdviceDeclaration aojNode = new AOJAdviceDeclaration(node, (
                AOJProgramElement) getLastMemberFromStack(), AOJAdviceKindEnum.AROUND);
426         if (getLastMemberFromStack() instanceof AOJAspectDeclaration)
427         ((AOJAspectDeclaration) getLastMemberFromStack()).getMembers().getAllAdvices
                ().add(aojNode);
428         elementStack.push(aojNode);
429         return super.visit(node);
430     }
431     @Override
432     public void endVisit(AroundAdviceDeclaration node) {
433         elementStack.pop();
434         super.endVisit(node);
435     }
436     @Override
437     public boolean visit(PointcutDeclaration node) {
438         AOJPointcutDeclaration aojNode = new AOJPointcutDeclaration(node, (
                AOJProgramElement) getLastMemberFromStack());
439         if (getLastMemberFromStack() instanceof AOJAspectDeclaration)
440         ((AOJAspectDeclaration) getLastMemberFromStack()).getMembers().getPointcuts
                ().add(aojNode);
441         else if (getLastMemberFromStack() instanceof AOJClassDeclaration)
442         ((AOJClassDeclaration) getLastMemberFromStack()).getMembers().getPointcuts()
                .add(aojNode);
443         elementStack.push(aojNode);
444         return super.visit(node);
445     }
446     @Override
447     public void endVisit(PointcutDeclaration node) {
448         elementStack.pop();
```

```
449         super.endVisit(node);
450     }
451     @Override
452     public boolean visit(DeclareSoftDeclaration node) {
453         AOJDeclareSoft aojNode = new AOJDeclareSoft(node, (AOJProgramElement)
                getLastMemberFromStack());
454         if (getLastMemberFromStack() instanceof AOJAspectDeclaration)
455         ((AOJAspectDeclaration) getLastMemberFromStack()).getMembers().
                getDeclareSofts().add(aojNode);
456         elementStack.push(aojNode);
457         return super.visit(node);
458     }
459     @Override
460     public void endVisit(DeclareSoftDeclaration node) {
461         elementStack.pop();
462         super.endVisit(node);
463     }
464     @Override
465     public boolean visit(VariableDeclarationStatement node) {
466         AOJVariableDeclarationStatement aojNode = new
                AOJVariableDeclarationStatement(node, (AOJProgramElement)
                getLastMemberFromStack());
467         addStatements(aojNode);
468         List<VariableDeclarationFragment> fragments = node.fragments();
469         for (VariableDeclarationFragment fragm : fragments) {
470             nodesVariableDeclarationStatement.put(fragm.getName().toString(), node);
471         }
472         elementStack.push(aojNode);
473         return super.visit(node);
474     }
475     @Override
476     public void endVisit(VariableDeclarationStatement node) {
477         elementStack.pop();
478         super.endVisit(node);
479     }
480     public Object getLastMemberFromStack() {
481         return elementStack.peek();
482     }
483     public static Stack<Object> getElementStack() {
484         return elementStack;
485     }
486     public static AOJClassHolder getAOJClassHolder(Object object) {
487         if (object == null)
488         return null;
489         if (object instanceof AOJClassHolder)
490         return (AOJClassHolder) object;
491         else
```

```
492         return getAOJClassHolder(((AOJProgramElement) object).getOwner());
493
494     }
495     @Override
496     public boolean visit(ClassInstanceCreation node) {
497         return super.visit(node);
498     }
499     @Override
500     public void endVisit(ClassInstanceCreation node) {
501         super.endVisit(node);
502     }
503     @Override
504     public boolean visit(SingleVariableDeclaration node) {
505         return super.visit(node);
506     }
507     @Override
508     public void endVisit(SingleVariableDeclaration node) {
509         super.endVisit(node);
510     }
511     @Override
512     public boolean visit(MethodInvocation node) {
513         return super.visit(node);
514     }
515     @Override
516     public void endVisit(MethodInvocation node) {
517         super.endVisit(node);
518     }
519     private void addStatements(AOJStatement aojNode) {
520         if (getLastMemberFromStack() instanceof AOJMethodDeclaration) {
521             ((AOJMethodDeclaration) getLastMemberFromStack()).getStatements().add(
                    aojNode);
522         } else if (getLastMemberFromStack() instanceof AOJConstructorDeclaration) {
523             ((AOJConstructorDeclaration) getLastMemberFromStack()).getStatements().
                    add(aojNode);
524         } else if (getLastMemberFromStack() instanceof AOJIfStatement) {
525             AOJIfStatement ifnode = (AOJIfStatement) getLastMemberFromStack();
526             if (ifnode.getIfElse() == 1) {
527                 ((AOJIfStatement) getLastMemberFromStack()).getStatements().add(
                        aojNode);
528             } else {
529                 ((AOJIfStatement) getLastMemberFromStack()).getElseStatements().add(
                        aojNode);
530             }
531         }
532         else if (getLastMemberFromStack() instanceof AOJStatement) {
533             ((AOJStatement) getLastMemberFromStack()).getStatements().add(aojNode);
534         }
```

```
535     }
536 }
```

Classe AOJMember.

```
1  @MappedSuperclass
2  public abstract class AOJMember extends ASTElement implements AOJAnnotable,
       AOJBindble {
3      @OneToOne (cascade=CascadeType.ALL)
4      private AOJModifier modifiers;
5      @OneToMany(cascade=CascadeType.ALL)
6      private List<AOJAnnotationDeclaration> annotations;
7      @OneToOne (cascade=CascadeType.ALL)
8      private AOJBindingModel bindingModel = new AOJBindingModel();
9      private String internalHandler;
10     private int createByAQL = 0;
11     @Lob
12     private String codeBody = "";
13     public String metaName = "undefined";
14     public int codeOrder = 0;
15     @Transient
16     public static int contadorInterno = 0;
17     public AOJMember(ASTNode node, AOJProgramElement owner) {
18         super(node, owner);
19         this.contadorInterno += 5;
20         this.codeOrder = this.contadorInterno;
21     }
22     public AOJMember(ASTNode node, AOJProgramElement owner, int isInsertByAQL) {
23         super(node, owner);
24         setCreateByAQL(1);
25     }
26     protected AOJMember() { }
27     protected void loadToReferenceMap() {
28         AOJReferenceManager.getInstance().addToMap(getInternalHandler(), this);
29     }
30     public AOJModifier getModifiers() {
31         if (modifiers == null)
32         modifiers = new AOJModifier(getModifiersAsInteger());
33         return modifiers;
34     }
35     public List<AOJAnnotationDeclaration> getAnnotations() {
36         if (null == annotations)
37         annotations = new ArrayList<AOJAnnotationDeclaration>();
38         return annotations;
39     }
40     public abstract int getModifiersAsInteger();
41     protected void setModifiers(AOJModifier modifiers) {
42         this.modifiers = modifiers;
```

```
43      }
44      public AOJBindingModel getBindingModel() {
45          return bindingModel;
46      }
47      public void setBindingModel(AOJBindingModel bindingModel) {
48          this.bindingModel = bindingModel;
49      }
50      protected abstract void loadHandler();
51      public String getInternalHandler() {
52          return internalHandler;
53      }
54      public void setInternalHandler(String handler) {
55          this.internalHandler = handler;
56      }
57      public String getMetaName() {
58          return metaName;
59      }
60      public void setMetaName(String metaName) {
61          this.metaName = metaName;
62      }
63      public String getCodeBody() {
64          return codeBody;
65      }
66      public void setCodeBody(String code) {
67          this.codeBody = code;
68      }
69      public int getCreateByAQL() {
70          return createByAQL;
71      }
72      public void setCreateByAQL(int createByAQL) {
73          this.createByAQL = createByAQL;
74      }
75      public int getCodeOrder() {
76          return codeOrder;
77      }
78      public void setCodeOrder(int codeOrder) {
79          this.codeOrder = codeOrder;
80      }
81  }
```

Classe `AOJStatement`.

```
1   @Entity
2   public class AOJStatement extends AOJMember {
3       @OneToMany(cascade=CascadeType.ALL)
4       protected List<AOJStatement> statements;
5       private String header;
6       protected AOJStatement() {
```

```
7          statements = new ArrayList<>();
8      }
9      public AOJStatement(ASTNode node, AOJProgramElement owner) {
10         super(node, owner);
11         statements = new ArrayList<>();
12     }
13     @Override
14     public int getModifiersAsInteger() {
15         return 0;
16     }
17     @Override
18     protected void loadHandler() { }
19     public List<AOJStatement> getStatements() {
20         return statements;
21     }
22     public void addStatements(AOJStatement statements) {
23         this.statements.add(statements);
24     }
25     public String getHeader() {
26         return header;
27     }
28     public void setHeader(String header) {
29         this.header = header;
30     }
31     public void addElement(AOJStatement aoj){
32         this.statements.add(aoj);
33     }
34 }
```

Classe `AOJClassDeclaration`.

```
1  @Entity
2  @DiscriminatorValue("CLA")
3  public class AOJClassDeclaration extends AOJTypeDeclaration implements
       AOJClassHolder, AOJInterfaceble {
4      @OneToMany(cascade=CascadeType.ALL)
5      private List<AOJReferencedType> implementedInterfaces;
6      @OneToOne (cascade=CascadeType.ALL)
7      protected AOJConcreteTypeMember members;
8      @OneToOne (cascade=CascadeType.ALL)
9      private AOJTypeMetrics metrics;
10     protected AOJClassDeclaration() {
11         setMetaName("Class");
12     }
13     private Boolean implementsInterface = Boolean.FALSE;
14     public AOJClassDeclaration(ASTNode node, AOJProgramElement owner) {
15         super(node, owner);
16         setMetaName("Class");
```

```
17        members = new AOJConcreteTypeMember();
18        metrics = AOJMetricsFactory.eINSTANCE.createTypeMetrics();
19        loadImplementedInterfaces();
20        if (getImplementedInterfaces().size() > 0){
21            implementsInterface = Boolean.TRUE;
22        }
23    }
24    protected void loadImplementedInterfaces() {
25        for (Object intf : getClassDeclaration().superInterfaceTypes())
26            getImplementedInterfaces().add(new AOJReferencedType((Type)intf, this));
27    }
28    protected void loadParent() {
29        if (null != getClassDeclaration().getSuperclassType()) {
30            setParent (new AOJReferencedType(getClassDeclaration().getSuperclassType
                  (), this));
31        }
32    }
33    public List<AOJReferencedType> getImplementedInterfaces() {
34        if (implementedInterfaces == null)
35            implementedInterfaces = new ArrayList<AOJReferencedType>();
36        return implementedInterfaces;
37    }
38    public String getASTTypeName() {
39        return getClassDeclaration().getName().toString();
40    }
41    public String getASTTypeFullQualifiedName() {
42        StringBuilder sb = new StringBuilder();
43        sb.append(getPackage().getName()).append(".").append(getClassDeclaration().
              getName());
44        return sb.toString();
45    }
46    @Override
47    public int getModifiersAsInteger() {
48        return getClassDeclaration().getModifiers();
49    }
50    @Override
51    protected void loadMembers() { }
52    public AOJConcreteTypeMember getMembers() {
53        return members;
54    }
55    public List<AOJContainer> getAnonymousClasses() {
56        return members.getAnonymousClasses();
57    }
58    public List<AOJContainer> getInnerClasses() {
59        return members.getInnerClasses();
60    }
61    public TypeDeclaration getClassDeclaration() {
```

```
62          ASTNode node = getNode();
63          return (TypeDeclaration)node;
64      }
65      @Override
66      public void loadMetrics() {
67          super.loadMetrics();
68          long lOfConstructors = getMetrics().getNumberOfConstructors() + getMembers()
                .getConstructors().size();
69          getMetrics().setNumberOfConstructors(lOfConstructors);
70      }
71      @Override
72      public AOJTypeMetrics getMetrics() {
73          return metrics;
74      }
75      public Boolean getIsImplementsInterface() {
76          return implementsInterface;
77      }
78      public void setIsImplementsInterface(Boolean implementsInterface) {
79          this.implementsInterface = implementsInterface;
80      }
81  }
```

Classe `AOJConstructorDeclaration`.

```
1   @Entity
2   @DiscriminatorValue("CTR")
3   public class AOJConstructorDeclaration extends AOJBehaviourKind implements
        AOJClassHolder {
4       protected AOJConstructorDeclaration() { }
5       @OneToMany(cascade=CascadeType.ALL)
6       protected List<AOJStatement> statements;
7       private Integer statementsBody = 0;
8       @Override
9       public String getMetaName() {
10          return "Constructor";
11      }
12      public AOJConstructorDeclaration(ASTNode node, AOJProgramElement owner) {
13          super(node, owner);
14          loadHandler();
15          loadToReferenceMap();
16          statements = new ArrayList<AOJStatement>();
17      }
18      protected void loadParameters() {
19          for (Object param : getConstructorDeclaration().parameters())
20              getParameters().add( new AOJParameter((SingleVariableDeclaration) param,
                    this));
21      }
22      @Override
```

```
23      protected void loadName() {
24          super.setName(getConstructorDeclaration().getName().toString());
25      }
26      @Override
27      public int getModifiersAsInteger() {
28          return ((MethodDeclaration) getNode()).getModifiers();
29      }
30      protected MethodDeclaration getConstructorDeclaration() {
31          ASTNode node = getNode();
32          return (MethodDeclaration)node;
33      }
34      @Override
35      protected void loadReturnType() {
36          setReturnType(null);
37      }
38      @Override
39      protected void loadThrownExceptions() {
40          for (Object exception : getConstructorDeclaration().thrownExceptions())
41          getThrownExceptions().add( new AOJException((Name) exception, this));
42      }
43      @Override
44      protected void loadCodeBody() {
45          setCode(getConstructorDeclaration().getBody().toString());
46      }
47      @Override
48      public void loadMetrics() {
49          getMetrics().setNumberOfStatements(getConstructorDeclaration().getBody() ==
                null ? 0 :
50          AOJSourceService.getNumberOfStatements(getConstructorDeclaration().getBody()
                .statements()));
51          long nOfLines = 0;
52          nOfLines = getConstructorDeclaration().getBody() == null ? 0 :
53          AOJSourceService.getNumberOfStatements(getConstructorDeclaration().getBody()
                .statements());
54          nOfLines += 3;
55          getMetrics().setNumberOfLines(nOfLines);
56      }
57      @Override
58      protected void loadHandler() {
59          setInternalHandler(StringUtil.buildHandler(getPackage().getName(), ((
                AOJTypeDeclaration)getOwner()).getName(), getSignature()));
60      }
61      public List<AOJStatement> getStatements() {
62          return statements;
63      }
64      public void setStatements(List<AOJStatement> statements) {
65          this.statements = statements;
```

```
66      }
67      public void addElement(AOJStatement aoj){
68          this.statements.add(aoj);
69      }
70      public Integer getStatementsBody() {
71          return statementsBody;
72      }
73      public void setStatementsBody(Integer statementsBody) {
74          this.statementsBody = statementsBody;
75      }
76      public void addStatements(AOJStatement statement){
77          this.statements.add(statement);
78      }
79  }
```

Classe `AOJMethodDeclaration`.

```
1   @Entity
2   @DiscriminatorValue("MET")
3   public class AOJMethodDeclaration extends AOJBehaviourKind {
4       private boolean isConstructor;
5       @OneToOne (cascade=CascadeType.ALL)
6       protected AOJConcreteTypeMember members;
7       @OneToMany(cascade=CascadeType.ALL)
8       protected List<AOJStatement> statements;
9       private Integer statementsBody = 0;
10      private String returnTypeName = "void";
11      private Boolean hasAnnotation = Boolean.FALSE;
12      protected MethodDeclaration getMethodDeclaration() {
13          ASTNode node = getNode();
14          return (MethodDeclaration)node;
15      }
16      protected AOJMethodDeclaration() {
17          setMetaName("Method");
18      }
19      protected void loadOnType() { }
20      protected void loadPosition() { }
21      public AOJMethodDeclaration(ASTNode node, AOJProgramElement owner) {
22          super(node, owner);
23          setMetaName("Method");
24          loadPosition();
25          loadOnType();
26          loadHandler();
27          loadToReferenceMap();
28          loadCodeBody();
29          members = new AOJConcreteTypeMember();
30          statements = new ArrayList();
31      }
```

```
32    public AOJMethodDeclaration(ASTNode node, AOJProgramElement owner, int
          isInsertByAQL) {
33        super(node, owner, isInsertByAQL);
34        setMetaName("Method");
35        members = new AOJConcreteTypeMember();
36        statements = new ArrayList();
37    }
38    public void loadMetrics() {
39        super.loadMetrics();
40        getMetrics().setNumberOfStatements(getMethodDeclaration().getBody() == null
              ? 0 :
41        AOJSourceService.getNumberOfStatements(getMethodDeclaration().getBody().
              statements()));
42        long nOfLines = 0;
43        nOfLines += getMethodDeclaration().getBody() == null ? 0 :
44        AOJSourceService.getNumberOfStatements(getMethodDeclaration().getBody().
              statements());
45        if (getOwner() instanceof AOJInterfaceDeclaration)
46            nOfLines++;
47        else if (getModifiers().exists(AOJModifierEnum. ABSTRACT))
48            nOfLines++;
49        else
50            nOfLines += 3;
51        getMetrics().setNumberOfLines(nOfLines);
52        getMetrics().setNumberOfInAffects(getBindingModel().getNumberOfIn());
53        getMetrics().setNumberOfOutAffects(getBindingModel().getNumberOfOut());
54    }
55    @Override
56    protected void loadParameters() {
57        for (Object param : getMethodDeclaration().parameters())
58        getParameters().add( new AOJParameter((SingleVariableDeclaration) param,
              this));
59    }
60    @Override
61    protected void loadThrownExceptions() {
62        for (Object exception : getMethodDeclaration().thrownExceptions())
63        getThrownExceptions().add( new AOJException((Name) exception, this));
64    }
65    @Override
66    protected void loadReturnType() {
67        AOJType result;
68        if (null == getMethodDeclaration().getReturnType2())
69            result = new AOJReferencedType(null, this);
70        else if (getMethodDeclaration().getReturnType2().isPrimitiveType())
71            result = new AOJPrimitiveType(getMethodDeclaration().getReturnType2(),
                  this);
72        else
```

```
73            result = new AOJReferencedType(getMethodDeclaration().getReturnType2(),
                  this);
74         setReturnType(result);
75     }
76     @Override
77     protected void loadName() {
78         super.setName(getMethodDeclaration().getName().toString());
79     }
80     @Override
81     public String getMetaName() {
82         return "Method";
83     }
84     public boolean isConstructor() {
85         return isConstructor;
86     }
87     @Override
88     public int getModifiersAsInteger() {
89         return getMethodDeclaration().getModifiers();
90     }
91     @Override
92     protected void loadCodeBody() {
93         if (getMethodDeclaration().getBody() != null){
94             setCodeBody(getMethodDeclaration().getBody().toString());
95             setStatementsBody(getMethodDeclaration().getBody().statements().size());
96         }
97     }
98     @Override
99     protected void loadHandler() {
100        setInternalHandler(StringUtil.buildHandler(getPackage().getName(), ((
               AOJTypeDeclaration)getOwner()).getName(), getSignature()));
101    }
102    public Integer getStatementsBody() {
103        return statementsBody;
104    }
105    public void setStatementsBody(Integer statementsBody) {
106        this.statementsBody = statementsBody;
107    }
108    public AOJConcreteTypeMember getMembers() {
109        return members;
110    }
111    public List<AOJStatement> getStatements() {
112        return statements;
113    }
114    public void addElement(AOJStatement aoj){
115        this.statements.add(aoj);
116    }
117    public void addStatements(AOJStatement aoj){
```

```
118         this.statements.add(aoj);
119     }
120     public String getReturnTypeName() {
121         return returnTypeName;
122     }
123     public void setReturnTypeName(String returnTypeName) {
124         this.returnTypeName = returnTypeName;
125     }
126     public Boolean getHaveAnnotation() {
127         return hasAnnotation;
128     }
129     public void setHaveAnnotation(Boolean hasAnnotation) {
130         this.hasAnnotation = hasAnnotation;
131     }
132     public void loadByAQL(){}
133 }
```

Classe `AOJWhileStatement`.

```
 1 @Entity
 2 public class AOJWhileStatement extends AOJStatement {
 3     private String code;
 4     private String body;
 5     private String expression;
 6     private boolean isCollection;
 7     public AOJWhileStatement(){}
 8     public AOJWhileStatement(ASTNode node, AOJProgramElement owner) {
 9         super(node, owner);
10         WhileStatement e = (WhileStatement)node;
11         setCodeBody(e.toString());
12         setCode(e.toString());
13         setBody(e.getBody().toString());
14         setExpression(e.getExpression().toString());
15         loadHeader();
16         setMetaName("Foreach");
17     }
18     private void loadHeader() {
19         setHeader( "while( " + getExpression() +" )" );
20     }
21     public String getCode() {
22         return code;
23     }
24     public void setCode(String code) {
25         this.code = code;
26     }
27     public String getBody() {
28         return body;
29     }
```

```
30      public void setBody(String body) {
31          this.body = body;
32      }
33      public boolean isCollection() {
34          return isCollection;
35      }
36      public void setCollection(boolean isCollection) {
37          this.isCollection = isCollection;
38      }
39      public String getExpression() {
40          return expression;
41      }
42      public void setExpression(String expression) {
43          this.expression = expression;
44      }
45  }
```

Classe `AOJEnhancedForStatement`.

```
1   @Entity
2   public class AOJEnhancedForStatement extends AOJStatement {
3       private String code;
4       private String body;
5       private String expressionEnhancedFor;
6       private String parameterEnhancedFor;
7       private boolean isCollection;
8       public AOJEnhancedForStatement(){}
9       public AOJEnhancedForStatement(ASTNode node, AOJProgramElement owner) {
10          super(node, owner);
11          EnhancedForStatement e = (EnhancedForStatement)node;
12          setCodeBody(e.toString());
13          setCode(e.toString());
14          setBody(e.getBody().toString());
15          setExpressionEnhancedFor(e.getExpression().toString());
16          setParameterEnhancedFor(e.getParameter().toString());
17          loadHeader();
18          setMetaName("Foreach");
19      }
20      private void loadHeader() {
21          setHeader( "for( " + getParameterEnhancedFor() + " : " +
                getExpressionEnhancedFor() +" )" );
22      }
23      public String getCode() {
24          return code;
25      }
26      public void setCode(String code) {
27          this.code = code;
28      }
```

```
29      public String getBody() {
30          return body;
31      }
32      public void setBody(String body) {
33          this.body = body;
34      }
35      public String getExpressionEnhancedFor() {
36          return expressionEnhancedFor;
37      }
38      public void setExpressionEnhancedFor(String expressionEnhancedFor) {
39          this.expressionEnhancedFor = expressionEnhancedFor;
40      }
41      public String getParameterEnhancedFor() {
42          return parameterEnhancedFor;
43      }
44      public void setParameterEnhancedFor(String parameterEnhancedFor) {
45          this.parameterEnhancedFor = parameterEnhancedFor;
46      }
47      public boolean isCollection() {
48          return isCollection;
49      }
50      public void setCollection(boolean isCollection) {
51          this.isCollection = isCollection;
52      }
53  }
```

Classe `AOJSwitchStatement`.

```
1   @Entity
2   public class AOJSwitchStatement extends AOJStatement {
3       private String bodyCode;
4       private String typeOfCondition;
5       public AOJSwitchStatement() { }
6       public AOJSwitchStatement(ASTNode node, AOJProgramElement owner) {
7           super(node, owner);
8           SwitchStatement e = (SwitchStatement) node;
9           setBodyCode(e.toString());
10          setMetaName("Switch");
11          if (e.getExpression() instanceof SimpleName) {
12              SimpleName sn = (SimpleName) e.getExpression();
13              if (owner instanceof AOJMethodDeclaration) {
14                  AOJMethodDeclaration method = (AOJMethodDeclaration) owner;
15                  if (method.getOwner() instanceof AOJClassDeclaration) {
16                      AOJClassDeclaration aojClass = (AOJClassDeclaration) method.
                            getOwner();
17                      for (AOJAttributeDeclaration element : aojClass.getMembers().
                            getAttributes())
18                          if (element.getName().equals(sn.getIdentifier()))
```

```
19                              typeOfCondition = element.getType().getName();
20                  }
21                  for (AOJParameter element : method.getParameters())
22                      if (element.getName().equals(sn.getIdentifier()))
23                          typeOfCondition = element.getType().getName();
24                  for (AOJMember element : method.getStatements())
25                      if (element instanceof AOJVariableDeclarationStatement) {
26                          AOJVariableDeclarationStatement AOJVar = (
                                AOJVariableDeclarationStatement) element;
27                          if (AOJVar.getName().equals(sn.getIdentifier()))
28                              typeOfCondition = AOJVar.getType();
29                      }
30              }
31          }
32      }
33      public String getBodyCode() {
34          return bodyCode;
35      }
36      public void setBodyCode(String bodyCode) {
37          this.bodyCode = bodyCode;
38      }
39      public String getTypeOfCondition() {
40          return typeOfCondition;
41      }
42      public void setTypeOfCondition(String typeOfCondition) {
43          this.typeOfCondition = typeOfCondition;
44      }
45  }
```

Classe AOJExpressionStatement.

```
1   @Entity
2   public class AOJExpressionStatement extends AOJStatement {
3       private Boolean isCollectionsSort = Boolean.FALSE;
4       public AOJExpressionStatement() {}
5       public AOJExpressionStatement(ASTNode node, AOJProgramElement owner) {
6           super(node, owner);
7           setMetaName("Expression");
8           if (null != node)
9               if (node instanceof ExpressionStatement) {
10                  ExpressionStatement e = (ExpressionStatement) node;
11                  this.setCodeBody(e.toString());
12                  if (e.toString().startsWith("Collections.sort"))
13                      isCollectionsSort = Boolean.TRUE;
14                  if (e.toString().contains("new "))
15                      setMetaName("Instance Expression");
16              } else if (node instanceof ReturnStatement) {
17                  ReturnStatement e = (ReturnStatement) node;
```

```
18                this.setCodeBody(e.toString());
19                setMetaName("Return Expression");
20                if (e.toString().contains("new "))
21                    setMetaName("Return Instance Expression");
22            }
23    }
24    public AOJExpressionStatement(ASTNode node, AOJProgramElement owner, int
          createByAQL) {
25        super(node, owner);
26        setCreateByAQL(1);
27    }
28    public void loadByAQL(){
29        String metaName = "";
30        if (getCodeBody().contains("return "))
31        metaName += "Return ";
32        if (getCodeBody().contains("new "))
33        metaName += "Instance ";
34        if (getCodeBody().contains("->"))
35        metaName += "Lambda ";
36        metaName += "Expression";
37        setMetaName(metaName);
38    }
39    public Boolean getIsCollectionsSort() {
40        return isCollectionsSort;
41    }
42    public void setIsCollectionsSort(Boolean isCollectionsSort) {
43        this.isCollectionsSort = isCollectionsSort;
44    }
45 }
```

Classe JGenerator.

```
1  public class JGenerator {
2      RefactoringHelp rh = new RefactoringHelp();
3      public void generator(List<AOJMember> obj) {
4          orderByCodeOrder(obj);
5          for (AOJMember member : obj)
6              this.generator(member);
7      }
8      private void generator(AOJMember member) {
9          if (member instanceof AOJIfStatement)
10             generator((AOJIfStatement) member);
11         else if (member instanceof AOJExpressionStatement)
12             generator((AOJExpressionStatement) member);
13         else if (member instanceof AOJMethodDeclaration)
14             generator((AOJMethodDeclaration) member);
15         else if (member instanceof AOJEnhancedForStatement)
16             generator((AOJEnhancedForStatement) member);
```

```
17          else if (member instanceof AOJInterfaceDeclaration)
18              generator((AOJInterfaceDeclaration) member);
19          else if (member instanceof AOJClassDeclaration)
20              generator((AOJClassDeclaration) member);
21          else if (member instanceof AOJConstructorDeclaration)
22              generator((AOJConstructorDeclaration) member);
23          else
24              System.err.print("Not found: " + member);
25      }
26      public void generator(AOJIfStatement statement) {
27          System.out.println(statement.getHeader() + "{");
28              orderByCodeOrder(statement.getStatements());
29              for (AOJStatement aoj : statement.getStatements())
30                  this.generator((AOJMember) aoj);
31              System.out.println("}");
32          if (statement.getElseStatements().size() > 0) {
33              System.out.println("else {");
34              orderByCodeOrder(statement.getElseStatements());
35              for (AOJStatement aoj : statement.getElseStatements())
36                  this.generator((AOJMember) aoj);
37              System.out.println("}");
38          }
39      }
40      public void generator(AOJExpressionStatement statement) {
41          System.out.print("\t" + statement.getCodeBody());
42      }
43      public void generator(AOJMethodDeclaration statement) {
44          if (statement.getOwner() instanceof AOJInterfaceDeclaration) {
45              if (statement.getStatements().size() == 0) {
46                  System.out.println(statement.getSignature() + ";");
47              } else {
48                  System.out.println(statement.getSignature() + "{");
49                  orderByCodeOrder(statement.getStatements());
50                  for (AOJStatement aoj : statement.getStatements())
51                      this.generator((AOJMember) aoj);
52                  System.out.println("}");
53              }
54          } else {
55              System.out.println(statement.getSignature() + "{");
56              orderByCodeOrder(statement.getStatements());
57              for (AOJStatement aoj : statement.getStatements())
58                  this.generator((AOJMember) aoj);
59              System.out.println("}");
60          }
61      }
62      public void generator(AOJEnhancedForStatement statement) {
63          System.out.println(statement.getHeader() + "{");
```

```
64          orderByCodeOrder(statement.getStatements());
65          for (AOJStatement aoj : statement.getStatements())
66              this.generator((AOJMember) aoj);
67          System.out.println("}");
68      }
69      public void generator(AOJInterfaceDeclaration statement) {
70          String modifier = "";
71          if (null != statement.getModifiers())
72              modifier = statement.getModifiers().getModifiers().get(0).getName().
                    toLowerCase();
73          System.out.println(modifier + " interface " + statement.getName() + "{");
74          String hquery = "Select method " + "FROM AOJMethodDeclaration method " + "
                JOIN method.owner interface" + " WHERE interface.id = " + statement.
                getId();
75          List<AOJMethodDeclaration> objetos = rh.session.createQuery(hquery).list();
76          for (AOJMethodDeclaration aoj : objetos)
77              this.generator((AOJMember) aoj);
78          System.out.println("}");
79      }
80      public void generator(AOJClassDeclaration statement) {
81          String modifier = "";
82          if (null != statement.getModifiers())
83              modifier = statement.getModifiers().getModifiers().get(0).getName().
                    toLowerCase();
84          System.out.println(modifier + " class " + statement.getName() + "{");
85          for (AOJAttributeDeclaration att : statement.getMembers().getAttributes())
86                  System.out.print(att.getCodeBody());
87          String hquery = "Select method " + "FROM AOJConstructorDeclaration method "
                + " JOIN method.owner interface" + " WHERE interface.id = " + statement.
                getId();
88          List<AOJConstructorDeclaration> construtores = rh.session.createQuery(hquery
                ).list();
89          for (AOJConstructorDeclaration aoj : construtores)
90              this.generator((AOJMember) aoj);
91          hquery = "Select method " + "FROM AOJMethodDeclaration method " + " JOIN
                method.owner interface" + " WHERE interface.id = " + statement.getId();
92          List<AOJMethodDeclaration> objetos = rh.session.createQuery(hquery).list();
93          for (AOJMethodDeclaration aoj : objetos)
94              this.generator((AOJMember) aoj);
95          System.out.println("}");
96      }
97      public void generator(AOJConstructorDeclaration statement) {
98          System.out.println(statement.getSignature() + "{");
99          orderByCodeOrder(statement.getStatements());
100         for (AOJStatement aoj : statement.getStatements())
101             this.generator((AOJMember) aoj);
102         System.out.println("}");
```

```
103     }
104     private void orderByCodeOrder(List<?> obj) {
105         Collections.sort(obj, new Comparator<Object>() {
106             @Override
107             public int compare(Object arg0, Object arg1) {
108                 AOJMember o1 = (AOJMember) arg0;
109                 AOJMember o2 = (AOJMember) arg1;
110                 return o1.getCodeOrder() < o2.getCodeOrder() ? -1 : (o1.getCodeOrder
                        () > o2.getCodeOrder() ? +1 : 0);
111             }
112         });
113     }
114 }
```