6
A Ferramenta de Medição e Avaliação

Neste capítulo é apresentada a ferramenta AJATO, acrônimo para *Ferramenta de Avaliação AspectJ*[11]. Esta ferramenta permite medir e avaliar sistemas implementados em Java [17] e AspectJ [45]. O principal objetivo da ferramenta AJATO é automatizar as duas primeiras atividades – medição e aplicação de regras – do método de avaliação proposto no Capítulo 4 deste documento. Para a atividade de medição, a ferramenta disponibiliza um conjunto de métricas orientadas a aspectos (OA), especialmente as métricas listadas na Tabela 1 (Subseção 4.2.1). A atividade de avaliação do software é feita após a medição, baseada nos resultados das métricas e utilizando regras heurísticas. As regras heurísticas suportadas pela ferramenta aparecem na Tabela 7 e Tabela 8 do capítulo anterior. É importante ressaltar que a avaliação proposta neste trabalho consiste em sugestões para melhorar a separação de interesses do sistema, e consequentemente, alguns atributos do software como acoplamento e coesão. Além das métricas e regras disponíveis, a ferramenta oferece mecanismos de extensão o que torna possível adicionar novos elementos.

A ferramenta AJATO pode ser usada apenas na fase de implementação, pois uma decisão deste trabalho foi utilizar como artefato de entrada o código fonte dos sistemas. As medições são efetuadas sobre este código e a aplicação das regras sobre o resultado das medições. Esta decisão se justifica pelo fato do código fonte ser a fonte mais confiável de informação do software, especialmente se estas informações são relativas a métricas de produto como é o foco deste trabalho. Apesar disso, extensões da ferramenta são planejadas para suportar avaliação de diagramas UML [5] no nível de projeto detalhado. A motivação para esta possível extensão é antecipar a identificação de problemas no processo de desenvolvimento de tal forma a solucioná-lo antes da fase de implementação. A

[11] O acrônimo é derivado do nome em inglês, *AspectJ Assessment Tool*.

ferramenta proposta é descrita neste capítulo em função das decisões arquiteturais (Seção 6.1) e detalhes de projeto e implementação (Seção 6.2).

6.1. Modelo Arquitetural

Esta seção descreve em mais alto nível os elementos que compõem a ferramenta de medição e avaliação AJATO. Esta ferramenta é composta por quatro módulos principais: Extrator de Modelo AspectJ, Gerenciador de Interesses, Coletor de Métricas e Analisador de Regras. Os módulos e suas dependências são ilustrados no diagrama da Figura 20. Além dos módulos da ferramenta, esta figura apresenta ainda os recursos de entrada e saída de cada módulo.

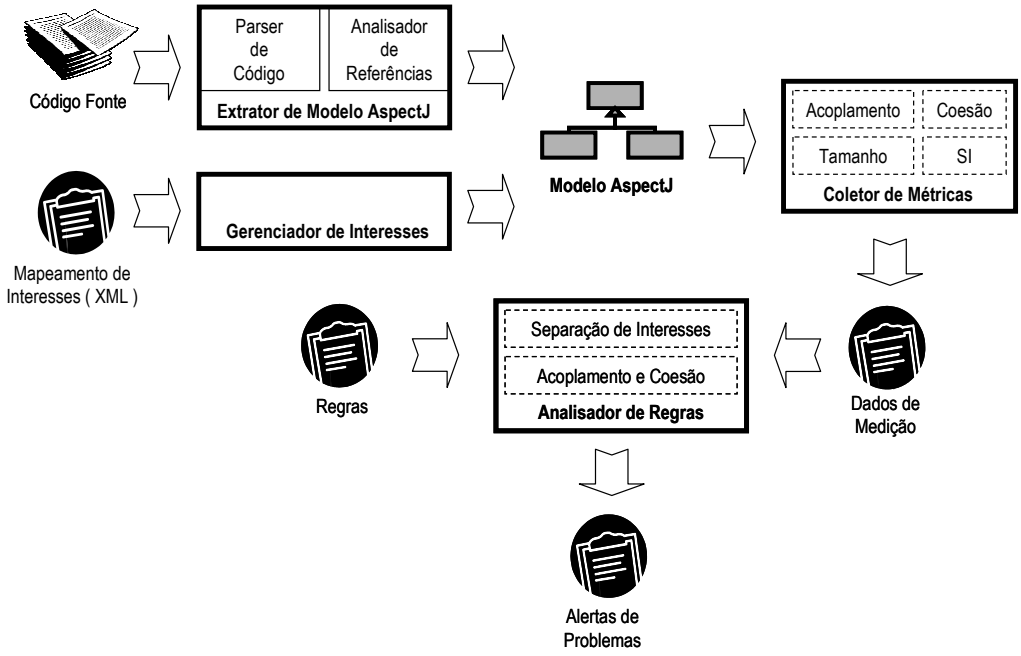

Figura 20 – Representação arquitetural da ferramenta AJATO

A entrada do módulo Extrator de Modelo AspectJ é o código fonte do sistema a ser avaliado e a saída é uma estrutura de dados representativa do sistema que este módulo constrói parcialmente, chamada Modelo AspectJ. O Modelo AspectJ se completa após o módulo Gerenciador de Interesses fazer o mapeamento dos elementos do modelo para os interesses do sistema. A entrada do Gerenciador de Interesses é uma descrição em formato XML [48] [49] deste mapeamento. O Modelo AspectJ, artefato de saída dos dois primeiros módulos, é

a entrada para o módulo Coletor de Métricas. Este terceiro módulo gera os dados das medições que são utilizados pelo Analisador de Regras e este, finalmente, produz um relatório com os alertas de possíveis problemas. Cada um dos quatro módulos da ferramenta é brevemente discutido a seguir em função de seu propósito principal e características.

Extrator de Modelo AspectJ

O módulo Extrator de Modelo AspectJ efetua o *parser* do código fonte de entrada e constrói um modelo representativo do sistema, o Modelo AspectJ (Figura 20). Este módulo detecta a estrutura dos arquivos de entrada em termos dos elementos sintáticos de Java e AspectJ, tais como componentes, atributos, conjuntos de junção, operações e comandos. O Extrator de Modelo é composto de dois submódulos: (i) *Parser* de Código e (ii) Analisador de Referências. O primeiro submódulo utiliza o ambiente de meta-programação MetaJ [54] para auxiliar no processo de extrair as informações do código. Este ambiente permite a manipulação de estruturas sintáticas da árvore de sintaxe abstrata do programa. O segundo submódulo, Analisador de Referências, é responsável por capturar os relacionamentos entre os elementos sintáticos. Exemplos de relacionamentos são importações, herança, associações e chamadas de métodos.

Gerenciador de Interesses

O módulo Gerenciador de Interesses faz o mapeamento de estruturas sintáticas do Modelo AspectJ (construído pelo módulo Extrator de Modelo AspectJ) para os interesses a serem avaliados. Este mapeamento é necessário para que as métricas de separação de interesses (SI) possam ser aplicadas. Desta forma, este módulo recebe como entrada um arquivo XML e então atualiza o modelo com estas informações. O arquivo XML descreve os elementos sintáticos do software e à quais interesses estes elementos pertencem. A atividade exercida pelo módulo Gerenciador de Interesses representa o sombreamento de código (Subseção 3.2.4) e deve preceder as medições de SI.

Coletor de Métricas

O módulo Coletor de Métricas é responsável por efetuar medições no modelo representativo do sistema (Modelo AspectJ). As métricas definidas neste módulo se classificam em quatro categorias principais: acoplamento, coesão, tamanho e separação de interesses. Para computar as métricas das três primeiras categorias não é necessário que tenha sido feito o mapeamento de interesse pelo módulo Gerenciador de Interesses, entretanto, as métricas de SI só podem ser obtidas após este mapeamento. O resultado da aplicação das métricas pode ser gravado em um arquivo de saída, representado na Figura 20 como Dados de Medição.

Analisador de Regras

O último módulo da ferramenta é o Analisador de Regras que tem como propósito principal a aplicação de regras heurísticas como as definidas no Capítulo 5. Estas regras são baseadas em resultados de medições e elaboradas de tal forma a gerarem alertas específicos de possíveis problemas relativos à separação de interesses do sistema. O módulo Analisador de Regras utiliza dois artefatos de entrada: (i) o resultado das métricas coletadas pelo módulo Coletor de Métricas e (ii) um arquivo que define as regras e a ordem em que estas devem ser aplicadas. O resultado da aplicação das regras feita por este módulo é o arquivo de Alertas de Problemas (Figura 20). Este artefato de saída lista os alertas, as regras associadas a eles e os valores resultantes da atividade de medição.

6.2. Projeto e Implementação

O projeto da ferramenta AJATO é orientado a aspectos com implementação em AspectJ. Em sua primeira versão, dois aspectos foram utilizados para separar os interesses de auditoria e persistência. Estes interesses foram implementados em aspectos por serem conhecidamente transversais [44] [45] [66]. A Figura 21 ilustra a tela principal da ferramenta em que podem ser notadas três regiões, divididas em três colunas, para apresentação de informações relativas ao sistema avaliado. As informações da coluna à esquerda da janela encontram-se organizadas em forma de árvore, hierarquizando as estruturas sintáticas do

sistema. A coluna ao centro apresenta informações relativas ao vértice selecionado na árvore e a coluna à direita da janela informa ao usuário da ferramenta sobre a avaliação do sistema.

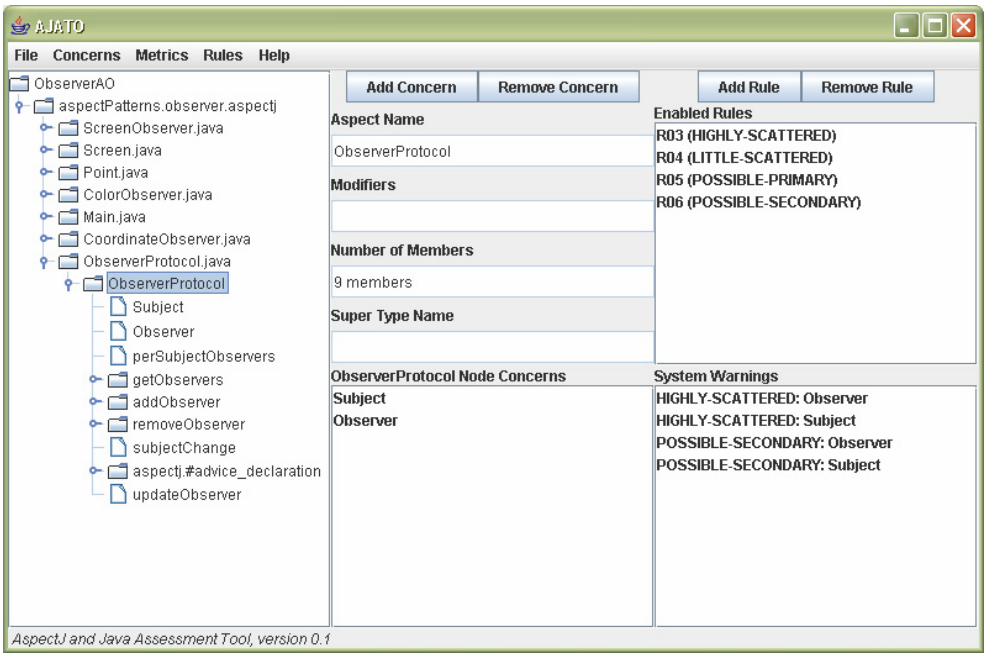

Figura 21 – Interface da ferramenta de avaliação

A hierarquia de estruturas sintáticas, apresentada na coluna esquerda da janela, é organizada da seguinte forma. O sistema como um todo é representado na raiz da árvore, no segundo nível aparecem os pacotes do sistema e no terceiro os arquivos Java e AspectJ. No exemplo da Figura 21 o sistema é chamado "ObserverAO", o único pacote do sistema é chamado "aspectPatterns.observer.aspectj" e existem ainda sete arquivos dentro deste pacote. Internamente aos arquivos encontram-se os componentes que podem ser classes, interfaces ou aspectos. Cada componente pode conter ainda atributos, operações, conjuntos de junção, declarações intertipo ou outros componentes aninhados. As operações de um componente podem possuir comandos. Na Figura 21 o arquivo "ObserverProtocol.java" encontra-se expandido para mostrar as estruturas internas.

As informações apresentadas ao centro da janela da ferramenta são dependentes da estrutura selecionada na árvore à esquerda (Figura 21). Em especial, os interesses que esta estrutura contribui para a implementação (centro inferior da janela). Como o vértice selecionado é o aspecto "ObserverProtocol", as informações apresentadas ao usuário na região ao centro são: o nome do aspecto,

modificadores, número de membros internos, super-aspecto (ou superclasse) e os interesses implementados por este componente. O usuário pode adicionar ou remover um interesse ao elemento selecionado na árvore da esquerda utilizando os botões “Add Concern” e “Remove Concern” no centro superior da janela.

Na coluna direita da janela (Figura 21) são apresentadas informações das regras e do resultado da avaliação do sistema. As regras ativas são mostradas na parte superior e o usuário pode ativar ou desativar regras através dos botões “Add Rule” e “Remove Rule”, respectivamente. Quatro regras se encontram ativas na janela da Figura 21: Regra 03, Regra 04, Regra 05 e Regra 06. Estas regras são descritas na Tabela 7 do Capítulo 5. Os alertas levantados pelas regras ativas são listados na região inferior direita da janela, em nosso exemplo são levantados alertas de elevado espalhamento e possivelmente secundário para os interesses *Observer* e *Subject*.

Além das três colunas que fornecem informações sobre o sistema sendo avaliado, a ferramenta disponibiliza um menu de opções na parte superior da janela. As principais opções deste menu são para carregar um sistema, salvar o mapeamento de interesses, configurar as preferências do usuário e disparar a avaliação heurística. Podem ser definidas preferências para métricas e regras e estas preferências incluem opções de visualização dos resultados e definição de valores limites. O menu de ajuda (*Help*, na Figura 21) oferece instrução para utilização das funcionalidades da ferramenta. Opções de ajuda também estão disponíveis em outras janelas da ferramenta como mostrado na Figura 22. A Figura 22 apresenta a janela da ferramenta para carregar um sistema (esquerda) e a ajuda disponível ao usuário para efetuar esta operação (direita). Esta ajuda é acessível pelo botão “Help” da janela apresentada à esquerda da figura.

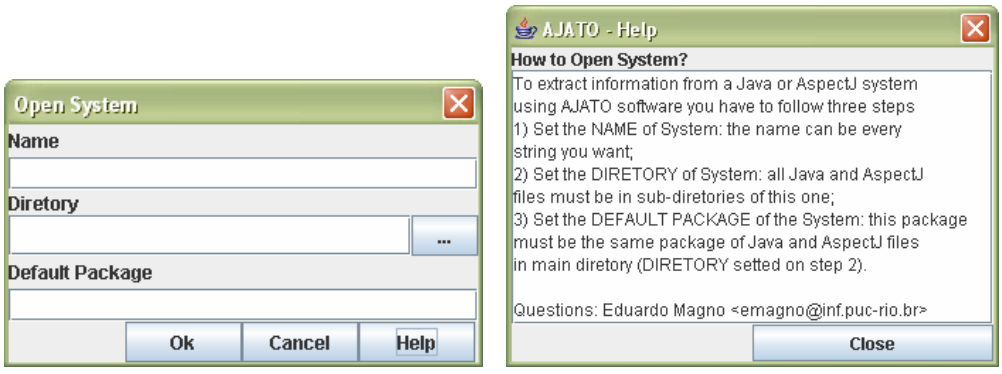

Figura 22 – Janelas para carregar um sistema e respectiva ajuda

A estrutura de diretórios da ferramenta AJATO é apresentada na Figura 23. Esta figura mostra ainda quais diretórios são responsáveis por implementar os quatro módulos (Analisador de Regras, Coletor de Métricas, Gerenciador de Interesses e Extrator de Modelo) e a estrutura de dados Modelo AspectJ da ferramenta. Os arquivos responsáveis pela construção da interface com o usuário da ferramenta estão localizados no diretório "gui". Além disso, a ferramenta utiliza um ambiente de metaprogramação chamado MetaJ [54] para auxiliar o *parser* do código e o caminhamento na árvore de sintaxe abstrata, como discutido na Subseção 6.2.2. Este ambiente é composto por uma biblioteca (metaenv.jar), um arquivo de configuração (plugins.spec) e um arquivo que permite a compilação de *templates* (TemplateCompiler.bat). Estes três arquivos e o diretório de *templates* MetaJ utilizados pela ferramenta também podem ser vistos na Figura 23.

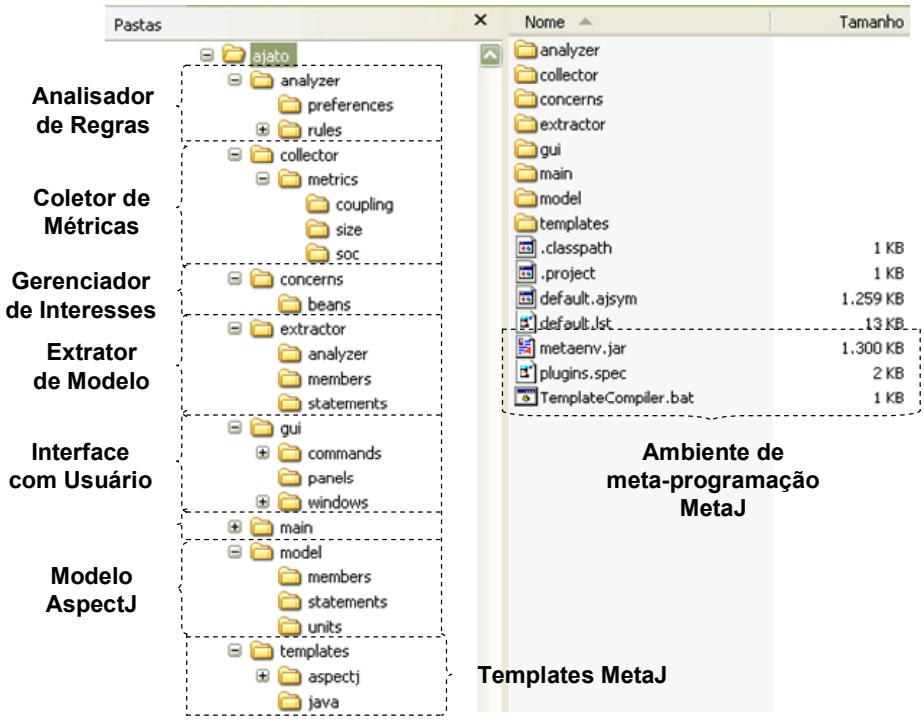

Figura 23 – Estrutura de diretórios da ferramenta AJATO

6.2.1. Modelo de Programas AspectJ

Com o objetivo de facilitar o processo de medição, é construído uma estrutura de dados representativa de programas AspectJ. Nesta estrutura de dados são utilizadas decomposições sucessivas para construção do modelo.

Decomposição [62] é uma técnica adotada para vencer barreiras de complexidade. Esta técnica é freqüentemente utilizada para representar modelos de programas sobre a forma de estrutura de decomposição. Uma estrutura de decomposição é organizada em forma de grafo dirigido no qual cada patamar corresponde a um nível de abstração. Esta estrutura registra as decomposições desde a idéia original até o detalhe mais elementar de sua implementação. Um conceito importante em estruturas de decomposição é a abstração raiz [62]. Abstração raiz é um elemento complexo demais para se dar uma solução direta, sendo dividido em elementos mais simples. Todos os vértices que podem ser decompostos na Figura 24 são considerados abstrações raiz. Quando utilizado recursão em uma decomposição, o problema da abstração raiz pode ser particionado em uma parte a ser resolvida diretamente e em uma parte restante a ser resolvida por recorrência [62]. A Figura 24 apresenta decomposição recursiva nos vértice "Componentes" e "Bloco de Comandos".

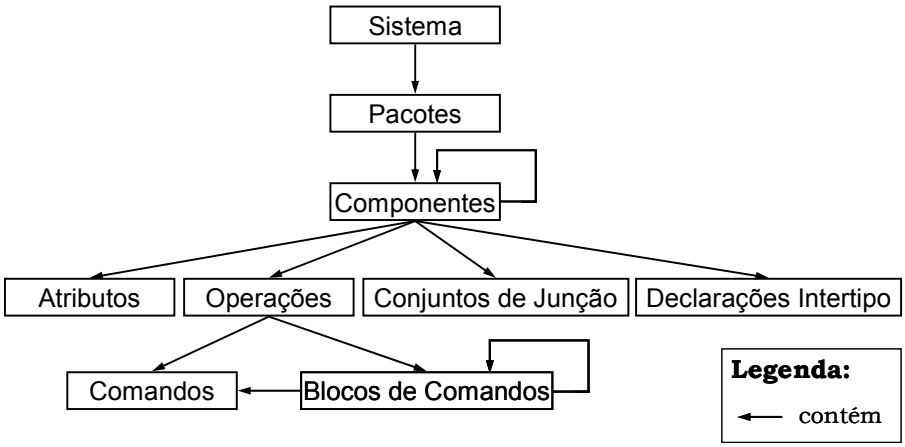

Figura 24 – Modelo de decomposição de sistemas AspectJ

Na Figura 24 é apresentada uma estrutura de decomposição para representar a organização adotada em sistemas implementados na linguagem AspectJ [66]. No nível mais abstrato desta estrutura aparece o sistema como um todo. Em um nível abaixo estão localizados os elementos que representam pacotes existentes no sistema. Assim como em Java, um sistema implementado na linguagem AspectJ deve possuir pelo menos um pacote. Os pacotes podem ser decompostos em componentes (classes, interfaces ou aspectos). Cada componente pode conter outros componentes internos (decomposição recursiva), além de atributos, operações, conjuntos de junção e declarações intertipos. As operações são

decompostas em blocos de comandos ou em comandos. E a decomposição de blocos de comandos pode ser recursiva ou em comandos.

A estrutura de decomposição apresentada na Figura 24 é modelada pelo diagrama de classes da Figura 25. Este diagrama compõe o Modelo AspectJ da ferramenta de medição e avaliação. As classes do diagrama correspondem aos vértices da estrutura de decomposição, exceto por algumas classes e interfaces adicionais que auxiliam na modelagem. Para efeito de simplificação as operações e os atributos foram ocultados no diagrama da Figura 25, bem como os componentes menos relevantes.

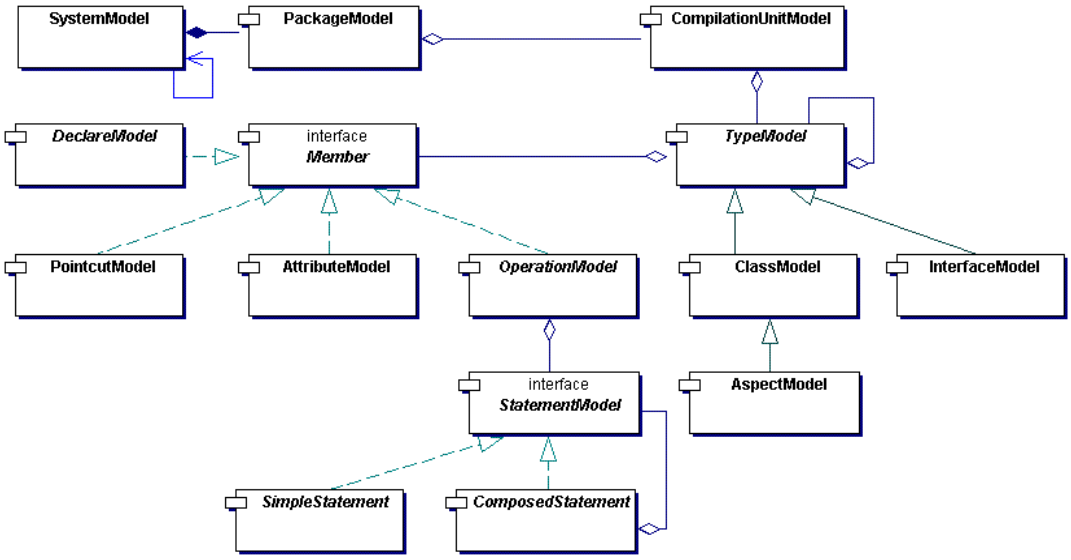

Figura 25 – Diagrama de classes parcial do Modelo AspectJ.

Na Figura 25, a raiz da decomposição é representada pela classe `SystemModel`. Esta classe implementa o padrão *Singleton* [26] para permitir seu acesso a partir de qualquer ponto da aplicação. As decomposições recursivas nos vértices "Componentes" e "Bloco de Comandos" da estrutura de decomposição (Figura 24) são modeladas, respectivamente, pelas classes `TypeModel` e `StatementModel`. Esta última implementa o padrão *Composite* [26] juntamente com as classes `SimpleStatement` e `ComposedStatement` de tal forma a permitir recursão. Outro detalhe interessante do diagrama de classes do Modelo AspectJ é que a classe `AspectModel` é subclasse de `ClassModel`. Note que a primeira classe representa um aspecto e a segunda uma classe do sistema alvo de avaliação. A decisão de estender `ClassModel` para `AspectModel` é coerente porque um aspecto pode possuir todos os elementos de uma classe.

6.2.2.
Módulo de Extração do Modelo

A ferramenta de medição e avaliação AJATO constrói o Modelo AspectJ (Subseção 6.2.1) a partir de informações obtidas do código fonte. Para auxiliar o processo de extrair as informações do código é utilizado MetaJ [54] [55]. MetaJ é um ambiente para meta-programação construído como uma extensão de Java e permite manipular código de programas descritos em qualquer linguagem de programação. Entretanto, na literatura são encontrados experimentos utilizando MetaJ apenas em programas Java. Felizmente, os autores de MetaJ [54] indicam direções para permitir a manipulação de programas escritos em outras linguagens utilizando a gramática da nova linguagem a ser manipulada. Neste trabalho de mestrado é utilizada a gramática de AspectJ para estender MetaJ para esta linguagem. A gramática AspectJ utilizada neste estudo pode ser obtida em [1].

O ambiente MetaJ oferece suporte à análise, transformação e geração de código. Neste trabalho a funcionalidade de análise é utilizada para percorrer o sistema em avaliação de tal forma a construir o Modelo AspectJ. Dois conceitos são importantes para utilização de MetaJ: *templates* e *p-reference*. Os *templates* são abstrações que encapsulam padrões de programas utilizando-se da sintaxe concreta da linguagem objeto (*by example*) [55]. *P-reference* (ou *program-reference*) é um elemento interno aos *templates* que armazena trechos dos programas manipulados. Este elemento esconde a representação interna do código armazenado e oferece apenas operações que garantem a consistência sintática do código. Informações completas sobre a utilização de MetaJ e seus elementos encontram-se disponíveis em [54] [55].

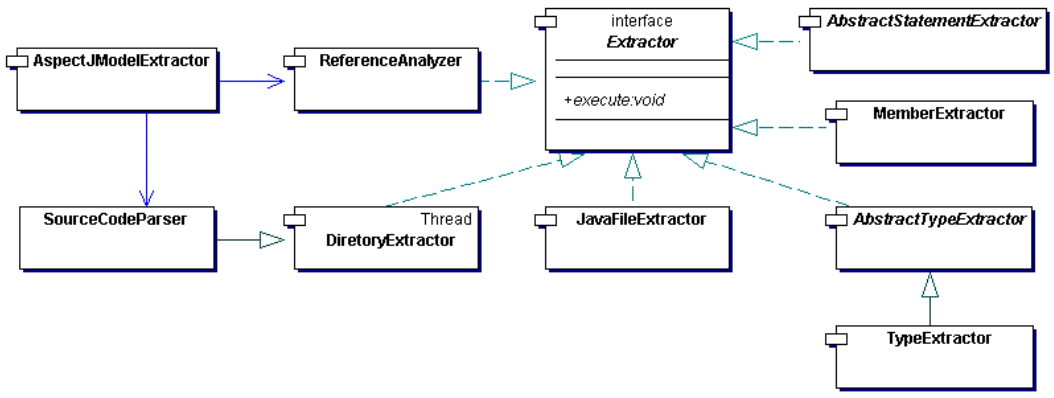

Figura 26 – Diagrama de classes parcial do módulo Extrator de Modelo AspectJ

Além de utilizar MetaJ, o módulo Extrator de Modelo AspectJ é implementado seguindo o padrão de projeto *Interpreter* [26]. A Figura 26 apresenta o diagrama de classes parcial deste módulo. A interface `Extractor` do diagrama desempenha o papel de *AbstractExpression* [26] sendo, portanto, implementada por todas as classes participantes do padrão. Esta interface disponibiliza o método `execute()`, responsável pela interpretação, que deve ser implementado em suas subclasses concretas. As classes `AspectJModelExtractor`, `SourceCodeParser` e `ReferenceAnalyzer` são clientes do padrão *Interpreter* [26]. Elas são responsáveis por iniciar o processo de extração de informação e por criar o Modelo AspectJ. A classe `SourceCodeParser` utiliza MetaJ para efetuar o *parser* do código e a classe `ReferenceAnalyzer` atualiza as referências cruzadas entre as estruturas sintáticas. As demais classes atualizam o Modelo AspectJ com informações obtidas da árvore de sintaxe abstrata do sistema.

6.2.3. Módulo de Gerenciamento de Interesses

O módulo Gerenciador de Interesses da ferramenta AJATO é composto por três componentes principais: as classes `ConcernManager` e `SystemConcerns` e o aspecto `ConcernPersistence`. Estes componentes e seus relacionamentos são apresentados no diagrama de classes da Figura 27.

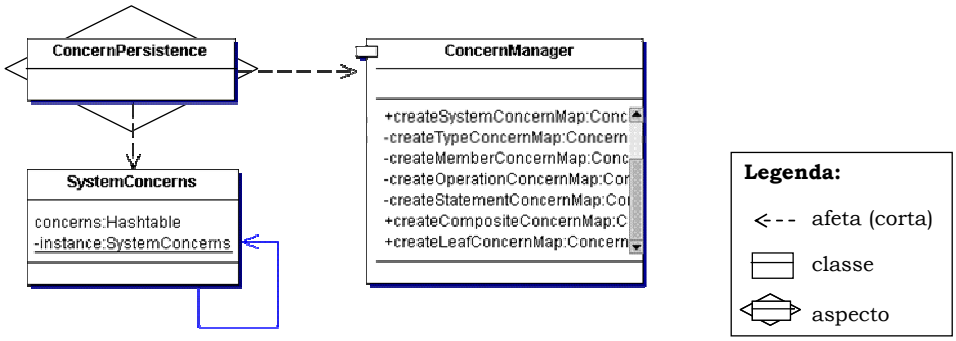

Figura 27 – Diagrama de classes parcial do módulo Gerenciador de Interesses

A classe `ConcernManager` é responsável pelo mapeamento entre as estruturas sintáticas e os interesses do sistema. As estruturas que podem ser mapeadas são todas aquelas que constituem o Modelo AspectJ, ou seja, sistema, pacotes, arquivos, classes, operações, etc. A classe `SystemConcerns` armazena em

uma *hashtable* todos os interesses do sistema e as estruturas que pertencem a cada interesse. Esta classe implementa o padrão *Singleton* [26] de tal forma a ser acessível de qualquer ponto da ferramenta. O terceiro componente deste módulo é o aspecto `ConcernPersistence` que é responsável pelo armazenamento em disco e recuperação do mapeamento de interesses. A decisão de implementar a persistência da ferramenta utilizando aspecto foi tomada porque este interesse (persistência) é tradicionalmente conhecido como transversal [42] [44].

O aspecto `ConcernPersistence` grava e recupera informações de estruturas sintáticas e interesses em um arquivo no formato XML. Este arquivo permite que outras ferramentas interajam com AJATO de tal forma a oferecer extensibilidade para a ferramenta. Para auxiliar na tarefa de gravar e recuperar arquivos XML, o aspecto `ConcernPersistence` utiliza classes do pacote "java.beans" da API de Java [17]. A classe `java.beans.XMLEncoder` é responsável pela conversão de uma estrutura de dados que armazena as estruturas sintáticas e seus respectivos interesses em um arquivo XML. Em contrapartida, a classe `java.beans.XMLDecoder` permite a recuperação das informações do arquivo XML.

6.2.4. Módulo de Medição

As métricas do módulo Coletor de Métricas são implementadas utilizando o padrão *Visitor* [26]. Nesta instância do padrão cada métrica é uma operação executada sobre a estrutura do Modelo AspectJ. A Figura 28 apresenta o diagrama de classes parcial deste módulo da ferramenta. Neste diagrama se pode verificar que a classe `SystemVisitor` é superclasse direta ou indireta de todas as outras classes. Esta classe desempenha o papel de *Visitor* no padrão e disponibiliza os métodos `visit()` para visitar todos os elementos do Modelo AspectJ. As classes concretas (i.e. as métricas) devem re-implementar estes métodos para obter a informação específica que se deseja medir. É importante notar que os componentes que desempenham o papel *Element* [26] do padrão *Visitor* são classes do Modelo AspectJ.

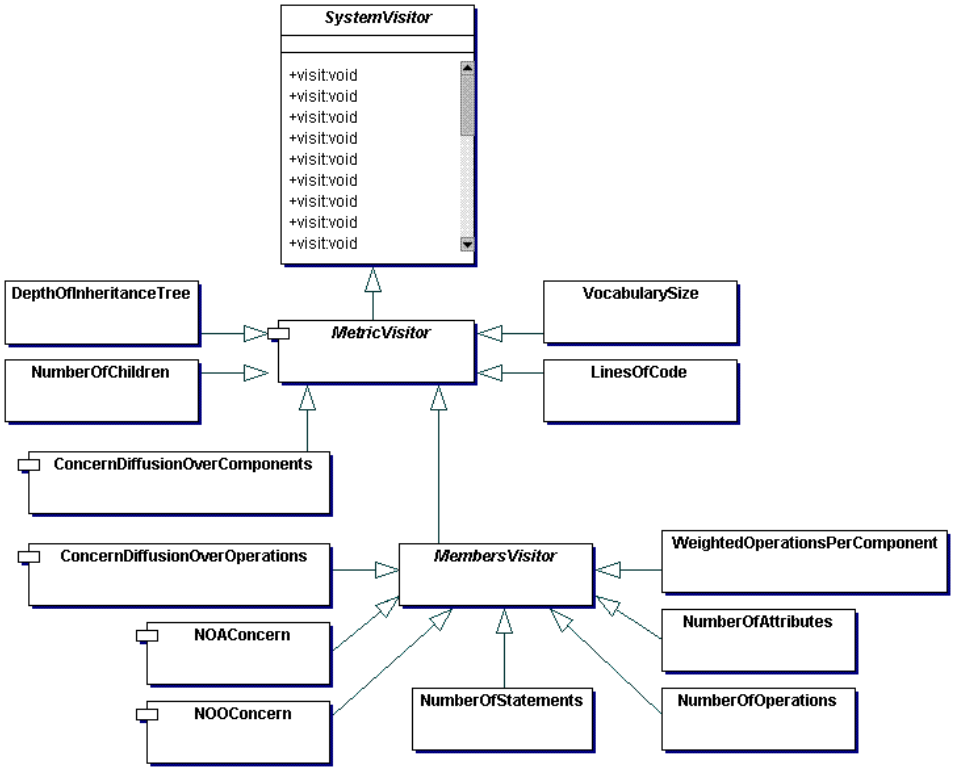

Figura 28 – Diagrama de classes parcial do módulo Coletor de Métricas

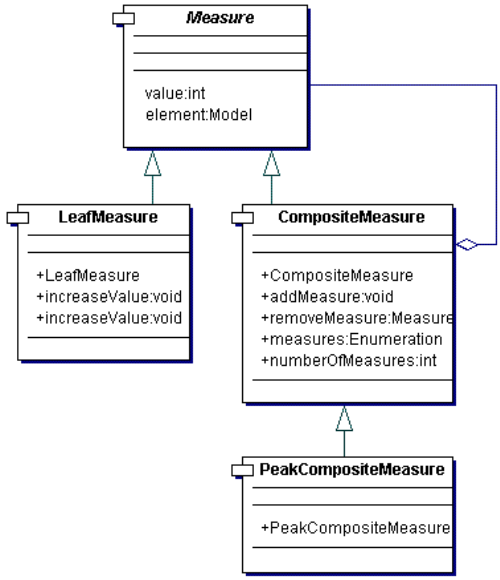

Figura 29 – Diagrama de classes da estrutura de medição

Além das classes que implementam as métricas da ferramenta, este módulo possui uma estrutura de dados para registrar o resultado da medição. Este resultado é armazenado em uma tupla com o elemento do modelo e o valor inteiro medido para este elemento. A estrutura de dados com o resultado da medição implementa o padrão de projeto *Composite* [26] e é ilustrada na Figura 29. A

classe `Measure` desempenha o papel de *Component*, a classe `LeafMeasure` desempenha o papel de *Leaf* e a classe `CompositeMeasure` o papel de *Composite* no padrão. A classe `PeakCompositeMeasure` estende `CompositeMeasure`. A diferença entre elas é que enquanto a primeira retorna o maior valor medido para os elementos da composição, a segunda retorna a soma destes valores.

O módulo Coletor de Métricas disponibiliza uma fachada, padrão *Façade* [26], para localizar em uma única classe o acesso às medições. A classe `MeasurementFacade` implementada a fachada do módulo. Esta classe possui dependência de com a classe `MetricVisitor` (Figura 28) e com todas as classes concretas que implementam métricas. Quando é solicitada qualquer medição através da fachada, o elemento da classe `Measure` retornado corresponde ao resultado da medição.

6.2.5. Módulo de Avaliação Heurística

A classe principal do módulo Analisador de Regras, `RuleAnalyzer`, inicia o processo de avaliação heurística. Esta classe implementa o padrão *Singleton* [26] como apresentado na Figura 30. Além disso, a classe `RuleAnalyzer` possui referência para outras duas classes do módulo: `SoCRuleAnalyzer` e `HeuristicRules`. A classe `SoCRuleAnalyzer` possui um vetor com as regras ativas do sistema, ou seja, aquelas que são utilizadas para alertar o usuário de problemas em interesses transversais. Esta classe possui uma instância de `SoCPreferences` que registra as preferências dos usuários em relação à aplicação das regras. Em especial, possui valores limites definidos para cada regra e um vetor com os interesses que o usuário deseja avaliar no sistema. A classe `HeuristicRules` armazena todas as regras heurísticas (ativas ou não) da ferramenta. As regras que não estão ativas podem ser ativadas pela interface da ferramenta (Figura 21).

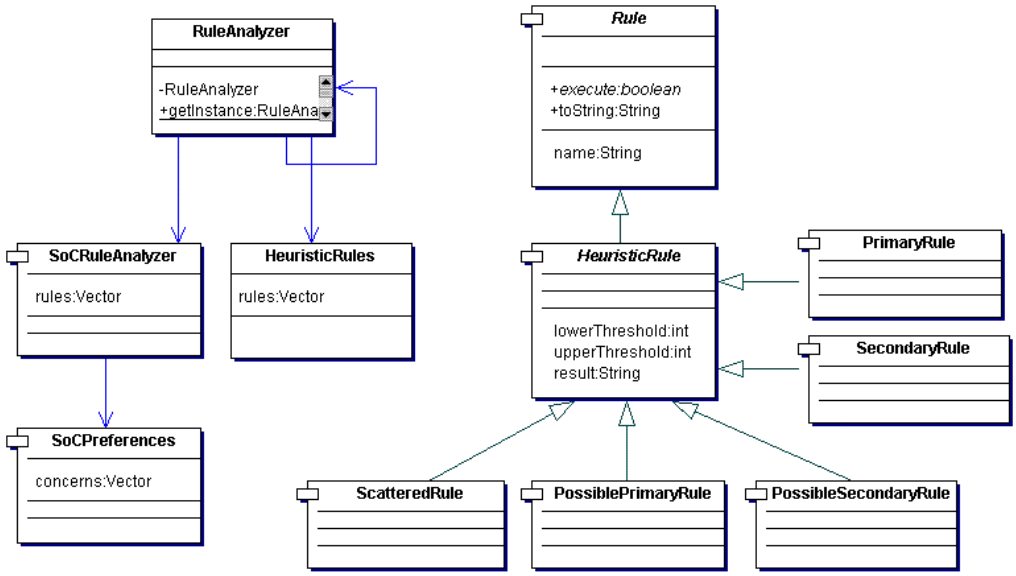

Figura 30 – Diagrama de classes parcial do módulo Analisador de Regras

Além das classes analisadoras, a Figura 30 apresenta também as regras heurísticas implementadas neste módulo. A versão atual da ferramenta disponibiliza regras de separação de interesse (Seção 5.2). As regras de acoplamento e coesão (Seção 5.3) ainda não foram implementadas porque dependem de métricas não disponíveis nesta versão. Como apresentado na Figura 30, a classe abstrata `Rule` é superclasse de todas as regras. Entretanto, as regras heurísticas concretas devem estender da classe `HeuristicRule` (que estende `Rule`). Esta classe define valores limites inferiores e superiores, além de armazenar uma *string* com informação sobre o resultado da aplicação da regra. Regras auxiliares (por exemplo, regras para calcular porcentagem e regras lógicas "e", "ou", "pelo menos um" e "para todos") podem ser subclasses diretas de `Rule`. Para simplificar o diagrama da Figura 30 as regras auxiliares não são mostradas.