5
QCDTool: Uma Ferramenta para Avaliar a Qualidade do *Design* em Modelos

Este capítulo apresenta a ferramenta desenvolvida para apoiar a aplicação, em diagramas de classes, de mecanismos de análise da qualidade em *design* OO. A ferramenta QCDTool (*Quality Class Diagram Tool*) automatiza a aplicação de métricas, estratégia de detecção e modelos da qualidade em diagramas de classes. Em particular, ela automatiza a aplicação do modelo QMOOD (Seção 2.3.1.1) e as estratégias para modelos definidas no Capítulo 4.

5.1.
Visão Geral da Ferramenta

A ferramenta incentiva a importância da engenharia de software dirigida por modelos e permite a análise da qualidade do *design* descrito por modelos UML. Diferentemente das ferramentas já existentes, QCDTool permite a utilização simultânea de diferentes métricas, estratégias de detecção e modelos da qualidade, com o objetivo de alcançar maior abrangência na análise da qualidade e permitir a avaliação do *design* segundo o domínio do problema que eles representam.

Para isto são oferecidos mecanismos de interpretação dos modelos, aplicação e definição de métricas em modelos, aplicação das estratégias de detecção e modelos da qualidade, e visualização e exportação dos resultados.

5.2.
Processo de Funcionamento

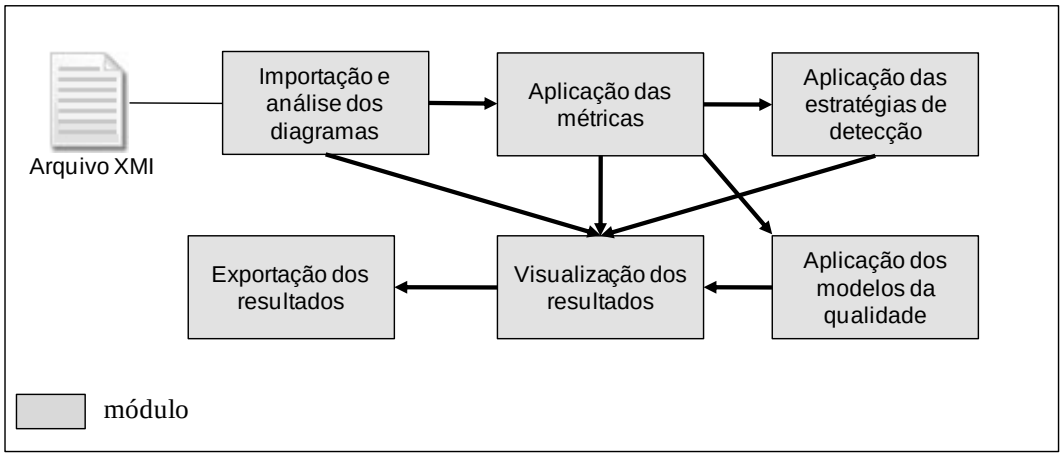

Figura 14 – Processo de funcionamento da ferramenta.

A ferramenta é composta por seis módulos: (i) importação de diagramas de classes, (ii) aplicação das métricas e (iii) aplicação das estratégias de detecção, (iv) aplicação dos modelos da qualidade, (v) visualização dos resultados, e (vi) exportação dos resultados. A Figura 14 apresenta o processo de funcionamento da ferramenta.

O módulo de importação de diagramas de classes recebe como entrada arquivos XMI (2000) (*XML Metadata Interchange*). Estes arquivos descrevem os modelos UML utilizados na representação do *design* dos sistemas a serem avaliados. Os arquivos XMI são normalmente gerados por ferramentas de modelagem UML. QCDTool analisa a informação contida nesses arquivos e gera a representação dos diagramas de classe em uma estrutura de dados apropriada para aplicação das métricas.

Depois de importados os diagramas de classe, o módulo de visualização apresenta a estrutura do *design* do sistema importado. O módulo de aplicação das métricas é responsável por calcular os valores das diversas métricas necessárias para estimar a qualidade do *design*. Esses valores são armazenados de forma que possam ser usados na computação das estratégias de detecção e modelos da qualidade, e são exportados para arquivos de texto.

O módulo de visualização mostra os resultados da aplicação das métricas para cada um dos componentes do diagrama avaliado. Posteriormente, o módulo de aplicação de estratégias de detecção usa os valores das métricas para classificar

as classes do diagrama de acordo com as estratégias de detecção implementadas. O módulo de visualização apresenta uma tabela com os nomes das classes detectadas por cada estratégia implementada.

Depois de aplicadas as métricas e computadas as estratégias de detecção, o módulo de aplicação dos modelos da qualidade utiliza os cálculos coletados para a estimativa dos atributos da qualidade desejados e o módulo de visualização apresenta uma tabela com os cálculos dos fatores e atributos da qualidade. Finalmente, os resultados obtidos (valores das métricas, estratégias de detecção, atributos da qualidade) podem ser exportados para arquivos texto (Anexo B).

5.3. Arquitetura e Implementação

QCDTool foi desenvolvida como uma aplicação *desktop* usando a tecnologia Java e o IDE NetBeans (2007). A Figura 15 apresenta os módulos da ferramenta e as dependências entre eles: Importação de Diagramas de Classes (`DiagramImporter`), Aplicação das Métricas (`MetricsExecutor`), Aplicação das Estratégias de Detecção (`DetectionStrategiesExecutor`), Aplicação dos Modelos da Qualidade (`QualityModelsExecutor`), Visualização dos Resultados (`GUI`) e Exportação dos Resultados (`ResultExporter`).

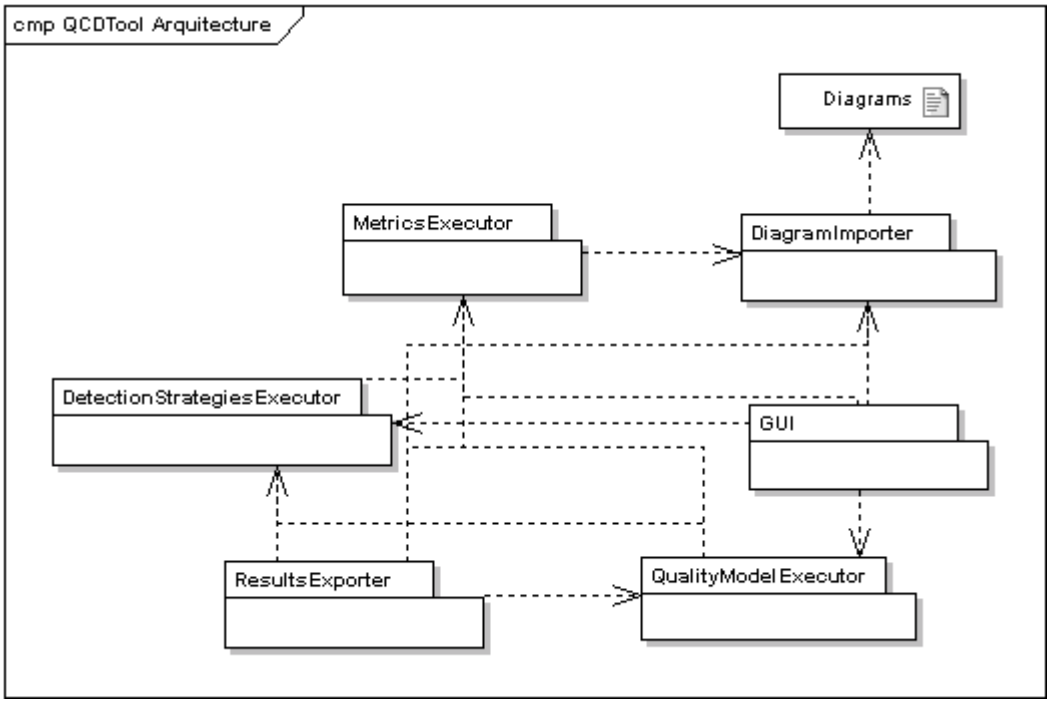

Figura 15 – Diagrama de módulos da ferramenta.

5.3.1.
Módulo de Importação de Diagramas de Classes

O módulo de importação de diagramas de classe é responsável pela importação e análise dos arquivos que contém os modelos UML. A Figura 16 mostra a seqüência de atividades necessária para importar os diagramas de classes (i) análise do arquivo XMI, (ii) geração do arquivo XML correspondente (entidade `Parser` do projeto), (iii) análise do arquivo XML gerado, e (iv) criação da estrutura de dados correspondente.

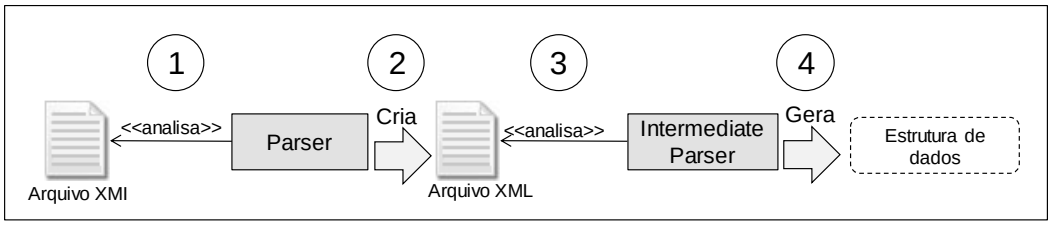

Figura 16 – Atividades da importação dos diagramas de classes.

Os modelos UML, em nosso caso os diagramas de classe, são representados no formato XMI (2000). Este formato foi escolhido devido por ser um padrão definido pela OMG (2003) (*Object Management Group*) para o intercâmbio de informação (metadados) via XML. Pode ser utilizado, também, por qualquer metadado cujo metamodelo possa ser expresso em termos do padrão MOF (2007) (*MetaObject Facility*). XMI define um conjunto de regras que especificam como serializar os elementos de um modelo UML no formato XML. Além disso, a maioria das ferramentas de modelagem UML oferece suporte para a importação e exportação dos modelos em arquivos XMI.

Como os arquivos XMI contêm muita informação referente à visualização dos componentes e outros diagramas UML, que a princípio não estão sendo usados pela QCDTool, foi necessária a filtragem destes arquivos. Além disso, as representações dos modelos UML podem mudar entre as diferentes versões do formato XMI. Portanto, é necessário definir um mecanismo de adaptação entre o formato de representação dos modelos UML e o processamento da informação, de tal forma, que outros formatos de representação dos modelos UML possam ser analisados pela ferramenta.

A entidade `Parser` foi projetada com o objetivo de oferecer a funcionalidade básica a todas entidades encarregadas de fazer a filtragem do conteúdo dos arquivos XMI (passo 1). Como resultado desta filtragem, é criado

um arquivo XML que conterá apenas a informação correspondente aos diagramas de classes (passo 2). A estrutura deste arquivo é definida pela DTD (*Document Type Definition*) apresentada no Anexo A. O arquivo gerado constitui um mecanismo de adaptação entre a representação dos modelos UML e os mecanismos de análise dos modelos implementados na ferramenta. Desta maneira, uma mudança no formato de representação dos modelos UML não influencia o funcionamento da ferramenta. Finalmente, o arquivo XML gerado é analisado pela entidade `IntermediateParser` (passo 3). Esta entidade constrói uma estrutura de dados para representar os modelos UML de tal forma que as métricas e outros mecanismos para a análise da informação dos modelos possam ser aplicados (passo 4). A Figura 17 mostra o projeto detalhado deste módulo.

A classe `DataSource` constitui a estrutura de dados gerada e oferece mecanismos de consulta para que essa informação possa ser utilizada. As entidades `Parser` e `IntermediateParser` foram implementadas utilizando a API para Java SAX (2004) (*Simple API for XML*). A escolha desta API é justificada pelas seguintes razões: (i) é um *parser* simples, (ii) sua definição é baseada em um modelo de eventos, o que permite que sejam analisadas apenas as partes do documento que são de interesse, (iii) o documento não é carregado em memória, e (iv) não são permitidas modificações na estrutura do documento, reduzindo, assim, a necessidade de manutenção.

Os serviços oferecidos pelo módulo de importação dos diagramas podem ser acessados pelos outros módulos da ferramenta por meio da interface `DiagramImportationServices`, a qual é implementada pela classe `DiagramImportationServicesImpl`. A classe `DataSource` armazena toda a informação do arquivo XMI necessária para a análise da qualidade do *design*.

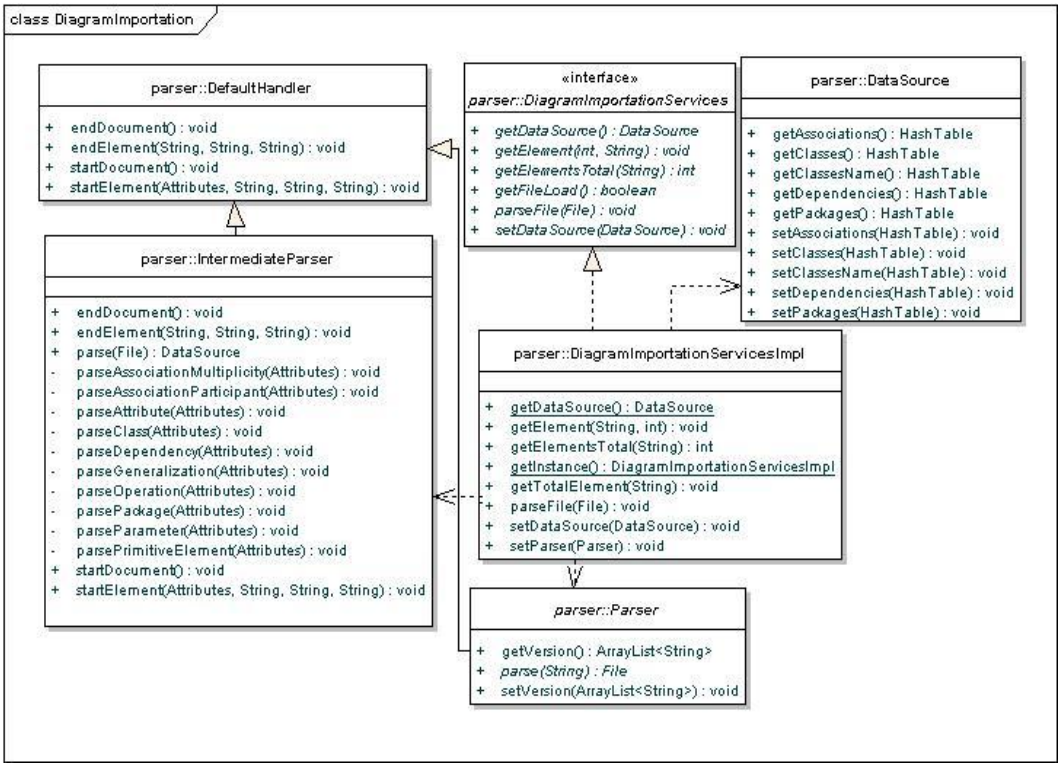

Figura 17 – Módulo de importação dos diagramas.

5.3.2. Módulo de Aplicação das Métricas

Este módulo possui mecanismos para o armazenamento de todas as métricas implementadas. A Figura 18 apresenta o projeto detalhado deste módulo. Todas as classes que definem o comportamento das métricas a serem utilizadas na ferramenta têm que herdar da classe abstrata `Metric`. O construtor da classe `Metric` insere a métrica instanciada na lista de métricas da classe `MetricServicesImpl`. Isto fornece uma forma transparente de armazenamento das métricas e garante que a classe `MetricServicesImpl` possua referências às métricas instanciadas na ferramenta. A funcionalidade da classe `MetricServicesImpl` não é apenas restrita ao armazenamento das métricas. Ela é encarregada de implementar todos os serviços oferecidos pela interface `MetricServices` tais como: computar as métricas, armazenar seus resultados, e mecanismos de consulta para que eles possam ser utilizados pelas técnicas de análise da qualidade e/ou ser exportados para análises futuras.

As métricas implementadas nesta dissertação correspondem às utilizadas no modelo QMOOD e nas estratégias de detecção para modelo apresentadas na Seção 2.3.1.1 e no Capítulo 4. O único padrão de projeto utilizado neste módulo

foi o *Singleton* (Gamma et al, 1995). O objetivo do uso deste padrão foi permitir apenas uma instanciação da classe `MetricServicesImpl` e garantir, desta maneira, que as métricas sejam sempre gerenciadas pela mesma instância da classe `MetricServicesImpl`.

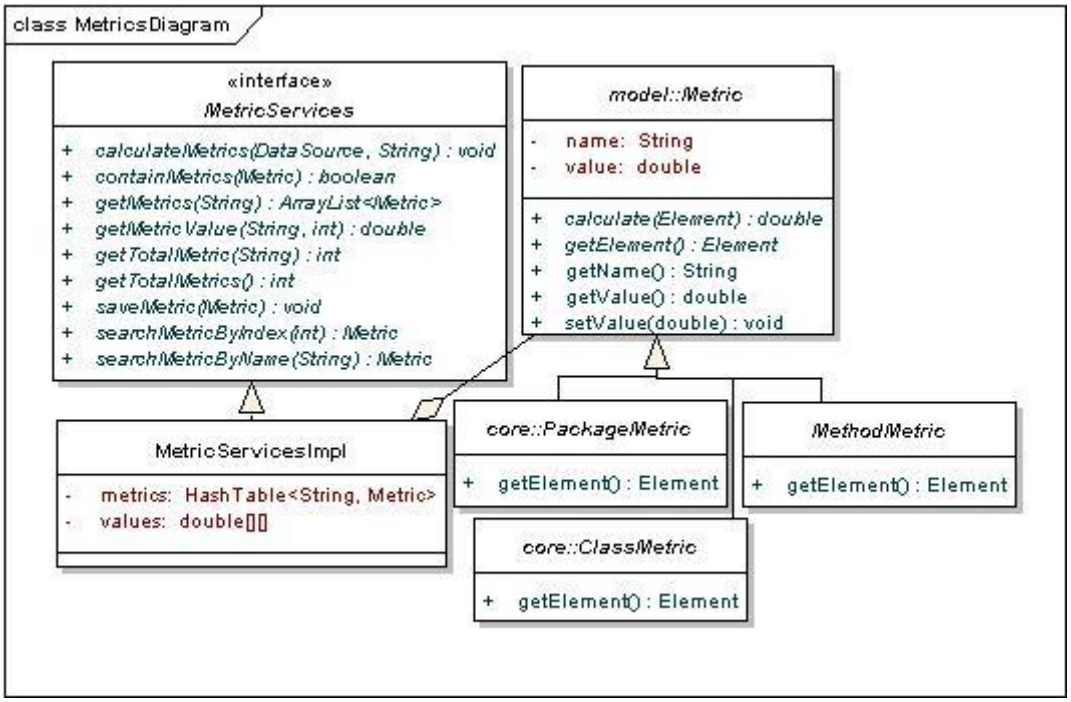

Figura 18 – Representação das métricas.

5.3.3. Aplicação das Estratégias de Detecção

Após a computação das métricas, o usuário pode selecionar a opção de aplicar as estratégias de detecção com o objetivo de identificar os elementos problemáticos do *design*. O módulo de aplicação das estratégias de detecção define um conjunto de classes para a manipulação das estratégias implementadas. A Figura 19 mostra o projeto detalhado para estas classes. A interface `DetectionStrategyServices` expõe os serviços disponibilizados pelo módulo os quais são implementados pela classe `DetectionStrategyServicesImpl`. Entre estes serviços podem ser mencionados: o cômputo das estratégias de detecção definidas, o armazenamento dos resultados obtidos após o cômputo, e os mecanismos de consultas para que os resultados possam ser utilizados e/ou ser exportados para análises futuras. Na Figura 19 pode ser observado o projeto destas duas entidades.

Para implementar as estratégias de detecção, foi necessário implementar os mecanismos de filtragem e os operadores aritméticos para permitir o uso combinado das métricas. Devido ao fato que as estratégias de detecção podem ser compostas por outras estratégias de detecção, decidiu-se utilizar o padrão de projeto *Composite* (Gamma et al, 1995) para modelar sua estrutura. Neste padrão, os elementos contidos em um objeto composto podem ser tratados como se fossem um único objeto. O mecanismo utilizado para compor as estratégias de detecção são os operadores lógicos ou operadores de conjunto (*and* e *or*). A estrutura definida está representada na Figura 19.

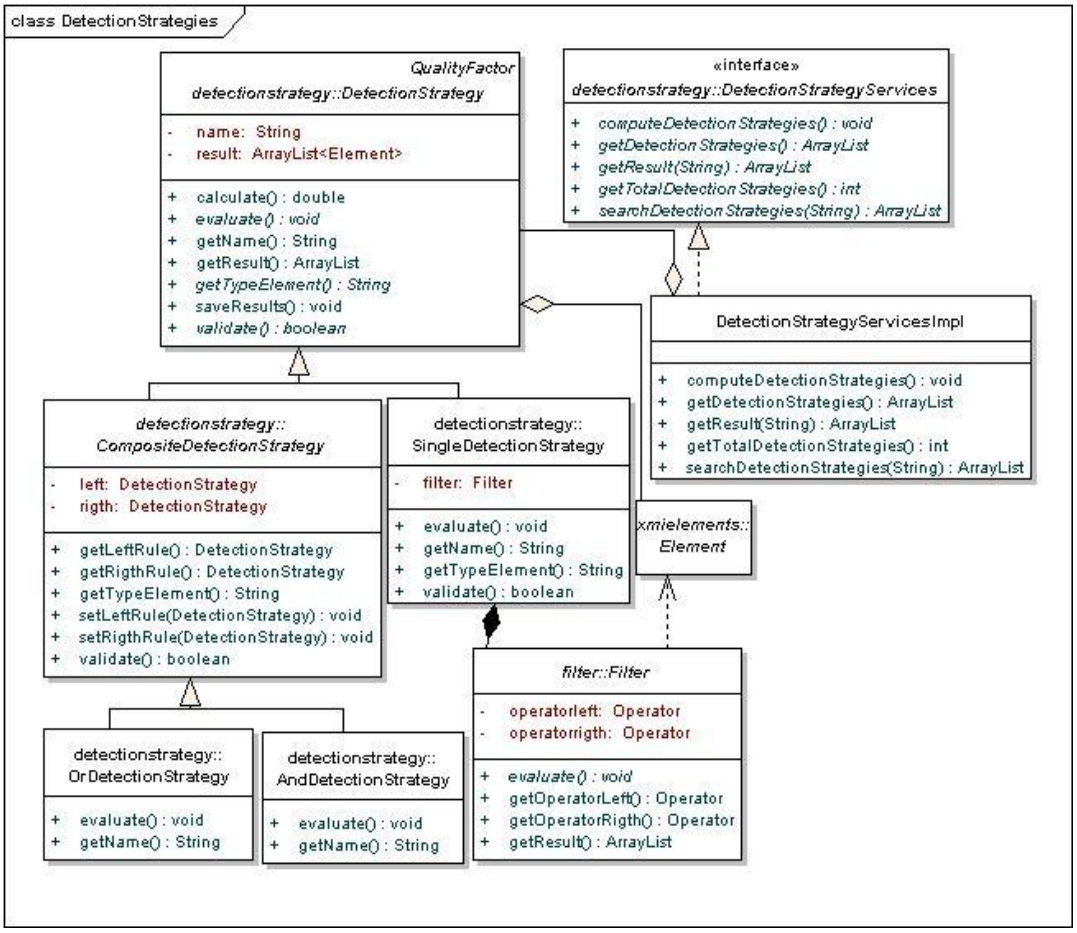

Figura 19 – Representação das estratégias de detecção.

As estratégias de detecção são representadas pela classe `DetectionStrategy` e podem ser computadas por meio do método `calculate`. Na implementação deste método foi utilizado o padrão de projeto *TemplateMethod* (Gamma et al, 1995). A escolha desse padrão é justificada já que, para computar uma estratégia de detecção, é executada uma seqüência de métodos, em que alguns deles são especificados nas subclasses da hierarquia

(Figura 19). Primeiramente a estratégia é validada. Uma estratégia é válida se todas as estratégias que a compõem identificam o mesmo tipo de componente (e.x. pacotes e classes). Esta validação é feita usando o método `validate` que é implementado nas subclasses. Se a estratégia for válida, passa-se à sua execução. Isto é feito por meio do método `evaluate`, implementado por cada uma das classes concretas. Finalmente, os resultados da execução da estratégia de detecção são armazenados por meio do método `saveResults`, definido na classe abstrata `DetectionStrategy`.

A execução dos métodos `validate` e `evaluate` nas estratégias de detecção compostas é feita de maneira recursiva, em que os resultados obtidos são combinados de acordo ao operador lógico especificado na estratégia composta. As estratégias de detecção simples serão consideradas válidas se todas as métricas que utilizam forem aplicadas ao mesmo tipo de componente de *design*. Já na execução do método `evaluate`, cada estratégia de detecção simples delega esta tarefa ao método `evaluate` de seu filtro associado (Figura 20).

Os filtros implementados são filtros marginais de limite semântico definidos por Marinescu e apresentados na Seção 2.3.2. Como pode ser observado na Figura 20, estes filtros estão compostos por dois operadores, que representam: (i) a expressão baseada em métricas a ser avaliada, e (ii) o limite percentual ou absoluto utilizado na filtragem. Por sua vez, o método `evaluate` de cada filtro combina, de acordo com sua semântica, os resultados obtidos após executar o método `evaluate` correspondente a cada um dos operadores associados.

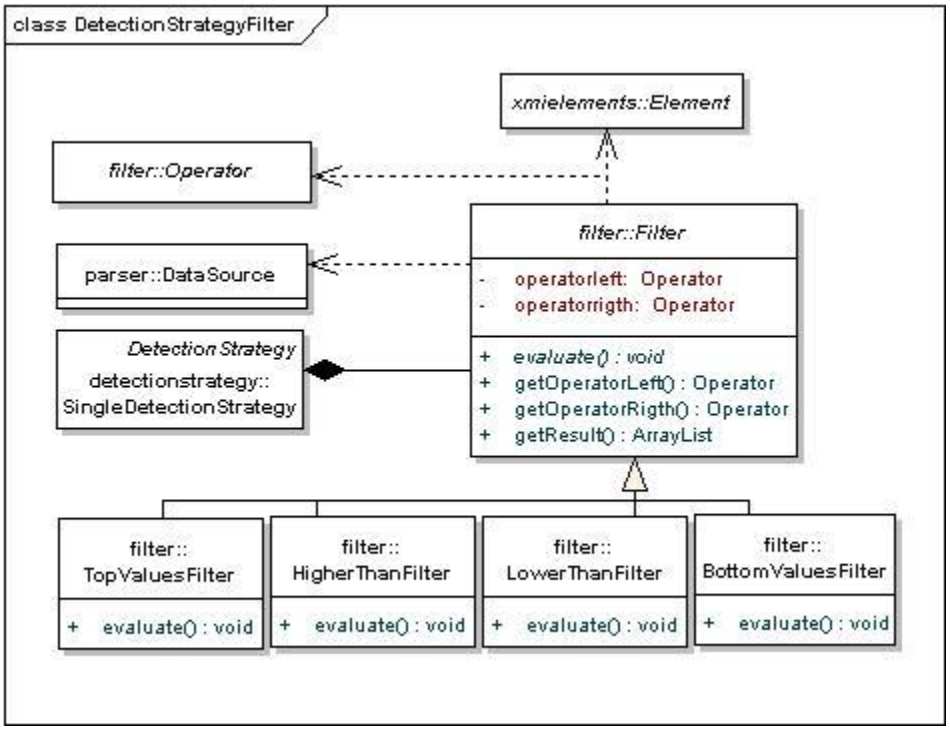

Figura 20 – Representação do mecanismo de filtragem.

O modelo correspondente à estrutura dos operadores é apresentado na Figura 21. Neste caso, também foi utilizado o padrão de projeto *Composite* já que uma instância da classe `Operator` pode estar constituída por outras instâncias dessa classe. A classe `SingleOperator` é utilizada para representar apenas um número, ou uma métrica. Ambas as representações são utilizadas para definir respectivamente, o valor limite dos filtros marginais semânticos e a expressão baseada em métricas que se deseja avaliar nas componentes do *design*. Já instâncias da classe `BinaryOperator` combinam instâncias de `SingleOperator` por meio das funções aritméticas (e.x. soma, divisão). Desta forma permite-se que a expressão baseada em métricas a serem avaliadas possa ser definida de uma maneira mais complexa, combinando também outras métricas e/ou números na sua definição.

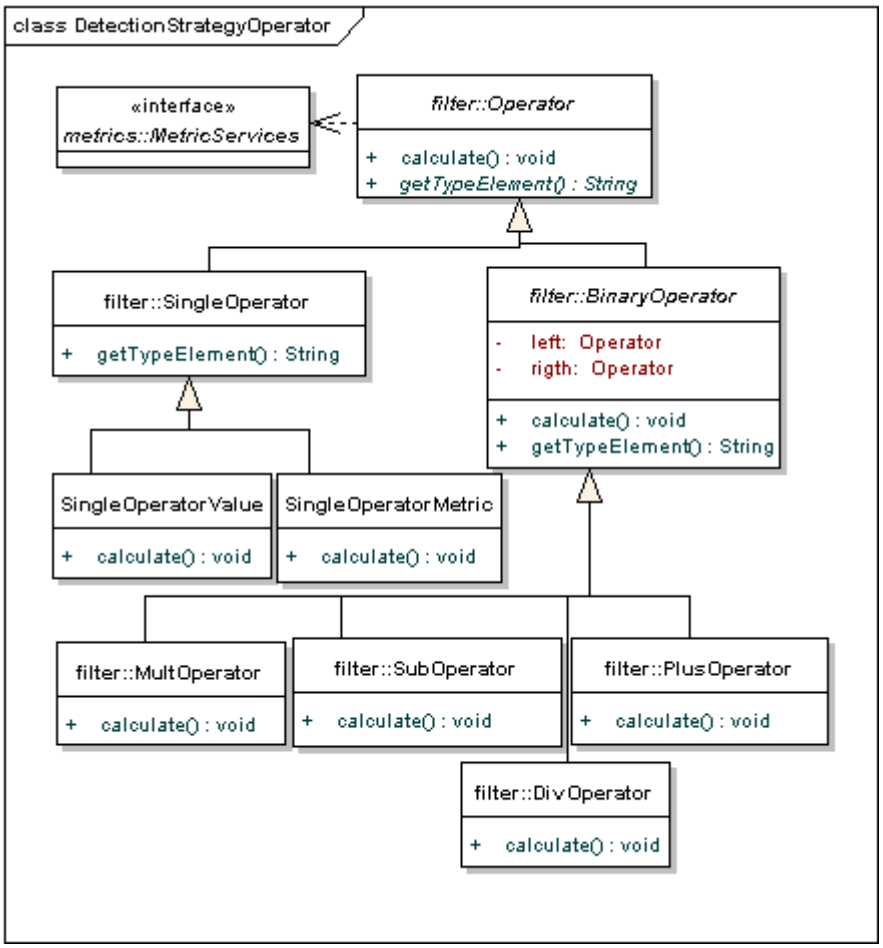

Figura 21 – Estrutura dos operadores.

Como a expressão a ser avaliada por uma estratégia de detecção é baseada em métricas, o método `evaluate`, definido na classe `Operator`, precisa dos componentes do *design* e dos valores das métricas computadas. Estas informações são obtidas por meio dos serviços oferecidos pelos módulos de importação dos diagramas e de aplicação das métricas, utilizando respectivamente, as interfaces `DiagramImportationServices` e `MetricServices`.

5.3.4. Aplicação dos Modelos da Qualidade

O módulo de aplicação de modelos da qualidade tem como objetivo a avaliação da qualidade mediante o uso de modelos definidos com este objetivo. O foco da avaliação deste módulo é a estimativa dos atributos da qualidade especificados em cada um dos modelos utilizados. Os resultados destas estimativas podem ser analisados, levando a reestruturação do *design* com o

objetivo de alcançar as estimativas desejadas. Desta maneira se poderá economizar tempo e esforço nas etapas posteriores.

O usuário pode aplicar os modelos da qualidade implementados na ferramenta, após terem sido computadas as métricas. Os resultados das métricas são utilizados pelos modelos para fazer as estimativas dos fatores da qualidade e conseqüentemente dos atributos da qualidade que realmente interessam aos usuários.

A Figura 22 apresenta o projeto do módulo de aplicação dos modelos da qualidade. Este módulo oferece seus serviços mediante a interface `QualityModelServices`, que é implementada pela classe `QualityModelServicesImpl`. Os modelos da qualidade podem ser previamente configurados, especificando qual atributo da qualidade se deseja maximizar. Também, cada atributo pode ser configurado de acordo com a influência de cada um dos fatores da qualidade associados (e.x. no modelo da qualidade QMOOD os fatores composição e polimorfismo influenciam mais no atributo flexibilidade que outros fatores, como encapsulamento e acoplamento).

A classe `QualityModel` provê uma interface genérica para a representação dos modelos da qualidade. Cada atributo da qualidade (e.x. manutenibilidade, facilidade de compreensão) associado é representado genericamente e manipulado pela classe `QualityAttribute`. Por sua vez, cada atributo da qualidade é estimado utilizando um conjunto de fatores da qualidade (quantidade de defeitos, acoplamento, entre outros) que são representados por meio da classe `QualityFactor`. Para o cálculo dos fatores da qualidade é necessário ter acesso aos valores das métricas para cada uma das componentes do *design*, assim como ao resultado das estratégias de detecção implementadas. Estas informações são acessadas utilizando as interfaces `MetricServices` e `DetectionStrategyServices` definidas, respectivamente, pelos módulos de aplicação das métricas e de aplicação das estratégias de detecção.

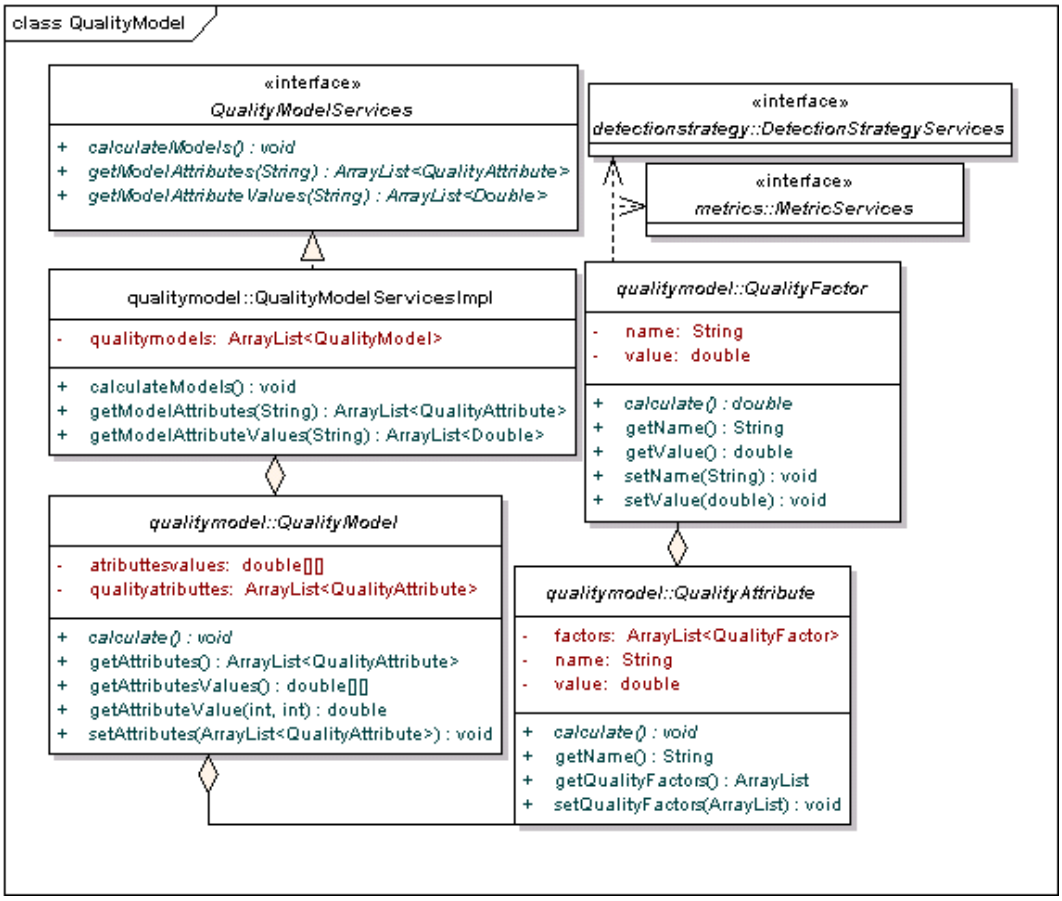

Figura 22 – Modelos da qualidade.

5.3.5. GUI

O módulo de visualização dos resultados é responsável por apresentar uma interface para que o usuário possa realizar as seguintes tarefas: (i) importar os diagramas, (ii) visualização da estrutura e dependências entre as classes dos diagramas, (iii) aplicar as métricas, (iv) aplicar as estratégias de detecção, (v) aplicar os modelos da qualidade, (vi) visualização dos resultados das métricas, estratégias de detecção e modelos da qualidade, e (vii) exportar os resultados da análise.

Após o usuário importar os diagramas, o módulo de visualização representa, utilizando um *treeview*, a distribuição das classes nos pacotes do diagrama importado (Figura 23a). Além disso, são apresentadas, mediante tabelas, as relações (dependência, herança) entre as classes (Figura 24).

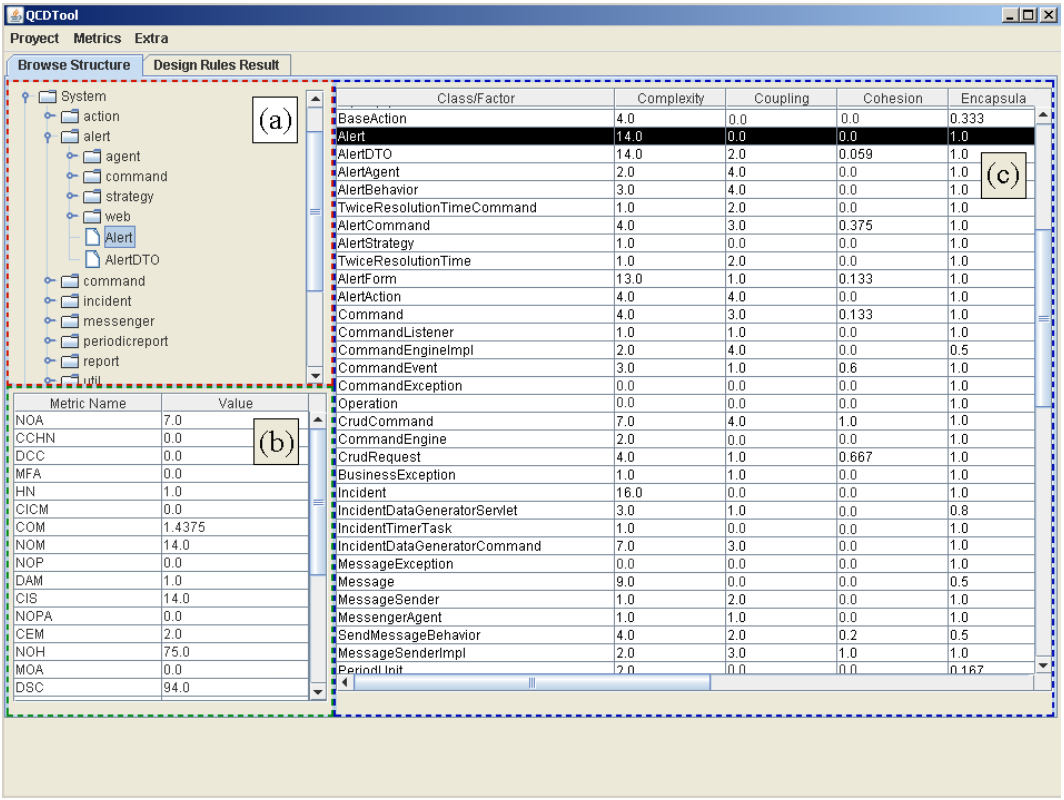

Figura 23 – QDCTool: (a) treeview com classes e pacotes; (b) tabela com resultados das métricas associadas a uma componente; (c) tabela com resultados das propriedades de *design*.

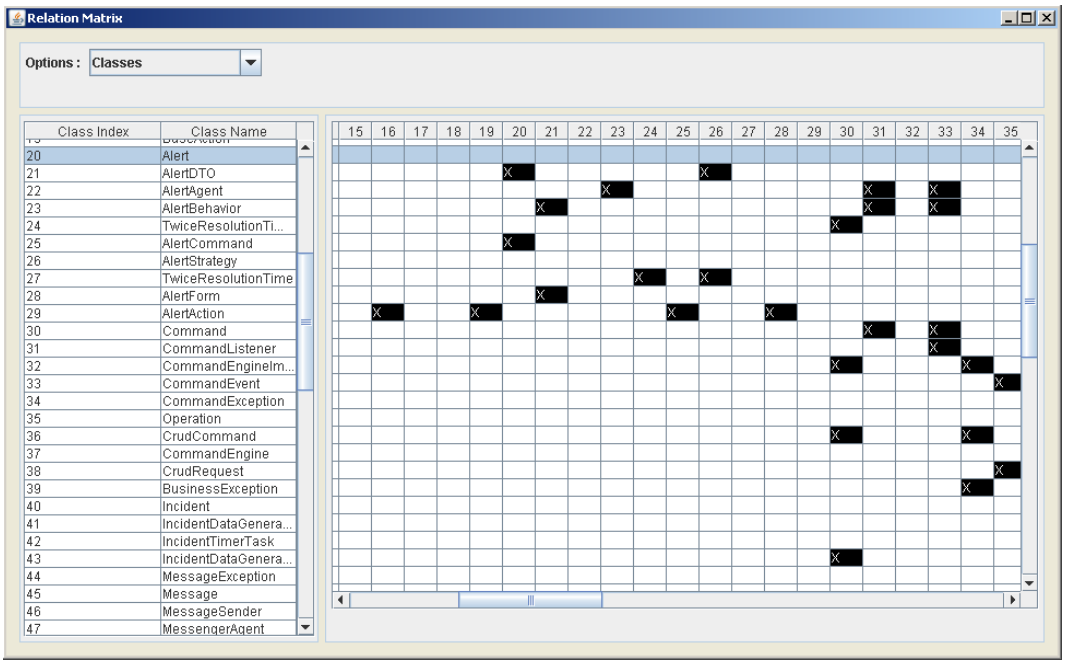

Figura 24 – Relações de dependência, herança e generalização entre as classes.

Depois da aplicação das métricas, o módulo apresenta, respectivamente, mediante tabelas, os resultados das métricas e propriedades do *design*, associados a cada uma das componentes representadas no *treeview*, Figura 23 (b) e (c). Os

componentes de *design* classificados como problemáticos de acordo com as estratégias de detecção são mostrados em uma tabela, associando cada um deles com o problema de *design* correspondente (Figura 25).

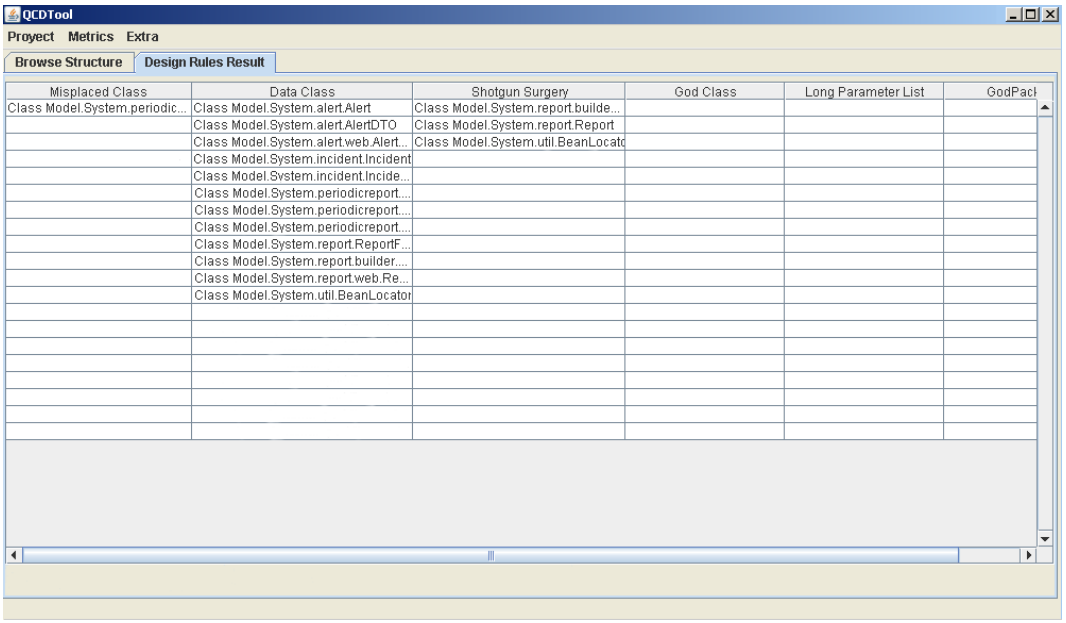

Figura 25 – Componentes de *design* classificados como problemáticos.

5.3.6.Exportação dos Resultados

O módulo de exportação dos resultados tem como objetivo permitir o armazenamento dos resultados obtidos na análise da qualidade realizada. Mediante este módulo os resultados da aplicação das métricas, estratégias de detecção e modelos da qualidade poderão ser exportados em diferentes formatos para serem utilizados em análises futuras.

Na definição da estrutura deste módulo foi utilizado o padrão de projeto *Visitor* (Gamma et al, 1995). Este padrão de projeto representa uma operação a ser executada em elementos de uma estrutura de objetos e permite a definição de uma nova operação sem alterar as classes dos elementos sobre os quais ela opera. A estrutura deste módulo é apresentada na Figura 26. Neste módulo, a entidade `ExportReport` foi projetada para ser alterada raramente e para permitir que diversas operações sejam efetuadas sobre ela (e.x. montar um arquivo texto ou um código HTML a partir dos resultados obtidos após realizar a análise).

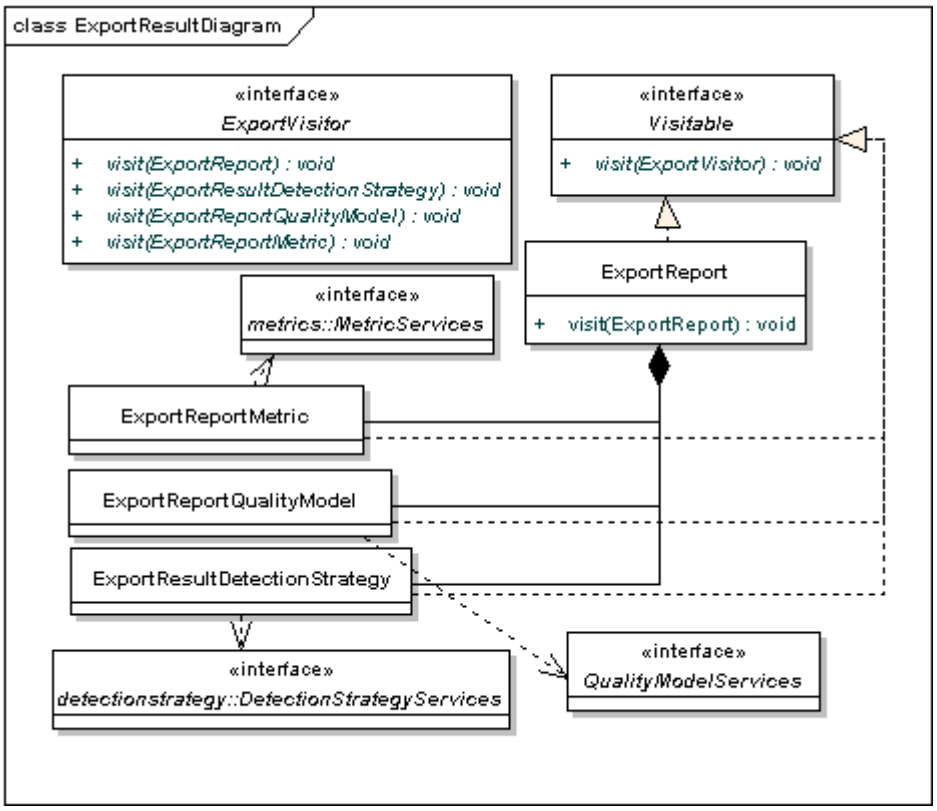

Figura 26 – Estrutura do módulo de exportação dos resultados.

5.4. Pontos Fixos e Pontos de Extensão

A ferramenta QCDTool é considerada um framework, já que novas funcionalidades podem ser inseridas visando ampliar seu escopo de utilidade. Esta seção detalha os pontos fixos e de extensão oferecidos.

As seguintes funcionalidades fazem parte do conjunto de pontos fixos da ferramenta:

- Estrutura de representação da informação do diagrama de classes. A ferramenta dispõe de uma estrutura de dados, em que é armazenada a informação proveniente do arquivo XMI. Esta estrutura pode ser consultada através dos serviços oferecidos pela interface `DiagramImportationServices`.
- Gerenciamento das métricas. O sistema dispõe de mecanismos para o gerenciamento das métricas. Entre estes mecanismos encontram-se o armazenamento dos resultados obtidos, as consultas para a análise dos resultados, entre outras.

- Gerenciamento das estratégias de detecção e modelos da qualidade. De forma semelhante às métricas, o sistema fornece mecanismos para o armazenamento e consultas dos resultados obtidos após executar as estratégias de detecção e modelos da qualidade
- Visualização das informações correspondentes a: representação da estrutura do *design* dos modelos, resultados dos cálculos das métricas, fatores e atributos da qualidade e entidades problemáticas do *design*.

As seguintes funcionalidades fazem parte do conjunto de pontos de extensão da ferramenta:

- Mecanismos para a importação dos diagramas.
- Métricas disponíveis.
- Estratégias de detecção disponíveis.
- Modelos da qualidade disponíveis.
- Mecanismos para a exportação dos resultados da análise.

5.5. Instanciação

A ferramenta é classificada como framework de tipo caixa-branca (*white-box*) já que novas classes devem ser criadas para instanciá-lo. Cada um dos pontos flexíveis da ferramenta está relacionado com alguma(s) classe(s) que deve(m) ser estendida(s) e com alguma(s) interface(s) que deve(m) ser implementada(s). A Tabela 5 apresenta a instanciação da ferramenta realizada para este trabalho de dissertação.

Tabela 5 – Instanciação da ferramenta.

Ponto de Extensão	**Instanciação**
Mecanismo para a importação dos diagramas	Formato XMI versão 1.3
Métricas disponíveis	PS, NOCC, NOCP, PC, DD, CAM, NOM, ATFD
Estratégias de detecção disponíveis	*Long Parameter List*, *Data Class*, *God Class*, *Shotgun Surgery*, *Misplaced Class*, *God Package*
Modelos da qualidade disponíveis	QMOOD (*Quality Model Object Oriented Design*)
Mecanismos para a exportação	Formato texto

As seções a seguir apresentam detalhadamente o conjunto de ações necessárias para realizar a instanciação.

5.5.1. Importação dos Diagramas

Para o ponto flexível de importação dos diagramas, foi escolhido o formato XMI versão 1.3. A princípio, a ferramenta analisa os diagramas de classes representados com essa formatação. Porém a ferramenta pode analisar informações de diagramas representados em outros formatos. A Figura 27 mostra a estrutura detalhada da instanciação realizada no contexto deste trabalho.

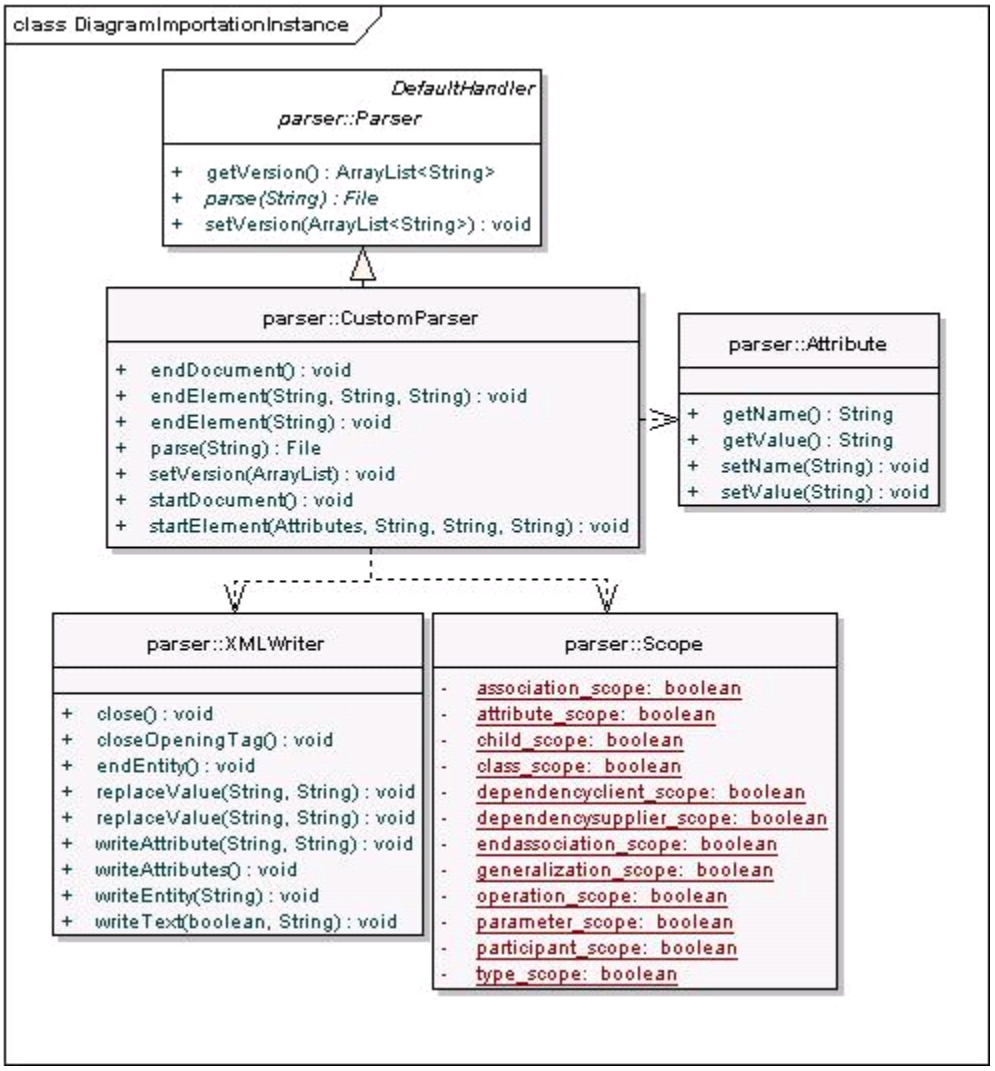

Figura 27 – Instanciação do módulo de importação dos diagramas.

Para incluir na ferramenta mecanismos para a análise de diagramas representados em outros formatos, é necessário definir uma classe que ofereça essa funcionalidade. A classe definida deve herdar da classe `Parser` (Figura 27) e, portanto, deverá implementar as funcionalidades definidas em `DefaultHandler` (pertencente ao conjunto de classes definidas no API SAX) que oferece os mecanismos necessários para analisar um documento XML. Como resultado final dessa análise a nova classe tem que criar um arquivo XML com uma estrutura correspondente ao DTD, apresentado no Anexo A.

Para a instanciação deste ponto de extensão, implementou-se a classe `CustomParser` que é responsável por analisar os documentos XMI versão 1.3 e gerar o arquivo XML com a estrutura necessária. As classes `XMLWriter`, `Scope` e `Attribute` (Figura 27) foram implementadas como ajuda na análise destes arquivos. As classes responsáveis pela instanciação estão destacadas em cinza na **Error! Reference source not found.**. De ai em adiante a ferramenta utiliza seu mecanismo para a análise do arquivo XML gerado e a criação da estrutura de dados que representa essa informação de tal forma que os mecanismos de análise possam ser aplicados.

5.5.2. Definição de Métricas

Como parte de seus pontos fixos, a ferramenta oferece um conjunto de métricas que podem ser utilizadas por qualquer mecanismo de análise da qualidade definido. Porém, embora tenha sido definido este conjunto de métricas como parte da ferramenta, quase sempre torna-se necessária a implementação de novas métricas a serem usadas pelos mecanismos de análise definidos.

Para a inserção de uma nova métrica na ferramenta, é necessário herdar da classe `MethodMetric`, `ClassMetric` ou `PackageMetric` (Figura 28), para métricas a serem aplicadas, respectivamente, sobre métodos, classes ou pacotes. Se a nova métrica se aplicar a outro tipo de entidade do *design* diferente de método, classe ou pacote, uma classe referente a esse tipo de entidade dever ser criada também. A classe da nova métrica deve herdar dessa nova classe. Por exemplo, se for necessário criar uma métrica que se aplique a relações entre classes, deve-se criar uma classe chamada, por exemplo, de `RelationMetric`.

Esta classe deve herdar da classe `Metric`. A classe que implementa a métrica deve, por sua vez, herdar da classe `RelationMetric`.

Neste trabalho, para a implementação das métricas utilizadas pelas estratégias de detecção e o modelo da qualidade foi necessária a implementação de diferentes classes. Estas classes estão destacas em cinza na Figura 28.

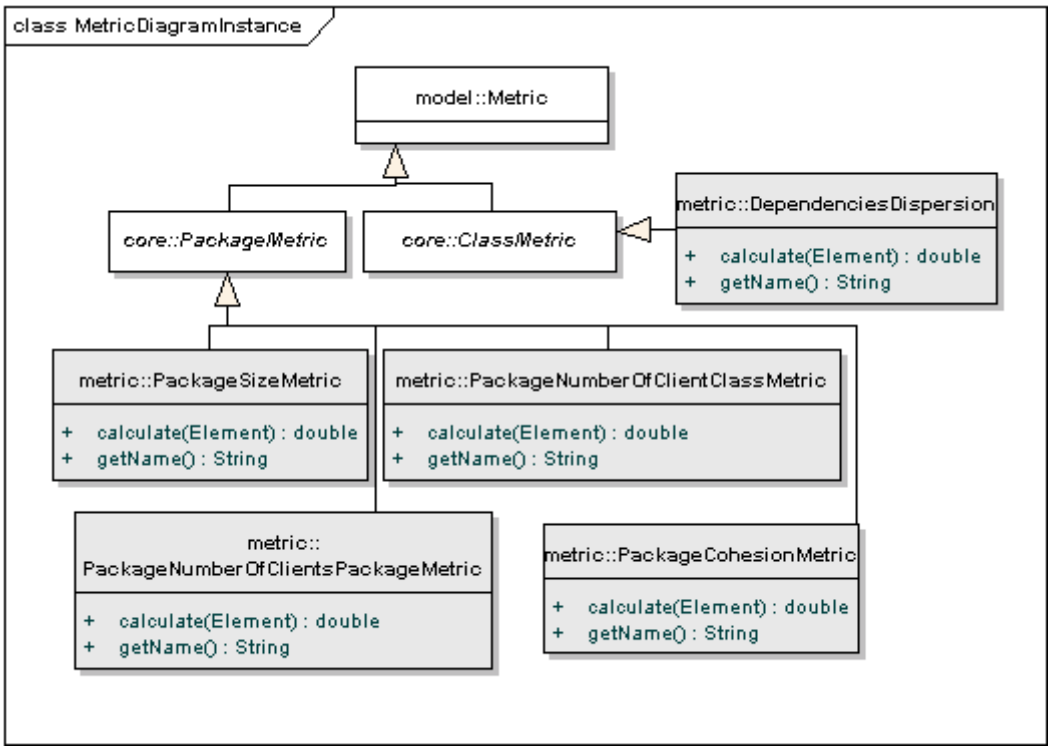

Figura 28 – Instanciação das métricas.

5.5.3. Definição de Estratégias de Detecção

Um dos principais objetivos da ferramenta é a detecção de problemas no *design* OO. Para isto, foi inserida no conjunto dos pontos flexíveis a definição de novas estratégias de detecção. Para incluir uma estratégia de detecção na ferramenta, é necessária a implementação de uma classe referente à estratégia de detecção desejada. A classe da nova estratégia de detecção deve herdar de alguma das seguintes classes: `AndDetectionStrategy`, `OrDetectionStrategy` ou `SingleDetectionStrategy` (Figura 29). As classes criadas para implementar as estratégias de detecção propostas para modelo (Capítulo 4) são destacadas em tom cinza na Figura 29.

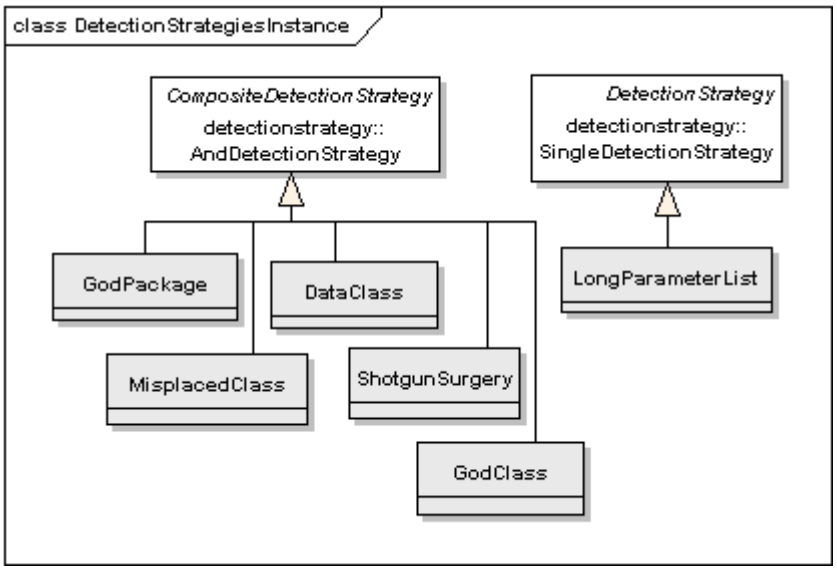

Figura 29 – Instanciação das estratégias de detecção.

5.5.4. Definição de Modelos da Qualidade

Outro objetivo da ferramenta é permitir a utilização de diferentes modelos da qualidade para avaliar os diagramas. Porém, para incluir novos modelos da qualidade na ferramenta, é necessária a implementação de algumas classes. As estruturas destas classes são apresentadas na Figura 30. Primeiramente, a classe que representa a lógica do novo modelo tem que herdar da classe `QualityModel`. Segundo, as classes implementadas com o objetivo de representação as funcionalidades e características próprias dos atributos da qualidade do novo modelo, têm que herdar da classe `QualityAttribute`. Finalmente, os fatores da qualidade associados aos fatores da qualidade definidos deverão herdar da classe `QualityFactor`.

Para a instanciação do modelo QMOOD, realizada neste trabalho, foi necessário implementar as seguintes classes: `QMOODQualityModel`, `QMOODQualityAttribute` e `QMOODQualityFactor`. A Figura 30 apresenta a estrutura destas classes destacadas em cinza.

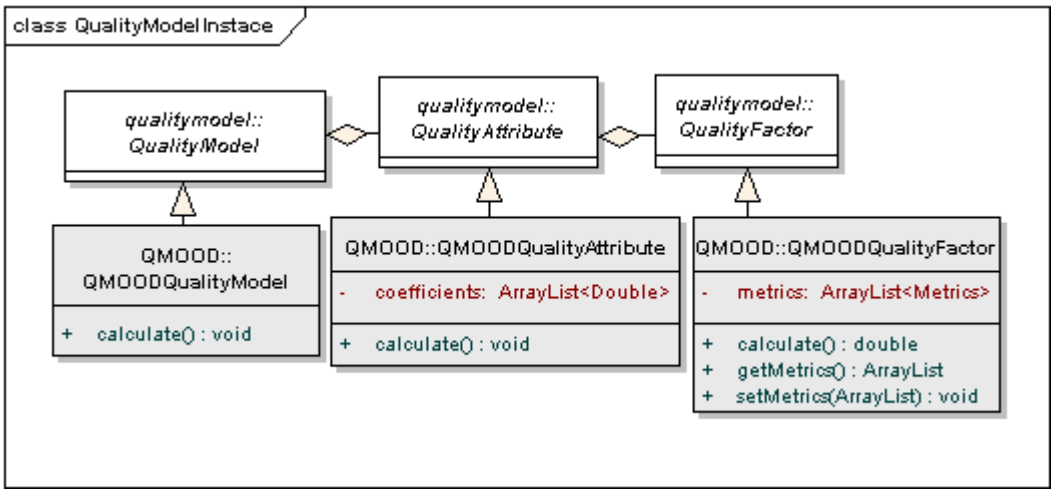

Figura 30 – Instanciação do modelo QMOOD.

5.5.5. Exportação dos Resultados da Análise

Uma das funcionalidades que a ferramenta oferece é a exportação dos resultados obtidos na análise da qualidade em diferentes formatos. Porém, para permitir a exportação para outros formatos é necessária a implementação da classe que ofereça esta funcionalidade. A Figura 31 apresenta a estrutura das classes envolvidas neste módulo. Como na implementação foi utilizado o padrão *Visitor* (Gamma et al, 1999), a classe responsável pela exportação ao novo formato tem que implementar a interface `ExportVisitor`. Em particular, na instanciação realizada deste módulo foi implementada a operação para montar um arquivo texto com os resultados da aplicação das métricas, estratégias de detecção e modelos da qualidade. A classe responsável por esta funcionalidade está destacada em cinza na Figura 31.

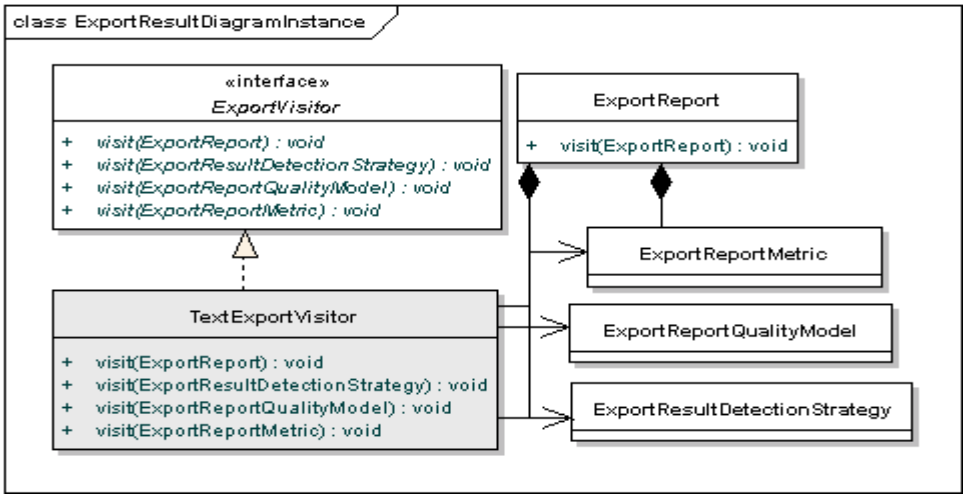

Figura 31 – Instanciação do módulo de exportação.