Automatização da Aplicação de Métricas Arquiteturais Convencionais e Sensíveis a Interesses

Willian Nalepa Oizumi, Thelma Elita Colanzi

Departamento de Informática – Universidade Estadual de Maringá (UEM)
Caixa Postal 87020-900 – Maringá – PR – Brazil

oizumi.willian@gmail.com, thelma@din.uem.br

Abstract. *The software architecture it is one of the most important artifacts in a software project. Therefore, it is necessary to analyze it in order to identify possible design flaws. This analysis can be performed with the support of conventional architectural metrics (CAM), which aims to analyze issues such as cohesion and coupling, and concern-driven metrics (CDM) that aims to identify design flaws related to architectural concerns. The realization of the measurement process manually can be slow and error prone. Thus, this paper presents the configuration of SDMetrics tool to automate the measurement process of CAM and CDM.*

Resumo. *A arquitetura de software é um dos artefatos mais importantes em um projeto de software. Por isso é necessário analisá-la a fim de identificar eventuais falhas arquiteturais. Essa análise pode ser realizada com o apoio de métricas arquiteturais convencionais (MAC), que buscam analisar questões como coesão e acoplamento, e de métricas sensíveis a interesses (MSI) que tem como objetivo identificar falhas de projeto relacionadas a interesses arquiteturais. A realização do processo de medição manualmente pode ser demorado e sujeito a erros. Sendo assim, este trabalho apresenta a configuração da ferramenta SDMetrics para automatizar o processo de medição de MAC e MSI.*

1. Introdução

Uma arquitetura de software é o conjunto das principais decisões arquiteturais que orientam um sistema de software [Bass, Clements e Kazman 2003]. Em um projeto de software ela influencia vários atributos, como performance, robustez, manutenibilidade e distribuição [Bosch 2000]. Ela desempenha um papel central no desenvolvimento e evolução de um projeto de software. Assim, a arquitetura constitui um dos artefatos mais importantes para gerar um produto de software com sucesso.

A identificação de anomalias em arquiteturas é imprescindível a fim de evitar que as falhas de projeto arquitetural sejam propagadas ao longo do ciclo de vida de um software. A fim de garantir a reutilização de componentes, de facilitar o gerenciamento do projeto e de minimizar os recursos necessários para a manutenção de software, é importante que os componentes da arquitetura sejam minimamente acoplados. Desta forma, mudanças em um componente teriam pouco ou nenhum impacto em outros componentes do produto de software. Por esse motivo, o conceito de modularidade se torna essencial para o desenvolvimento de software, e deve ser priorizada desde a definição da arquitetura inicial.

As métricas arquiteturais de software fornecem indicadores de modularidade e são definidas com o objetivo de medir as abstrações relacionadas a componentes e interfaces [Sant'Anna et al. 2003]. Portanto, elas conseguem revelar questões como: acoplamento e coesão de componentes e complexidade de uma interface.

Muitas dos interesses presentes em um projeto de software são inerentemente transversais, isto é, afetam o escopo do software em diversos pontos. A complexidade do projeto arquitetural pode aumentar em conseqüência de interesses transversais e muitas vezes, esses interesses não são modularizados adequadamente. Sant'Anna et al. [2007] afirmam que as métricas arquiteturais convencionais (MAC) não são sensíveis a interesses arquiteturais e, portanto, não conseguem auxiliar o arquiteto de software a identificar anomalias de modularidade relacionadas a interesses transversais. Assim, a aplicação de conceitos da orientação a aspectos (OA) [Kiczales, 1997] torna-se fundamental em projetos de software. Esses mesmos autores destacam a necessidade de técnicas de avaliação que permitam uma identificação efetiva de anomalias de modularidade referentes a interesses transversais no contexto de projeto arquitetural.

A abordagem OA tem propiciado uma forma de isolar os interesses transversais em módulos denominados aspectos. Com isso, o projeto arquitetural OA tem se tornado mais evidente por facilitar o projeto de modularidade. Porém, como a OA introduz novos mecanismos e abstrações que propiciam diferentes maneiras de decompor o projeto de um software, o uso inadequado dos seus mecanismos pode introduzir anomalias exclusivas de modularidade, além de anomalias similares às encontradas em sistemas OO [Bertrán, Garcia e von Staa 2009]. Desta forma, é preciso dispor de mecanismos que auxiliem o arquiteto de software na tentativa de identificar possíveis anomalias de modularidade. Nesse sentido, Sant'Anna [2008] propôs um conjunto de métricas arquiteturais sensíveis a interesses (MSI) para avaliar arquiteturas de software OA. Ele realizou estudos empíricos aplicando o seu conjunto de métricas em arquiteturas de software construídas a partir do código-fonte, ou seja, usando engenharia reversa. O resultado dos seus estudos foi positivo e mostrou que as MSI são efetivas na detecção de anomalias de modularidade em arquiteturas de software OA construídas em um processo de engenharia reversa.

Para a realização de novos estudos empíricos utilizando as MSI e futura adoção delas na indústria, é imprescindível contar com apoio automatizado para o processo de medição. No entanto, o protótipo desenvolvido por Sant'Anna [2008] não se encontra operacional. A SDMetrics [Wüst 2010] é uma ferramenta para medição de projeto orientado a objetos para UML. Ela pode ser configurada para aplicar algumas de suas métricas convencionais no contexto de projeto arquitetural. Além disso, é possível definir novas métricas na SDMetrics para serem aplicadas em artefatos UML.

Visando à automatização das MSI e a realização de experimentos que utilizem MAC e MSI, a SDMetrics foi configurada para automatizar a medição dos dois conjuntos de métricas arquiteturais. O objetivo deste trabalho é apresentar como algumas MAC foram adaptadas na SDMetrics e como as MSI foram inseridas nessa ferramenta.

Este trabalho foi realizado no contexto de um projeto de pesquisa cujo foco principal é a avaliação de arquiteturas de Linhas de Produto de Software (LPS) [van der Linden, Schmid, Rommes 2007]. Nesse projeto de pesquisa pretende-se realizar experimentos para comparar os dois conjuntos de métricas arquiteturais: MAC e MSI. A

inserção das MSI e adaptação de algumas MAC na SDMetrics viabiliza a automatização da medição de arquiteturas de LPS, e de outros tipos de arquitetura, usando os dois conjuntos de métricas.

Este artigo está organizado em seis seções. A segunda seção contém detalhes referentes às métricas arquiteturais utilizadas neste trabalho. O funcionamento da SDMetrics é explicado na terceira seção. Na Seção 4 estão as informações referentes ao processo de automatização e validação das métricas por meio da SDMetrics. Os resultados e discussões são detalhados na Seção 5. Por fim, na Seção 6 é apresentada a conclusão do trabalho.

2. Métricas Arquiteturais

Esta seção está dividida em duas subseções. Na primeira é apresentado o conjunto de MAC constituído por métricas de acoplamento [Martin 1997], de coesão [Martin 2003] e de herança [Briand 1994] que foi adaptado na SDMetrics para o contexto de projeto arquitetural. Na segunda subseção, é apresentado o conjunto de MSI [Sant'Anna 2008] que foi definido na SDMetrics.

2.1. Métricas arquiteturais convencionais

2.1.1. Métricas de acoplamento de Martin [1997]

As métricas de acoplamento definidas por Martin [1997] têm como principal objetivo mensurar a qualidade de projetos orientados a objetos. Ele estabeleceu um padrão de projeto desejado que está diretamente relacionado com a estabilidade, pois ela é o cerne de todo o projeto de software. Os princípios que norteiam esse padrão são: princípio das dependências estáveis e princípio das abstrações estáveis.

O princípio das dependências estáveis diz que as dependências entre pacotes em um projeto devem ser direcionadas para a estabilidade dos pacotes. Um pacote deve depender somente de pacotes que são mais estáveis que ele [Martin 1997]. Isso significa que um pacote só pode depender de pacotes que sofrem um número de mudanças menor que o dele, pois as mudanças em um pacote podem causar efeitos colaterais nos pacotes que dependem dele.

De acordo com o princípio das abstrações estáveis, pacotes que possuem máxima estabilidade devem possuir máxima abstração. Pacotes instáveis devem ser concretos. A abstração de um pacote deve ser proporcional a sua estabilidade [Martin 1997]. Dessa forma, os pacotes estáveis podem ser facilmente estendidos por meio de sua abstração, e os pacotes mais concretos podem ser modificados com facilidade por serem instáveis.

Dentre as métricas definidas por Martin [1997] foi selecionado um conjunto composto por métricas de estabilidade. Essas métricas se baseiam na contagem de dependência entre pacotes, e têm como finalidade auxiliar na verificação do princípio das dependências estáveis em um projeto de software. As métricas desse conjunto são Afferent Coupling (Ca), Efferent Coupling (Ce) e Instability (I).

A métrica Ca conta a quantidade de classes externas que dependem de classes internas a um pacote. A métrica Ce conta a quantidade de classes internas que dependem de classes externas a um pacote. Por fim, a métrica I se utiliza dos resultados das duas métricas anteriores para indicar o nível de instabilidade de um pacote, por

meio da seguinte fórmula: `I = Ce/(Ca + Ce)`.O valor da métrica I está no intervalo [0,1], em que I=0 indica que o pacote possui estabilidade máxima e I=1 indica que o pacote possui máxima instabilidade.

2.1.2. Métrica de Coesão de Martin [2003]

Considerando que coesão é definida como sendo o relacionamento funcional dos elementos de um módulo, Martin [2003] propôs um conjunto de princípios de coesão que têm como finalidade auxiliar os projetistas a decidir como particionar classes dentro de pacotes. Dentre as métricas propostas por ele está a métrica de coesão relacional (H).

A métrica H conta o número médio de relacionamentos entre classes e interfaces dentro de um pacote. Considerando N como sendo o número de classes e interfaces dentro do pacote, e R como sendo o número de relacionamentos internos ao pacote, temos que: `H=(R + 1)/N,` onde o valor um é somado a R para evitar que H seja 0 quando N é 1. A métrica H auxilia os projetistas a identificar pacotes que possuem elementos pouco relacionados, o que pode significar que os pacotes são pouco coesos.

2.1.3. Métricas de Herança de Briand [1994]

Briand [1994] definiu e validou um conjunto de métricas de alto nível baseadas em acoplamento e coesão de módulos. Essas métricas têm como objetivo a avaliação de projetos de alto nível a fim de identificar defeitos e falhas de projeto. Dentre as métricas definidas por Briand [1994] está o conjunto de métricas baseadas em relacionamentos *is_component_of* [Ghezzi et al. 1991]. De acordo com Briand [1994] os módulos que possuem relacionamentos do tipo *is_component_of* formam estruturas chamadas de bibliotecas hierárquicas de módulos. Esse tipo de estrutura é como uma árvore, na qual os nós são módulos e as setas ligando os nós são relacionamentos *is_component_of*.

Dentro desse conjunto foram selecionadas duas métricas para o conjunto de métricas arquiteturais convencionais: profundidade máxima (Max_Depth) e profundidade média (Avg_Depth). A métrica Max_Depth conta o nível do nó mais profundo dentro da hierarquia e a métrica Avg_Depth conta o nível médio de profundidade dentro da hierarquia.

2.2. Métricas Sensíveis a Interesses

Um interesse pode ser definido como sendo uma propriedade ou parte de um problema que os *stakeholders* consideram como uma unidade conceitual e tratam isso de maneira modular. Os interesses podem variar de conceitos de alto nível, como segurança e qualidade de serviço a conceitos de baixo nível, como *cache* e *buffer*. Eles podem ser funcionais, como regras de negócio e características, ou não-funcionais, como sincronização e gerenciamento de transação.

As métricas propostas por Sant'Anna [2008] têm como principal objetivo ajudar os engenheiros de software a identificar falhas em projetos arquitetônicos causadas pela falta de modularidade nos interesses relevantes ao projeto, e consequentemente comparar as possíveis soluções para resolver os problemas relacionados aos interesses do projeto.

O conjunto das métricas sensíveis a interesses propostas por Sant'Anna [2008] que se utilizou neste trabalho é constituído das seguintes métricas: Concern Diffusion over Architectural Components (CDAC), Concern Diffusion over Architectural Interfaces (CDAI),

Concern Diffusion over Architectural Operations (CDAO), Component-level Interlacing Between Concerns (CIBC), Interface-level Interlacing Between Concerns (IIBC), Operation-level Overlapping Between Concerns (OOBC) e Lack of Concern-based Cohesion (LCC).

As métricas CDAC, CDAI, e CDAO estão relacionadas com o conceito de difusão de interesses, isto é, dado um determinado interesse elas contam a quantidade de componentes, interfaces e operações, respectivamente, que contribuem para a realização desse interesse. Já as métricas CIBC, IIBC e OOBC realizam contagens relacionadas ao entrelaçamento de interesses, assim, dado um interesse elas contam a quantidade de outros interesses que se relacionam com ele nos níveis de componente, interface e operação, respectivamente. A métrica LCC conta a quantidade de interesses tratados por um dado componente, a fim de analisar a coesão do componente com base nos interesses. Um módulo que se relaciona com muitos interesses é considerado pouco coeso, pois quanto maior for a quantidade de interesses maior será a probabilidade de o módulo sofrer efeitos colaterais oriundos de mudanças nos interesses.

3. Funcionamento da SDMetrics

SDMetrics é uma ferramenta de software cujo principal objetivo é medir a qualidade de projetos de software desenvolvidos com a UML [Wüst 2010]. Ela trabalha com qualquer ferramenta de modelagem UML que exporte seus diagramas para o formato XMI [OMG 2010].

O formato XMI (*XML Metadata Interchange*) é um padrão baseado em XML, desenvolvido pela OMG para troca de informações de metadados. É comumente utilizado como formato de intercâmbio de modelos UML entre ferramentas de modelagem.

A SDMetrics utiliza quatro arquivos para realizar suas medições:

- *XMI Source File*: Arquivo no formato XMI que descreve o diagrama que se pretende medir, no contexto deste trabalho, um diagrama que representa a arquitetura projetada;
- *Metamodel Definition*: Arquivo no formato XML que descreve o metamodelo da UML utilizado pelo diagrama arquitetural do arquivo *XMI Source File*;
- *XMI Transformation*: Arquivo no formato XML que possui as especificações necessárias para realizar o mapeamento do arquivo *XMI Source File* com o arquivo *Metamodel Definition*;
- *Metrics Definition*: Arquivo no formato XML contendo a definição das métricas que se pretende mensurar por meio da SDMetrics.

Juntamente com a versão 2.02 da SDMetrics foi disponibilizado um conjunto de métricas previamente automatizadas. Dentre elas estão as métricas apresentadas nas Subseções 2.1.1 e 2.1.2, além de uma série de métricas relacionadas a classes.

Os resultados das métricas são exibidos por meio de tabelas e gráficos. As métricas podem ser definidas para diferentes elementos arquiteturais, então os resultados das métricas são agrupados de acordo com o elemento arquitetural medido.

4. Automatização e validação das métricas na SDMetrics

O processo de automatização das métricas na SDMetrics passou por três estágios: (i) criação das definições das métricas não disponíveis na versão 2.0.2 da SDMetrics, (ii) adaptação de algumas métricas já existentes para o nível arquitetural desejado e (iii) validação da adaptação e da automatização das métricas envolvidas neste trabalho. Esses estágios são descritos nas Subseções 4.1, 4.2 e 4.3, respectivamente.

4.1. Automatização das métricas na SDMetrics

Como foi citado na Seção 3 a automatização das métricas definidas por [Martin 1997] [Martin 2003] foi obtida por meio de um conjunto de métricas previamente automatizadas por Wüst.

A SDMetrics permite que as métricas sejam automatizadas por meio de sua definição em um arquivo no formato XML que contém as especificações necessárias para que a SDMetrics interprete as métricas e realize a contagem na arquitetura fornecida por meio do arquivo XMI.

A definição do arquivo de métricas é feita usando *tags* e atributos pré-definidos. A seguir exibe-se a estrutura básica de um arquivo de métricas:

```
 1  <?xml version="1.0"?>
 2  <!DOCTYPE sdmetrics SYSTEM 'metrics.dtd'>
 3  <sdmetrics version="2.0" ruleexemption="taggedvalue" exemptiontag="tagname">
 4  <metric name="IsBusiness" domain="operation" category="aspect">
 5  <description>
 6  Indicates if the operation is assigned to Business.
 7  </description>
 8  <projection relset="stereotypes" target="stereotype" condition="name='Business'"/>
 9  </metric>
10  </sdmetrics>
```

O elemento da primeira linha define a versão da XML utilizada, na linha 2 é definido o arquivo DTD que contém as regras e restrições para as *tags* e atributos utilizados no arquivo. As palavras `xml`,`sdmetrics`,`metric`,`description` e `projection` são *tags* que representam diversos elementos utilizados pela SDMetrics para realizar a contagem. Palavras como *version* e *name* representam atributos para as *tags*. Cada atributo possui um valor atribuído, que será utilizado pela SDMetrics para realizar as contagens. A *tag* `metric` (linha 4) representa uma métrica, podendo existir diversas *tags* desse tipo dentro de um arquivo de definição de métricas.

No exemplo anterior, a métrica recebe o nome IsBusiness por meio do atributo "name", o domínio da métrica é "operation" (linha 4), isto significa que ela será contada para operações. As métricas de um domínio devem possuir nomes únicos, isto é, não é possível definir mais de uma métrica com um determinado nome para o mesmo domínio, mas podem-se definir métricas com o mesmo nome para domínios diferentes. O atributo "category" indica a categoria da métrica, esse atributo não é necessário, mas pode ser definido para melhor entendimento das métricas. A *tag* "description" é utilizada para fazer a definição textual da métrica. A *tag* "projection" é o elemento que define as condições e restrições para a contagem da métrica (linha 8). O atributo "relset" define o conjunto de elementos relacionados, "target" define o tipo dos elementos relacionados e "condition" filtra os elementos para que sejam contados apenas os elementos que satisfazem a condição definida. No exemplo, a métrica IsBusiness conta quantos estereótipos da operação possuem o nome igual a "Business".

As MSI foram definidas para o nível arquitetural de componentes. A forma encontrada para identificar interesses nos elementos arquiteturais foi por meio de estereótipos, pois a UML em sua versão atual não oferece suporte para modelagem de interesses em seus diagramas. Dessa forma, no exemplo anterior a métrica "IsBusiness" tem valor igual a zero para operações que não possuem o estereótipo "Business" indicando que esse interesse não está associado à operação e valor maior que zero para as operações que possuem esse estereótipo.

Como a SDMetrics não oferece suporte para definição de variáveis ou para que as métricas sejam contadas com base em entradas do usuário, foi necessário replicar todas as métricas para cada um dos interesses que pudessem estar presentes na arquitetura projetada. Assim, para que seja acrescentado um novo interesse a ser medido, basta replicar as definições existentes para cada uma das MSI. Além disso, as métricas de difusão de interesses (CDAC, CDAI, e CDAO) e de coesão baseada em interesses (LCC) foram agrupadas nos mesmos arquivos, pois as métricas auxiliares utilizadas pelos dois grupos são compatíveis. Por outro lado, as métricas de entrelaçamento de interesses (CIBC, IIBC e OOBC) foram definidas em outros arquivos por possuírem características diferentes.

As métricas de Briand [1994] foram automatizadas na SDMetrics usando a adaptação citada a seguir na Subseção 4.2 e conforme a estrutura básica definida pela SDMetrics.

4.2. Adaptação de métricas para o nível de projeto arquitetural

Sant'Anna [2008] definiu suas métricas baseado em uma arquitetura de componentes. Sendo assim, para possibilitar a realização de estudos comparativos entre as MAC e as MSI, as métricas de coesão e acoplamento [Martin 1997] [Martin 2003] que foram definidas baseando-se no conceito de pacotes, foram adaptadas para o nível de componentes. Pode-se ilustrar esta adaptação por meio do seguinte exemplo referente à métrica H:

```
1  <metric name="H" domain="package" category="Cohesion">
2  <description></description>
3  <compoundmetric term="(R+1)/(NumCls+NumInterf)" fallback="0" />
4  </metric>
```

Para realizar a adaptação dessa métrica para o nível de componentes foi necessário alterar o atributo "domain" de "package" para "component", mantendo a mesma estrutura da métrica para pacotes. Assim, a linha 1 foi substituída pela seguinte linha:

```
1  <metric name="H" domain="component" category="Cohesion">
```

Briand [1994] definiu e validou suas métricas por meio de modelos baseados na linguagem de programação ADA. Esses modelos não foram definidos por meio da UML. Então, para possibilitar a utilização das métricas de Briand [1994] neste trabalho, adaptou-se o conceito de bibliotecas hierárquicas de módulos para a abordagem orientada a objetos, considerando os módulos como sendo classes e os relacionamentos *is_component_of* como sendo relacionamentos de herança. Dessa forma, neste trabalho considera-se que as métricas Max_Depth e Avg_Depth são computadas para pacotes ou componentes dependendo do nível arquitetural utilizado. Como a hierarquia pode estar

completa dentro de um pacote ou espalhada em vários pacotes, as métricas consideram o grau de herança das classes presentes no pacote, mesmo que essa herança venha de classes de outros pacotes.

4.3. Validação das métricas automatizadas

Com o intuito de validar as métricas de acoplamento (Ca, Ce e I) [Martin 1997] e coesão (H) [Martin 2003], primeiramente realizou-se a contagem manual de cada métrica para o modelo da Figura 1. A partir do modelo da Figura 1 no formato XMI e do arquivo de definição destas métricas foi realizada a contagem por meio da ferramenta SDMetrics. Na sequência, os resultados obtidos por contagem manual foram comparados com os obtidos por meio da SDMetrics. Além disso, para as métricas Ca, Ce e I esses dois resultados foram comparados com os resultados apresentados por Martin [1997]. Dessa forma, verificou-se que os resultados das métricas convencionais de acoplamento e coesão automatizados pela SDMetrics estão de acordo com a sua definição [Martin 1997] [Martin 2003] e com a contagem manual. O Quadro 1 apresenta os resultados obtidos para o modelo da Figura 1.

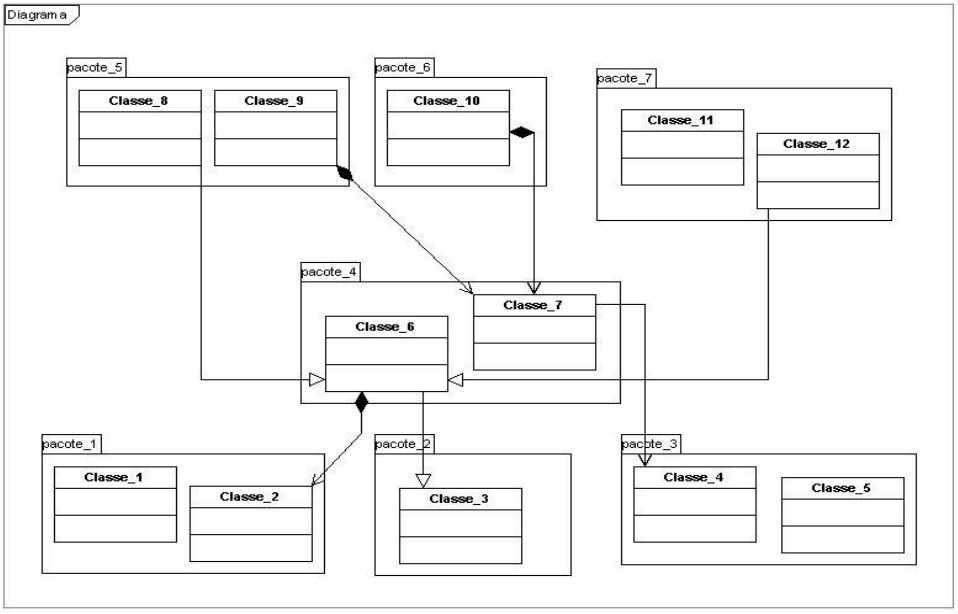

Figura 1. Modelo utilizado para validar as métricas convencionais [Martin 1997]

Para verificar a validade da automatização das métricas de herança (Max_Depth e Avg_Depth) [Briand 1994] utilizou-se também o modelo da Figura 1. O processo de validação foi o mesmo utilizado para as métricas de Martin, exceto pelo fato de não ser possível comparar os resultados obtidos com os resultados apresentados pelo autor, porque o modelo utilizado no trabalho de Briand [1994] não segue o padrão UML. Os resultados obtidos por meio da SDMetrics para as métricas de herança são apresentados no Quadro 1.

O resultado da métrica Ca para o pacote_2 da Figura 1 é um, pois existe uma classe externa ao pacote que depende de uma classe interna a ele por meio de um relacionamento de composição. Para o pacote_6 a métrica Ce é um porque existe um relacionamento de composição em que uma classe do pacote_6 depende de uma classe externa a ele. Assim, considerando que a métrica Ca para o pacote_6 é zero, temos pela

definição que o resultado da métrica I para o pacote_6 é um. Para o pacote_2 a métrica H é um porque existe uma classe interna ao pacote e nenhum relacionamento interno. As métricas Max_Depth e Avg_Depth para o pacote_5 são respectivamente dois e um, pelo fato de existirem duas classes dentro do pacote_5, sendo que uma delas possui grau dois e a outra zero dentro da hierarquia de herança.

Quadro 1. Resultados das métricas de coesão, acoplamento e herança para o modelo da Figura 1.

Pacote	Ca	Ce	I	H	Max_Depth	Avg_Depth
pacote_1	1	0	0	0.5	0	0
pacote_2	1	0	0	1	0	0
pacote_3	1	0	0	0.5	0	0
pacote_4	4	3	0.42857143	0.5	1	0.5
pacote_5	0	2	1	0.5	2	1
pacote_6	0	1	1	1	0	0
pacote_7	0	1	1	0.5	2	1

No processo de validação dos resultados das MSI utilizou-se uma arquitetura baseada em um sistema de informação real chamado Health Watcher [Soares et al. 2002] utilizada por Sant'Anna [2008]. Essa arquitetura é apresentada na Figura 2.

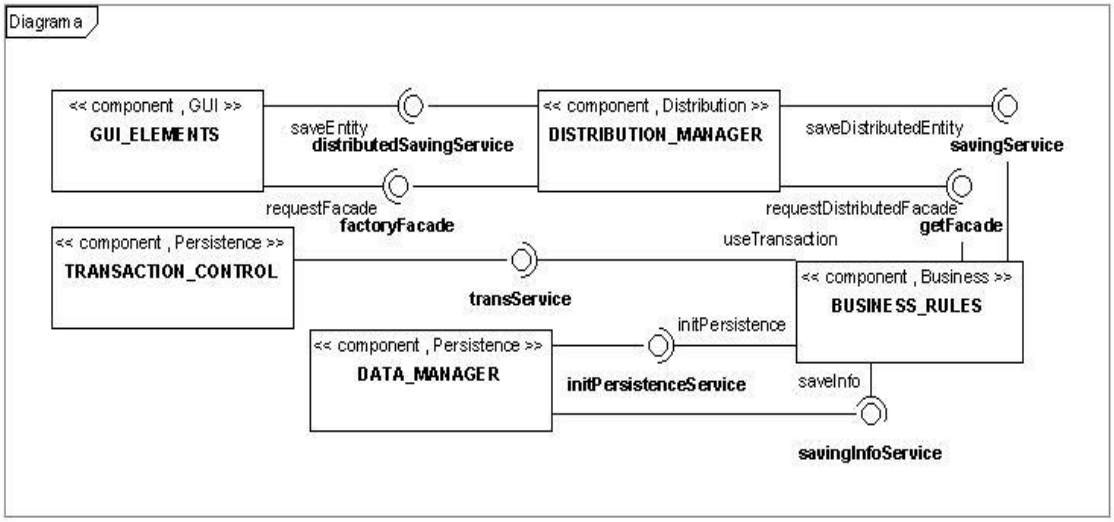

Figura 2. Arquitetura do sistema Health Watcher [Sant'Anna 2008].

Assim como nas métricas convencionais, primeiramente realizou-se a contagem manual das métricas para o modelo da arquitetura de Health Watcher. Posteriormente, obteve-se o modelo dessa arquitetura no formato XMI, e com base nesse arquivo XMI nos arquivos de definição das MSI realizou-se a contagem por meio da SDMetrics. Os resultados obtidos por contagem manual e pela SDMetrics foram iguais, porém houve divergência com relação aos resultados apresentados por Sant'Anna [2008] em seus exemplos para a métrica CIBC. Mas, o resultado obtido pela SDMetrics para esta métrica está de acordo com a definição formal apresentada por Sant'Anna [2008]. Dessa forma, averiguou-se que se trata de um erro no exemplo de Sant'Anna [2008], e que os

resultados apresentados pela SDMetrics para as MSI estão corretos. Os resultados das MSI para a arquitetura Health Watcher são exibidos no Quadro 2 e na Figura 3.

Quadro 2. Resultados das métricas de difusão de interesses e de entrelaçamento de interesses para a arquitetura Health Watcher.

Interesse	Difusão de Interesses			Entrelaçamento de Interesses		
	CDAO	CDAI	CDAC	OOBC	IIBC	CIBC
Business	3	5	1	0	2	2
Distribution	4	4	1	1	2	2
Exception Handling	5	2	2	2	4	4
GUI	0	2	1	0	0	0
Persistence	4	6	4	1	4	4

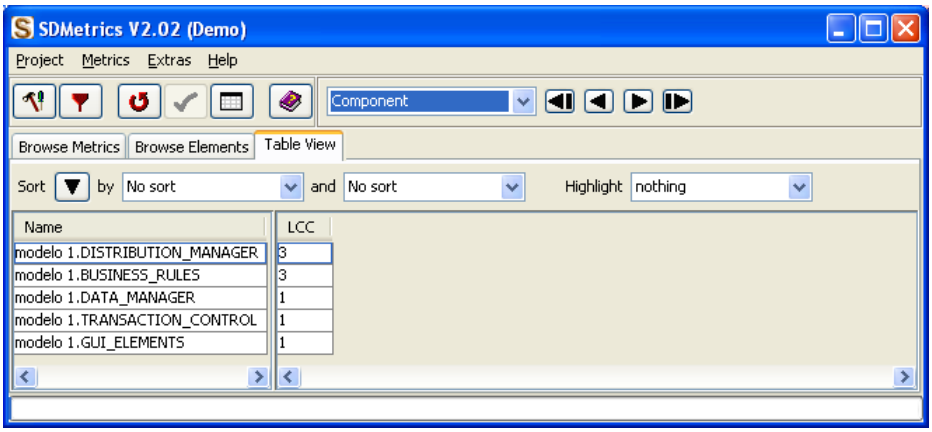

Figura 3. Resultado da métrica de coesão baseada em interesses na SDMetrics

Na arquitetura da Figura 2 para as métricas CDAO, CDAI e CDAC existem, respectivamente, quatro operações, quatro interfaces e um componente associados ao interesse *Distribution*. Para OOBC, IIBC e CIBC, *Distribution* está entrelaçado com um interesse em nível de operação, dois interesses em nível de interface e dois interesses em nível de componente. Para o componente GUI_ELEMENTS, LCC tem valor um porque ele está associado ao interesse *Distribution*.

5. Resultados e Discussões

Algumas limitações foram encontradas na SDMetrics com relação a definição de métricas genéricas. A ferramenta não possibilita a definição de métricas que recebam parâmetros fornecidos pelo usuário, dessa forma as métricas sensíveis a interesses só puderam ser definidas para interesses fixos. Sendo assim, como mencionado anteriormente foi necessário replicar diversas métricas para cada um dos interesses utilizados no trabalho, isso gerou uma quantidade de código muito grande. Então as métricas foram divididas em vários arquivos para facilitar o manuseio e entendimento.

As MSI foram automatizadas para os interesses: *Business, Concurrency, Distribution, Exception Handling, GUI, Logging e Persistence*. Para realizar a medição de interesses diferentes, é necessário que as automatizações de das MSI sejam replicadas para cada interesse novo.

Apesar dessa limitação, a SDMetrics tem facilidade para operar com modelos UML e é eficiente para medir arquiteturas, o que atende o objetivo maior do projeto que

é automatizar a medição para realizar estudos empíricos. Existem outros protótipos de ferramentas de medição, como a COMET [Sant'Anna 2008], mas nenhum deles é maduro como a SDMetrics.

Ao final do projeto obtiveram-se nove arquivos XML com as definições das métricas convencionais e sensíveis a interesses conforme relação a seguir:

- metricasOO.xml: Contém as métricas convencionais de acoplamento definidas por Martin [1997] e de coesão definida por Martin [2003] para o nível de pacotes além da adaptação feita para o nível de componentes. Além disso, contém as métricas Max_Depth e Avg_Depth tanto para o nível de pacotes como para o nível de componentes [Briand 1994].
- Concern_Based_Difusion_and_Cohesion.xml: Contém as métricas de difusão de interesses e de coesão baseada em interesses definidas por Sant'Anna [2008].
- Interaction_Between_Concerns_[`nome do interesse`].xml: devido a uma limitação da SDMetrics, a definição das métricas de entrelaçamento de interesses ficaram atreladas a interesses específicos. Dessa forma, para cada interesse houve a replicação das definições em arquivos distintos, constituindo-se sete arquivos que levam em seu nome o interesse para o qual as métricas foram especificadas.

6. Conclusão

A aplicação de métricas arquiteturais convencionais e sensíveis a interesses em arquiteturas de LPS possibilita a identificação de anomalias presentes no projeto da arquitetura da LPS. Com o objetivo de automatizar a medição de métricas arquiteturais sensíveis a interesses e convencionais, realizou-se a automatização desses conjuntos de métricas na ferramenta SDMetrics.

Após a especificação dos arquivos com métricas para a SDMetrics, foi realizada a validação da especificação usando os exemplos adotados por Martin [1997] e por Sant'Anna [2008].

A automatização de métricas com a SDMetrics permite a aplicação de métricas em diversos modelos, evitando que a contagem seja realizada manualmente, assim evita-se que ocorram erros de contagem em conseqüência de falhas humanas. Além disso, um modelo com uma alta complexidade pode ser mensurado de forma rápida e eficiente por meio da SDMetrics.

Finalmente, conforme foi relatado, a automatização de métricas arquiteturais facilita o processo de medição tanto de modelos arquiteturais simples como de modelos arquiteturais complexos, evitando que ocorram falhas de contagem. Além disso, por meio dos resultados apresentados pelas métricas, é possível avaliar possibilidades de solução arquitetural para um determinado projeto de software.

Futuramente pretende-se realizar estudos empíricos, com base nos resultados deste trabalho, para analisar a eficácia dos dois conjuntos de métricas automatizados com relação à detecção de anomalias de modularidade em arquiteturas de LPS. Além disso, planeja-se a realização de estudos de caso que permitam uma análise comparativa e a obtenção de indicadores dos ganhos obtidos com o uso da ferramenta configurada com as MAC e MSI.

Referências

BASS, L.; CLEMENTS, P.; KAZMAN, R. (2003) Software Architecture in Practice, Addison-Wesley, 2nd edition.

BERTRÁN, I. M.; GARCIA, A.; von STAA, A. (2009) "Estratégias de Detecção de Anomalias de Modularidade em Sistemas Orientados a Aspectos", Anais do III Latin American Workshop on Aspect-Oriented Software Development, XXIII Simpósio Brasileiro de Engenharia de Software. Fortaleza/CE.

BRIAND, L.; MORASCA, S.; BASILI, V. (1994) "Defining and validating high-level design metrics", CS-TR 3301, Univ. of Maryland, College Park.

BOSCH, J. (2000) Design and Use of Software Architectures, Addison-Wesley.

GHEZZI, C.; JAZAYERI, M.; MANDRIOLI, D. (1991) Fundamentals of Software Engineering, Englewood Cliffs, NJ., USA: Prentice Hall.

KICZALES, G. et al. (1997) "Aspect-Oriented Programming", EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING (ECOOP), LNCS (1241), Springer-Verlag, Finland.

MARTIN, R. (2003) Agile Software Development: Principles, Patterns, and Practices, Prentice Hall.

MARTIN, R. (1997) "Stability", C++ Report.

OMG (2010) XMI, http://www.omg.org/spec/XMI/2.1.1/, May.

POHL, K., BOCKLE, G., van der LINDEN, F. (2005) "Software Product Line Engineering: Foundations, Principles and Techniques", Springer.

SANT'ANNA, C. N. (2008) "On the Modularity of Aspect-Oriented Design: A Concern-Driven Measurement Approach", Doctoral Thesis. PUC-Rio, Rio de Janeiro.

SANT'ANNA, C.; FIGUEIREDO, E.; GARCIA, A.; LUCENA, C. (2007) "On the Modularity of Software Architectures: A Concern-Driven Measurement Framework", In: Proceedings of the 1st European Conference on Software Architecture, September, Madrid, Spain.

SANT'ANNA, C.; GARCIA, A.; CHAVEZ, C.; LUCENA, C.; von STAA, A. (2003) "On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework", XVII SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE. Anais, p. 19-34.

SOARES, S.; LAUREANO, E.; BORBA, P. (2002) Implementing Distribution and Persistence Aspects with AspectJ, Proceedings of the 17th ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA' 02, ACM Press, p. 174–190, November.

van der LINDEN, F. SCHMID, K. ROMMES, E. (2007) "Software Product Lines in Action – The Best Industrial Practice in Product Line Engineering", Springer.

WÜST, J. (2010) SDMetrics, http://www.sdmetrics.com/, May.