CodeMI – Source Code as XML Metadata Interchange
Uma Representação de Código-fonte para Coleta de Métricas

João Paulo O. dos Santos, Márcio de O. Barros

Centro de Ciências Exatas e Tecnologia
Universidade Federal do Estado do Rio de Janeiro (UNIRIO)
Rio de Janeiro, RJ - Brasil
{joao.santos, marcio.barros}@uniriotec.br

Abstract. *XML based software representations allow easier processing, analysis, and manipulation of source code for automated tools. These representations provide significant details about the structure of the source code, but the verbosity of some representations may bring difficulties to code sharing and to the execution of software evolution studies in the industry. In this paper, we present CodeMI, a XML-based source code representation that uses XMI extension to define elements with emphasis on the structure of source code, enabling the collection of structural metrics without showing the dynamics of the software execution.*

Resumo. *Representações de software baseadas em XML possuem nível de abstração adequado para o processamento, análise e manipulação de código-fonte por ferramentas. Estas representações fornecem detalhes expressivos sobre a estrutura do código-fonte. A alta verbosidade de algumas representações e a exposição de detalhes do código-fonte em excesso dificultam a realização de estudos de evolução de softwares industriais. Neste artigo, apresentamos CodeMI, uma representação de código-fonte baseada em XML que utiliza a extensão do formato XMI para definir elementos, com ênfase na estrutura do código-fonte, que viabilizem a coleta de métricas sem revelar a dinâmica de execução do software.*

1. Introdução

A representação de código-fonte em formato texto é largamente utilizada por programadores para a codificação de algoritmos desde as primeiras linguagens de programação de computador até os dias de hoje. É também considerada como o formato universal para a construção de código-fonte, dadas as facilidades de manipulação e intercâmbio entre ferramentas, como editores de texto, sistemas de controle de versão, entre outras [Badros 2000]. No entanto, o formato texto não fornece a mesma versatilidade quando se deseja coletar métricas a partir do código-fonte.

A principal limitação da representação textual é a dificuldade de se obter, a partir de um processo automático, a estrutura do software, pois isto exige a construção de um analisador sintático (*parser*) especifico para cada linguagem de programação. Por estrutura, entendemos os pacotes e classes que formam o programa, os atributos e métodos destas classes e os comandos de controle de fluxo que fazem parte da implementação destes métodos. Em geral, a análise sintática do código-fonte é

desempenhada pelos compiladores e por uma pequena parte das ferramentas de engenharia de software. Ferramentas como LEX/YACC [Levine 1992] suportam o desenvolvimento dos analisadores sintáticos, porém a dependência deles implica em que cada ferramenta que envolva extração de medidas de código-fonte implemente um *parser* próprio. Esta implementação, além de dispendiosa, envolve o conhecimento de teoria de compiladores e não é tarefa simples.

O recente crescimento do interesse por pesquisas em evolução de software, que tangem ao modo como o desenvolvimento e a manutenção de sistemas ocorrem ao longo do tempo, demanda a utilização de grandes repositórios de código-fonte para a coleta de informações sobre como a forma como o software foi desenvolvido. A partir destes repositórios são extraídas as métricas desejadas do software e são formadas séries temporais para serem analisadas.

Com o objetivo de facilitar a análise da estrutura do código-fonte de sistemas por ferramentas automatizadas, surgiram as primeiras representações de código no formato XML (*eXtensible Markup Language*) [XML 2006]. Este formato, embora possa ser exposto em uma representação textual, é capaz de apresentar elementos e relações entre estes elementos de forma estruturada. Assim, uma representação de código-fonte em formato XML evidencia os elementos estruturais de um código fonte e viabiliza a extração de medidas estruturais do software. Dada a gama de ferramentas existentes para a realização de consultas, manipulação e transformação sobre informações em formato XML, a realização de análises sobre estas medidas se torna uma tarefa mais simples do que o processamento do código-fonte original.

Sendo assim, a representação XML do código fonte de um sistema traz uma série de benefícios: (a) estrutura o código de forma explícita, o que facilita sua manipulação; (b) possibilita criar consultas mais poderosas sobre os componentes do código-fonte, pois não será mais feita uma pesquisa textual através de expressões regulares, mas uma busca em marcadores onde se pode utilizar linguagens e ferramentas apropriadas; (c) possui representação extensível, o que facilita criar extensões no código, características que o formato textual não permitia; (d) possibilita fazer referência cruzada entre elementos de código, o que não é possível em representações textuais, pois os elementos são referenciados através de linhas e colunas; e (e) suporte amplo, uma vez que XML é suportado por várias plataformas [Mendonça 2004].

A utilização de sistemas de software livre é bastante comum na realização de estudos de evolução de software, dada a facilidade de obter o código-fonte a partir de repositórios públicos. No entanto, muitas características referentes ao desenvolvimento de software livre diferem das formas adotadas no desenvolvimento e manutenção de sistemas industriais. Por outro lado, os sistemas de informação utilizados nas empresas são, em sua maioria, protegidos por direitos autorais e/ou de propriedade intelectual. Isto inviabiliza a generalização das conclusões dos estudos realizados com base em software livre e dificulta a realização de qualquer tipo de estudo que tenha a necessidade de acesso direto ao código-fonte desenvolvido pelas empresas. Surge então a necessidade de uma representação de código-fonte que preserve as características necessárias para a realização de estudos de evolução de software, ao mesmo tempo em que esconda os detalhes do código-fonte, de modo a não infringir a propriedade intelectual das empresas que o desenvolveram.

Embora existam diversas representações de código-fonte em formatos mais estruturados do que o formato textual das linguagens de programação (entre elas, algumas que utilizam o próprio formato XML), nenhuma das representações analisadas em um estudo comparativo se mostrou capaz de atender aos critérios estabelecidos para a extração de métricas estruturais do código-fonte de sistemas industriais.

Visando criar uma representação de código-fonte, independente de linguagem de programação, que permita a extração de métricas estruturais e que não evidencie os segredos industriais refletidos na lógica do código, propomos a CodeMI (*Source Code as XML Metadata Interchange). CodeMI é* uma representação de código-fonte em formato XML que utiliza o formato XMI v2.1[XMI 2007] para descrever elementos da estrutura de classes de um projeto de software orientado a objetos, como os pacotes, as classes, os atributos e os métodos do sistema. XMI é um formato padronizado pela OMG *(Object Management Group)* para a troca de informações sobre modelos UML [UML 2007]. Este formato é amplamente utilizado por ferramentas de modelagem baseadas em repositórios MOF *(Meta-Object Facility)* [MOF 2002].

Para descrever a estrutura da implementação dos métodos foi utilizado o mecanismo de extensão do formato XMI. Por extensão, entende-se a introdução de novos elementos ao formato XMI padronizado, sem gerar incompatibilidade para as ferramentas que já utilizam o formato original. Assim, as ferramentas que conhecem as extensões poderão fazer uso destas informações, enquanto as demais ferramentas as ignorarão. Este mecanismo de extensão foi utilizado para representar estruturas de repetição, desvios condicionais e incondicionais, comandos e outros aspectos da estrutura do código-fonte que não são relevantes para os fins mais usuais do formato XMI (como a criação e compartilhamento de diagramas UML, por exemplo), mas devem ser considerados para a extração de métricas estruturais. Sendo o formato XMI um tipo específico de XML, será possível utilizar as mesmas ferramentas de análise e manipulação existentes sobre os dados obtidos na representação CodeMI para a análise e coleta de métricas. Com a vantagem de poder analisar estruturas do código-fonte.

Na seção 2 serão apresentados alguns trabalhos relacionados à representação de código-fonte em XML e um breve estudo comparativo entre estas representações. Na seção 3 será apresentado o formato XMI e suas principais utilizações. Na seção 4 apresentaremos o formato CodeMI, descrevendo seus elementos e a ferramenta utilizada para converter o código-fonte nesta representação. Na seção 5 abordaremos sobre os métodos e ferramentas que podem ser utilizadas para a coleta de métricas estruturais a partir da CodeMI e na seção 6 apresentaremos algumas conclusões e sugestões de trabalhos futuros.

2. Trabalhos Relacionados

Os formatos das representações de código-fonte em XML existentes são classificados em representação específica de uma linguagem de programação e representação genérica. As representações específicas utilizam marcadores para especificar estruturas e comandos restritos à linguagem de programação para a qual foram planejadas. Já as representações genéricas, não se destinam a representar apenas uma linguagem de programação, mas as características comuns a um conjunto de linguagens, como as linguagens de programação orientadas a objetos.

Entre as representações específicas da linguagem Java destacam-se a JavaML de Mamas e Kontogiannis [Mamas and Kontogiannis 2000], a JavaML de Badros [Badros 2000], XJava [XJava 2008] e JavaML 2.0 [Aguiar 2004]. As representações específicas da linguagem Java distinguem-se fundamentalmente nos seguintes critérios: (a) representação dos elementos essenciais da linguagem; (b) armazenamento de informações estruturais; (c) verbosidade; (d) preservação das linhas em branco, (e) preservação dos comentários; (f) preservação da árvore sintática abstrata (*Abstract Syntax Tree - AST*) do código-fonte; (g) exposição do código-fonte e; (h) granularidade do arquivo. Estes critérios serão adotados no estudo comparativo realizado a seguir.

Na JavaML de Mamas e Kontogiannis os elementos do código-fonte são especificados nos marcadores (*tags*) da representação XML e os valores destes elementos são representados nos atributos destes marcadores (*tag attributes*). O corpo do marcador (*tag body*) não é utilizado para armazenar informações do código-fonte. Esta representação é capaz de descrever os elementos essenciais da linguagem de programação, como as classes, seus métodos e atributos, porém não armazena informações estruturais da implementação dos métodos, como número de linha e coluna dos comandos, início e fim de bloco, formatação e comentários. Outra desvantagem desta representação é a alta verbosidade, o que torna sua representação em XML bastante longa e pouco legível.

A representação JavaML de Badros *(*ou *JavaML 1.0)* [Badros 2000] assemelha-se bastante à JavaML de Mamas e Kontogiannis, pois ambas preocupam-se em manter informações da AST do código-fonte no formato XML. Ambas as representações também descrevem cada elemento do código através de marcadores, apresentando seus valores em atributos, sem utilizar o corpo dos marcadores. A representação JavaML, no entanto, fornece informações estruturais, como o número da linha inicial e final de cada elemento, além de ser mais sucinta (menos verborrágica que a representação de Mamas e Kontogiannis), pois utiliza uma quantidade menor de marcadores para representar os elementos do código-fonte, com nomes menores e intuitivos. Por outro lado, as desvantagens desta representação envolvem a não preservação de informações de formatação, linhas em branco e comentários.

A representação XJava [XJava 2008] é obtida através da transformação do código-fonte por um *parser* Java desenvolvido através da ferramenta BeaultyJ [XJava 2008]. Esta representação, diferentemente das anteriores, não representa a AST do código-fonte, não armazena informações estruturais, de formatação e espaços em branco. No entanto, ela também se baseia no formato XML e utiliza marcadores e atributos para representar pacotes, classes, métodos e atributos de forma simples e semelhante à representação JavaML de Badros. A principal vantagem da XJava é o fato desta apresentar os comentários feitos pelos programadores no código-fonte e também por apresentar baixa verbosidade, dada sua semelhança com JavaML de Badros.

Finalmente, a representação JavaML 2.0 [Aguiar 2004] apresenta diversas melhorias em relação à primeira versão apresentada por Badros. Entre estas, se destaca: (a) a preservação da formatação e o armazenamento dos comentários realizados pelos programadores; (b) uma maior riqueza de detalhes sobre informações estruturais, totalizando noventa marcadores distintos; e (c) informações semânticas relacionadas com a definição de símbolos, referências e tipos.

Tabela 1. Comparativo entre representações específicas para Java

	JavaML M&K	JavaML 1.0	XJava	JavaML 2.0
Elementos essenciais	Sim	Sim	Sim	Sim
Informações estruturais	Não	Sim	Não	Sim
Verbosidade	Alta	Média	Baixa	Alta
Linhas em branco	Não	Não	Não	Sim
Comentários	Não	Não	Sim	Sim
Representa AST	Sim	Sim	Não	Sim
Exposição do Código	Sim	Sim	Sim	Sim
Granularidade	Classe	Classe	Sistema	Classe

Após a análise comparativa das representações específicas da linguagem Java, destacamos a JavaML 2.0 por atender a totalidade dos critérios adotados, conforme mostrado na Tabela 1. No entanto, apesar da JavaML 2.0 possuir riqueza de detalhes sobre a estrutura da linguagem Java, sua expressiva verbosidade dificulta a extração de métricas, sobretudo às relacionadas ao código-fonte implementado internamente ao corpo do método.

A principal desvantagem destas representações específica da linguagem é a exposição do código-fonte, que foi observada nas quatro representações analisadas. Este critério, permite que o código-fonte possa ser extraído a partir do XML, que em diversas situações devem ser proibidas, pois caracterizam violação de restrições de propriedade de software ou intelectual, além de facilitar uma possível reengenharia do sistema. Por outro lado, as representações com baixa verbosidade, como XJava, possuem a desvantagem da falta de informações estruturais da implementação dos métodos. Critérios estes, que inviabilizam totalmente o propósito deste estudo.

Em relação à granularidade do arquivo foi observado que a maioria das representações analisadas gera um arquivo para cada classe. Isto dificulta a coleta de métricas baseadas em mais de uma classe, por exemplo, métricas de pacote. Por outro lado, o mesmo não ocorre na XJava, que fornece um único arquivo para todo o sistema. Caso o sistema seja extremamente grande, podem surgir problemas de processamento para as ferramentas de consulta e manipulação. Em contrapartida as consultas a este arquivo podem ser facilmente elaboradas.

Uma outra desvantagem é a diversidade de marcadores e atributos utilizados pela JavaML 2.0 para representar a AST do código-fonte, que torna a tarefa de extrair métricas estruturais bastante árdua, dada a dificuldade de identificar quais marcadores serão utilizados na extração de determinada medida. Por exemplo, quais marcadores devem ser considerados quando desejamos extrair uma simples métrica, como o número de comandos em um método? Ou quando calcular a complexidade ciclomática [McCabe 1976] de um determinado método?

Devido à falta de representações especificas da linguagem Java com baixa verbosidade e que forneçam informações estruturais suficientes para a extração de métricas baseadas na estrutura do código-fonte, optamos por buscar outros formatos de representação. Analisamos então as representações genéricas de código-fonte, dentre as

quais se destacam: OOML [Mamas and Kontogiannis 2000], GXL [Holt 2000] e srcML [Maletic 2002].

A representação OOML [Mamas and Kontogiannis 2000] também foi proposta por Mamas e Kontogiannis e, sendo uma representação genérica, apresenta características comuns utilizadas pelas linguagens Java e C++. Apesar de ser uma representação que serviria para muitas linguagens, sua desvantagem é a perda das características próprias de cada linguagem, já que nem todas possuem as mesmas estruturas.

A representação GXL [Holt 2000] é uma representação de estruturas de código em forma de grafos direcionados. As entidades são representadas como nós e as arestas simbolizam os relacionamentos entre elas. Foi originalmente projetada para ser um padrão de troca de artefatos de software em diferentes níveis de abstração. Por isso, não armazena informações relativas as estruturas do código-fonte.

A representação srcML [Maletic 2002] foi projetada para representar o código-fonte por completo sem perda de informação. Para isto adiciona marcadores ao código-fonte, preservando informações como espaços em branco, comentários e formatação do texto. Isto torna o código-fonte muito extenso e com abundancia de informações não essenciais para sua manipulação.

Não foram encontrados critérios comuns para realizar um estudo comparativo entre estas representações genéricas de código-fonte citadas acima, pois cada uma destas possui uma finalidade especifica. Com isto não foram encontradas representações com baixa verbosidade capazes de representar estruturas do código-fonte.

3. O Formato XMI

O formato XMI *(XML Metadata Interchange)* é, segundo a OMG, o formato para representação, troca e compartilhamento de modelos de sistemas orientados a objeto. Este formato representa uma tentativa de solucionar o problema de interoperabilidade entre ferramentas de modelagem, repositórios de meta-dados e outras ferramentas de desenvolvimento, utilizando para tal, o formato XML.

O formato XMI define uma notação padronizada para os esquemas conceituais do MOF [MOF 2002], que é um padrão utilizado para definir, manipular e integrar meta-modelos para o desenvolvimento de software orientado a objetos. Como é uma linguagem para definição de meta-modelos, o MOF é conhecido como meta-meta-modelo (ou modelo nível M3). No MOF são definidos os conceitos de pacotes, classes, métodos e atributos. O XMI especifica como devem ser gerados documentos XML para a representação de modelos compostos por elementos definidos pelo MOF. A figura 1 ilustra algumas possibilidades de utilização do XMI no desenvolvimento sistemas.

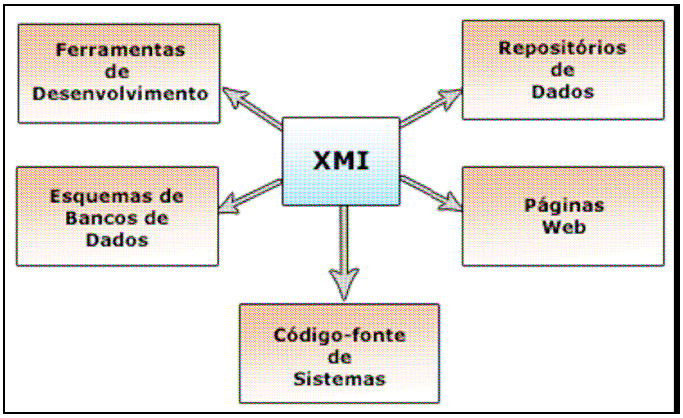

Figura 1 - Uso do XMI no desenvolvimento de sistemas

O principal objetivo da XMI é facilitar o intercâmbio de meta-dados, termo genérico para qualquer dado que, de alguma forma, descreve uma informação que pode ser compartilhada entre ferramentas de modelagem baseadas na UML *(Unified Modeling Language)* e repositórios de meta-dados baseados no MOF, ainda que em um ambiente heterogêneo. Através do XMI são integrados três padrões da indústria de software: XML, UML e MOF.

Existem diversas versões de XMI disponíveis: 1.0, 1.1, 1.2, 2.0 e 2.1. Entretanto, não há compatibilidade entre as versões 1.x e 2.x, devido a modificações significativas de sintaxe, como o desaparecimento dos elementos *header* e *content* existentes na versão 1.x [XMI 2007].

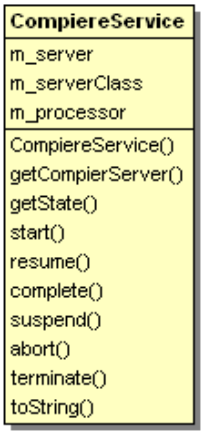

Figura 2 - Classe *CompiereService* em UML.

Como exemplo da utilização de XMI, considere a classe *CompiereService*, apresentada na Figura 2 em sua notação UML. Esta classe foi obtida a partir do sistema *Compiere*, que é um sistema de CRM e ERP distribuído como software livre. *Compiere* é considerado um sistema de gestão completo, pois integra as diversas áreas de negócio de uma empresa.

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.1" xmlns:uml="http://schema.omg.org/spec/UML/2.1" xm
<uml:Model xmi:type="uml:Model" xmi:id="modelo" name="XmiJavaParser">
     <packagedElement xmi:type="uml:Package" xmi:id="PARSER_120" name="org.c
         <packagedElement xmi:type="uml:Class" name="CompiereService" xmi:id
             <ownedAttribute xmi:type="uml:Property" name="m_server" xmi:id=
             <ownedAttribute xmi:type="uml:Property" name="m_serverClass" xm
             <ownedAttribute xmi:type="uml:Property" name="m_processor" xmi:
             <ownedOperation xmi:type="uml:Operation" name="CompiereService"
             <ownedParameter xmi:type="uml:Parameter" name="processor" xmi:i
             <ownedParameter xmi:type="uml:Parameter" name="serverClass" xmi
             </ownedOperation>
             <ownedOperation xmi:type="uml:Operation" name="getCompierServer
             </ownedOperation>
             ...
             <ownedOperation xmi:type="uml:Operation" name="toString" xmi:id
             </ownedOperation>
         </packagedElement>
     </packagedElement>
</uml:Model>
</xmi:XMI>
```

Figura 3.- Classe *CompiereService* em XMI.

Na figura 3, temos a representação da classe *CompiereService* no formato XMI v2.1. Nesta podem ser observados os elementos *packagedElement*, com atributo *xmi:type=”uml:Package”* para representar o pacote *org.compiere.process*, e com atributo *xmi:type=”uml:Class”*, para representar a classe *CompiereService*, *ownedAttribute* e ownedOperation utilizados para representar, respectivamente, atributos e métodos da classe.

4. A Representação CodeMI

Com objetivo de preencher a lacuna existente entre as representações específicas de código-fonte e as representações genéricas, no que tange ao armazenamento de informações estruturais em nível adequado, sem a expressiva quantidade de marcadores e atributos, apresentamos CodeMI (Source Code as XML Metadata Interchange). Esta representação complementa a estrutura de um sistema orientado a objeto (pacotes, classes, métodos e atributos) representada em formato XMI, adicionando marcadores para descrever estruturas relativas à implementação interna dos métodos.

A representação CodeMI foi assim designada por introduzir ao formato XMI elementos que representam a implementação dos métodos. Estes novos elementos visam representar a estrutura do código-fonte sem fornecer detalhes que revelem informações sobre sua dinâmica de execução. Além disso, os novos elementos adicionados fornecem à representação, a legibilidade e verbosidade adequada à extração de métricas. Na tabela 2 podemos visualizar os principais elementos adotados por esta representação.

O processo de conversão do código-fonte para a representação CodeMI é realizado por um *parser* desenvolvido em Java. Este *parser* utiliza a JRefactory [Seguin 2000] para percorrer a AST *(Abstract Syntax Tree)* do código dos métodos. Cada elemento estrutural identificado na AST é então convertido em um dos elemento da CodeMI descartando detalhes da implementação.

Tabela 2. Elementos da representação CodeMI

Marcadores	Atributos	Descrição
<if> </if>	conditions	Representa o desvio condicional *if*. O atributo *conditions* armazena o total de expressões condicionais que fazem parte da decisão do desvio condicional.
<else> </else>		Representa o *else* do *if*, aplicado no corpo do marcador *if*.
<switch> </switch>		Representa a estrutura de controle *switch*.
<case> </case>		Representa estrutura de controle *case*, dentro de um *switch*
<for> </for>		Representa a estrutura de repetição *for*.
<while> </while>		Representa a estrutura de repetição while.
<try> </try>		Representa a estrutura de tratamento de exceção *try/catch*.
<statement>		Representa um comando.
<localvar>	Type	Representa uma declaração de variável. O atributo *type* armazena o tipo da variável.
<jump>		Representa um desvio incondicional.
<break>		Representa o comando *break*.
<return>		Representa o retorno do método.

Para apresentar a conversão de um código-fonte Java para CodeMI utilizamos a implementação do método *terminate* da classe *CompiereService*, conforme exibido abaixo na figura 4.

```
public boolean terminate()
{
    if (super.terminate())
    {
        if (m_server != null && m_server.isAlive())
        {
            try
            {
                m_server.interrupt();
            }
            catch (Exception e)
            {
            }
        }
        log.info("terminate - done");
        return true;
    }
    return false;
}   //  terminate
```

Figura 4.- Exemplo método *terminate* em Java.

Após a execução do *parser* sobre a classe *CompiereService* obtemos sua representação em CodeMI, como pode ser visualizado na figura 5.

```
<ownedOperation xmi:type="uml:Operation" name="terminate"
<xmi:extension>
   <if conditions = "1" >
       <if conditions = "2" >
           <try>
               <statement/>
           </try>
       </if>
       <statement/>
       <return/>
   </if>
   <return/>
</xmi:extension>
</ownedOperation>
```

Figura 5.- Exemplo método *terminate* em CodeMI.

Na figura 5 podemos observar que os elementos que representam a estrutura do código-fonte possuem fácil leitura e baixa verbosidade devido a inexistência de excessivos detalhes e atributos. Outro destaque da *CodeMI* que pode ser observado é que a partir desta representação não é possível recuperar o código-fonte original, assegurando a restrições de software possivelmente existentes nos softwares industriais. Ao final deste processo de conversão para a CodeMI será gerado um arquivo *xmi* para cada pacote. Cada arquivo armazenará informações sobre as diversas classes do pacote analisado.

5. Extraindo Métricas

Para a extração de métricas estruturais sobre a representação CodeMI foram utilizadas transformações XSLT - Extensible Stylesheet Language Transformations [XSLT 2001]. Uma transformação XSLT é expressa como um documento XML que inclui elementos definidos pela linguagem de transformação e elementos externos definidos pelo usuário. Uma transformação expressa em XSLT descreve regras de transformação de uma árvore XML em outra através da associação de padrões a modelos. Um padrão é casado com elementos do documento, como nós ou atributos. Um processador XSLT percorre todos os nós do documento, compara-o a cada padrão do modelo, e caso eles sejam compatíveis, a regra é aplicada. Uma de suas principais aplicações é a transformação de dados XML em HTML.

Para exemplificar a extração de métricas sobre a representação CodeMI a partir de transformações XSLT selecionamos a métrica de complexidade ciclomática [McCabe 1976]. Esta é uma medida que visa obter o número de caminhos independentes no código-fonte, para estabelecer o número mínimo de casos de teste que devem ser construídos para testar adequadamente um programa.

Tabela 3. Complexidade Ciclomática do software *Compiere.*

Métricas	Resultado
Total de Pacotes	66.
Total de Métodos	37.347.
Complexidade Ciclomática (Média)	2,14.

Foi então aplicado um XSLT para extrair a complexidade ciclomática através da totalização das disjunções (if, case, for, while) somando mais uma unidade referente ao fluxo principal. Aplicando esta transformação à representação *CodeMi* do software *Compiere*, obtemos a complexidade ciclomática média do software igual à 2,14, conforme apresentado na tabela 3. Esta média foi obtida através da divisão da complexidade ciclomática total pelo número de métodos, resultando numa média de 2,14 disjunções/método. Assim, seria necessário em média aplicar 2,14 casos de testes para cada método do sistema *Compiere* visando assegurar que todos os caminhos (fluxos) foram testados.

Além do XSLT, outras ferramentas de manipulação de XML podem ser utilizadas para extrair métricas desta representação, como: XPath [XPath 1999] e XQuery [XQuery 2007]. Para consultas simples de um determinado caminho ou somatório de marcadores podem ser utilizadas consultas no XPath. Já as consultas mais complexas e que envolvam mais de um arquivo XML normalmente são utilizadas consultas XQuery.

6. Conclusões

Conforme proposto inicialmente, apresentamos a representação CodeMI com o objetivo de fornecer uma representação de código-fonte orientado a objeto capaz de armazenar informações estruturais de código-fonte com baixa verbosidade, facilitando sua leitura e não revelando toda a sintaxe implementada pelos comandos dos métodos e construtores

A baixa verbosidade é garantida pelo conjunto reduzido de marcadores para representar a implementação interna ao corpo dos métodos e construtores. Outra característica apresentada pela representação é a impossibilidade de extração do código-fonte, pois nenhuma transformação XSLT sobre esta representação será capaz de restaurar o código-fonte original do sistema, favorecendo a análise de repositórios de sistemas industriais.De posse do arquivo em CodeMI, foi possível obter a medida de complexidade ciclomática média de uma classe através de transformação XSLT. Isto fornece indícios da possibilidade de extração de métricas estruturais sobre esta representação.

Como proposta de trabalho futuro, estamos ampliando o domínio de medidas passíveis de ser obtidas a partir da CodeMI, a partir do estudo de novas métricas (suite de Lorenz e Kidd, Chidamber e Kemerer) e criação de novas transformações XSLT. Novos marcadores e atributos poderão surgir desde que a CodeMI não perca suas características essenciais.

Agradecimentos

Os autores gostariam de agradecer e reconhecer o suporte financeiro a esta pesquisa dado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e pela Fundação de Apoio à Pesquisa do Estado do Rio de Janeiro (FAPERJ).

Referências

Aguiar A., David G., Badros G. J., “JavaML 2.0: Enriching the Markup Language for Java Source Code”, in XML: Aplicações e Tecnologias Associadas (XATA’2004), Porto, Portugal, Fevereiro de 2004.

Badros G. J., “JavaML: A Markup Language for Java Source Code”, In Proceedings of the 9th Int. Conf. On the World Wide Web (WWW9), Amsterdam, Netherlands, May 2000.

Holt, R. C., Winter, A., Schürr, A. “GXL: Toward a Standard Exchange Format”, in Proc. of the 7th Working Conf. on Reverse Engineering (WCRE’00), Brisbane, Australia, pp. 162–171, November 2000.

Levine J. R., Lex & YACC. O’Reilly & Associates, Inc., Sebastopol, California, 2nd edition, 1992.

Maletic, J.I., Collard, M.L. and Marcus, A., “Source Code Files as Structured Documents”, in Proc. of the10th Int. Workshop on Program Comprehension (IWPC ’02), Paris, France, pp. 289–292, June, 2002.

Mamas, E. and Kontogiannis, C., “Towards Portable Source Code Representations Using XML”, in Proc, of the 7th Working Conf. on Reverse Engineering (WCRE’00), Brisbane, Australia, pp. 172-18, November 2000.

McCabe, T.J., “A Complexity Measure”, IEEE Transactions on Software Engineering, 1976.

Mendonça, N. C., Maia, P. H. M., Fonseca, L. A., and Andrade, R. M. C. (2004), “RefaX: A Refactoring Framework Based on XML”, In 20th. IEEE International Conference on Software Maintenance (ICSM 2004), pages 147-156, IEEE Computer Society.

MOF. Meta Object Facility Specification, Version 1.4. Object Management Group 2002a.

Seguin, C., JRefactory Home. ACM, 2000. Disponível em: http://jrefactory.sourceforge .net

Xjava 1.1. BeautyJ Home Page. Disponível em: http://beautyj.berlios.de/beautyJ.html. Acessado em: 11/06/2008.

XMI 2.1. XML Metadata Interchange Specification, Version 2.1. OMG Document Number: formal/2007-12-01. Disponível em: http://www.omg.org/spec/XMI/2.1 /PDF

XML. Extensible Markup Language (XML). W3C Recommendation 16 august 2006. Disponível em: http://www.w3.org/TR/REC-xml/

XPath 1.0: XML Path Language. W3C Recommendation 16, November 1999. Disponível em: http://www.w3.org/TR/xpath

XQuery 1.0: An XML Query Language. W3C Recommendation 23, January 2007. Disponível em: http://www.w3.org/TR/xquery/

XSLT 2.0: Extensible Stylesheet Language for Transformation. W3C Working Draft 14 February 2001. Disponivel em: http://w3.org/TR/xslt20req

UML 2.1.2. Unified Modeling Language. Version 2.1.2. OMG Document Number formal/2007-11-04. Disponível em: http://www.omg.org/spec/UML/2.1.2/