FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA – UNIFOR
Centro de Ciências Tecnológicas – CCT

Leonardo Abreu Fonseca

XCARE - UM FRAMEWORK BASEADO EM XML PARA O DESENVOLVIMENTO DE FERRAMENTAS DE ANÁLISE DE CÓDIGO

Fortaleza

Março de 2005

FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA – UNIFOR
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT

Leonardo Abreu Fonseca

XCARE - UM FRAMEWORK BASEADO EM XML PARA O DESENVOLVIMENTO DE FERRAMENTAS DE ANÁLISE DE CÓDIGO

Dissertação apresentada ao Curso de Mestrado em Informática Aplicada da Universidade de Fortaleza como requisito parcial para obtenção do Título de Mestre em Informática Aplicada.

Orientador: **Prof. Dr. Nabor das Chagas Mendonça**

Fortaleza, Março de 2005.

À minha esposa
Erika Nattrodt Barros Soares,
Minha amada filha Carolina Nattrodt Abreu,
à minha mãe
Maria do Carmo Abreu Fonseca
e familiares pelo amor,
incentivo e compreensão.

Agradecimentos

Agradeço primeiramente a Deus por ter me dado a oportunidade por estar participando de um mestrado, apesar de todas as dificuldades, do tempo escasso para o correto desempenho como estudante, mas sempre procurei me esforçar para tentar cumprir os objetivos almejados.

Agradeço a meu orientador, Nabor das Chagas Mendonça, que com muita paciência soube me orientar durante todo o tempo em que solicitei a sua ajuda.

Agradeço a empresa em que eu trabalho, SERPRO – Serviço Federal de Processamento de Dados, pela aposta feita em mim para o cumprimento desta meta.

Agradeço ao meu chefe, Carlos Augusto Parente Neiva Santos, que me incentivou desde o ínicio para a participação e engrandecimento de minha carreira profissional, pessoa esta que para mim é um exemplo de dedicação, de companherismo e de liderança, bem como exemplo de atitudes e ações para o desenvolvimento das pessoas de seu convívio.

Agradeço aos meus amigos de trabalho (Micheline Araújo, Clayton Silva, João Daniel Alves), que me ajudaram nos momentos difíceis no trabalho, em que precisava estar no mestrado.

Agradeço ao José de Aguiar Moraes, que tenho como exemplo de dedicação, de aluno exemplar, de modelo a ser seguido como pesquisador, onde este me motivou de várias formas, onde fizemos trabalhos juntos, e este nunca me deixou de ser solicito.

Gostaria de agradecer também a uma pessoa muito especial, Paulo Henrique Maia, onde fizemos trabalhos de pesquisa juntos e que me ajudou e acompanhou nesta caminhada, me incentivando para que eu terminasse esta meta.

Não poderia deixar de agradecer a minha amada esposa Érika, que me incentivou desde o inicio do mestrado, tendo a paciência para entender os momentos de dificuldade, onde de certa forma o mestrado acompanhou boa parte de nosso convívio, desde o casamento até a gravidez da nossa filha Carolina, que em 23 de

Janeiro de 2005 nasceu para a nossa felicidade e amor. Carolina é a razão para que nunca tenha desistido dos meus sonhos, mesmo os mais difíceis.

Agradeço a minha amada mãe, Maria do Carmo Abreu, que como sempre me dava forças para eu não desistisse de terminar o mestrado, e sempre me perguntava (diariamente) e “ai terminou a tese meu filho? Termine logo !!!”.

Agradeço a meu pai, Matheus Ferreira Fonseca, que me deu a vida, o amor, apesar de distante, e a nova família que ele constituiu, com minha irmãzinha Mariana e meus dois irmãos Matheus e Marcos.

Agradeço aos meus sogros, José Soares e Vera Nattrodt que também me incentivaram a conclusão do mestrado, compreendendo às vezes em que não pude dar-lhes atenção, por exemplo, ir ao Pecém ou a casa deles.

Agradeço a minha cunhada Patrícia Nattrodt e ao meu cuncunhado Adriano Sá, que também me deram força para perseguir os objetivos de concluir um mestrado, e que me forneceram todas as dicas para a obtenção deste, bem como livros e exemplos de teses.

Agradeço também a família de minha esposa, vô, vó, tia Celina, tio Fernando que me incentivaram para o termino do mestrado.

Por fim, agradeço a todos que de certa forma fizeram, fazem, e farão parte dos sucessos e dos fracassos em minha vida, alegrias e tristezas. Todos vocês são muito importantes para mim, sem vocês eu nada seria.

Leonardo Abreu Fonseca

Resumo

Há um crescente interesse dentro da comunidade de engenharia de software no uso da linguagem XML e suas tecnologias como uma maneira de facilitar a implementação, reuso e integração das ferramentas de suporte ao processo de desenvolvimento de software. No entanto, os trabalhos realizados nesta área têm-se concentrado na definição de padrões XML para representação de código e na construção de ferramentas para extrair estas representações de forma automática dos artefatos de software de um sistema.

A implementação da maioria das ferramentas de desenvolvimento atuais, em particular das ferramentas de análise de código fonte, ainda continua sendo feita de forma “fechada”, isto é, suas estruturas internas e rotinas de manipulação de dados não são padronizadas e, por esse motivo, são de difícil utilização. Claramente, esta característica restringe as possibilidades de customização destas ferramentas pelos seus usuários, além de dificultar o seu reuso no contexto de novos ambientes de desenvolvimento.

Este trabalho propõe um ambiente baseado em padrões e tecnologias XML para facilitar o desenvolvimento de ferramentas de análise de código. Este ambiente, denominado XCARE (*XML-based Code Analysis and Reverse Engineering*), emprega tecnologias XML não apenas como mecanismos para importação e exportação de dados de código fonte, mas também para implementar as próprias estruturas de representação e manipulação desses dados.

A flexibilidade do ambiente foi demonstrada através da implementação de uma variedade de operações de análise de código para a linguagem Java, incluindo métricas, críticas de projeto e engenharia reversa. Experimentos foram realizados utilizando dados XML extraídos do código fonte de aplicações Java disponíveis publicamente, tais como Eclipse e Jdk, o que permitiu avaliar aspectos de desempenho e escalabilidade do framework quando instanciado com diferentes tipos de ferramentas de consulta a dados XML.

Abstract

There is an increasing interest within the software engineering community in the use of the XML language and its related technologies as a means to facilitate the implementation, reuse and integration of software development tools. However, work in this area has thus far focused on the definition of XML-based source code representations, and the implementation of automated tools for extracting those representations from source code artifacts.

Most current software development tools, particularly code analysis tools, are still implemented in a "closed" fashion, that is, their internal data structures and manipulation routines are not standardized and, for that reason, are extremely difficult to reuse. Clearly, such characteristic restricts the capacity of the users of customizing those tools for their specific needs, and makes them difficult to reuse in the context of new software development environments.

This dissertation proposes an XML-based software development framework aimed at facilitating the development of code analysis tools. This framework, called XCARE (XML-based Code Analysis and Reverse Engineering), uses XML-base technologies not only as the mechanism to import and export source code data, but also to implement the structures and routines necessary to manipulate those data.

The framework's flexibility was demonstrated through the implemenation of a variety of code analysis operations for the Java programming language, including software metrics, design critics, and reverse engineering. Some experiments were also carried out using XML data extracted from the source code of publicly-available Java applications, such as Eclipse and JDK, which allowed us to evaluate the framework's performance and scalability when instantied using different XML query technologies.

Índice

Índice de Figuras

Índice de Tabelas

CAPÍTULO 1 -Introdução

1.1 Motivação

A manutenção de software é uma tarefa difícil e cara. Estima-se que cerca de 50% do tempo de um engenheiro de software é gasto com tarefas de manutenção e compreensão de código, e que ao longo das últimas três décadas mais de 60% dos custos de desenvolvimento de software das organizações foram gastos com manutenção [45]. Um grande problema na atividade de manutenção é a necessidade de acessar, organizar, e atualizar uma enorme quantidade de informações relacionadas ao código fonte do sistema, tais como, métricas, grafos de chamada, dependência de dados, padrões de projeto, etc.

Para o auxílio na tarefa de manutenção de software, vários tipos de ferramentas de análise de código têm sido propostos (por exemplo, [27][37][48][49][56][74]). Apesar da grande variedade de ferramentas disponíveis, é difícil encontrar uma ferramenta, ou um conjunto de ferramentas, que satisfaça plenamente a necessidade de diferentes tipos de usuários e situações de uso, haja vista a sua grande diversidade. Como conseqüência, existe uma crescente demanda por ferramentas de desenvolvimento que sejam mais facilmente extensíveis e customizáveis.

1.2 Problema

Na maioria das ferramentas de análise de código existentes, tanto suas estruturas de dados internas quanto as suas rotinas para manipulação desses dados não estão facilmente acessíveis a seus usuários. Os usuários invocam as operações de consulta e manipulação de dados oferecidos pela ferramenta, através de uma interface gráfica ou de programação, mas não têm acesso aos programas que implementam essas operações. Claramente, isto limita sobremaneira a capacidade de se customizar essas ferramentas para usuários e domínios específicos, além de dificultar o seu reuso no contexto de novos ambientes de desenvolvimento.

Alternativas têm sido tradicionalmente consideradas na busca por ferramentas de análise de código mais facilmente reusáveis. Uma alternativa consiste no desenvolvimento de ferramentas de código aberto cujo código fonte esteja livre para ser acessado e modificado por qualquer programador ou usuário interessado. A maior desvantagem, nesse caso, está na necessidade do usuário ter um conhecimento detalhado tanto da linguagem de programação da ferramenta quanto de suas rotinas e estruturas internas.

Outra alternativa baseia-se no desenvolvimento de ferramentas configuráveis pelo usuário final, ou seja, ferramentas cujo comportamento possa ser alterado sem a necessidade de modificação no seu código fonte (por exemplo, Refine[52]). O problema dessa solução é que, em geral, as possibilidades de alteração disponíveis para o usuário são bastante restritas, limitando-se a mudanças de “alto nível” que não têm acesso às estruturas internas da ferramenta.

Uma terceira alternativa, que tem sido foco de algumas pesquisas recentes, consiste na utilização da linguagem XML(Extensible Markup Language)[61] e de seus padrões e tecnologias como uma maneira prática para extrair, representar e facilitar a troca de informações de código fonte. Esta abordagem é particularmente atraente para o desenvolvimento de ferramentas de análise de código, uma vez que os desenvolvedores das ferramentas podem beneficiar-se da abundância de tecnologias de consulta e manipulação de dados XML atualmente disponíveis.

Além disso, XML viabiliza a troca de dados entre computadores em redes heterogêneas, é uma linguagem padrão da Web para intercâmbio de dados, possui

baixo custo na recuperação das informações, uma vez que informações de um SGBD podem ser acessadas por outros computadores que não possuam esse tipo de software.

A troca de informações via XML se torna mais rápida devido à ausência de formatação do arquivo. As informações representadas no XML são contextuais, ou seja, estão dentro das marcações que dão mais semântica aos dados do documento.

Entretanto, a maioria dos trabalhos realizados nesta área tem-se concentrado na definição de padrões XML para representar informações de código fonte e na construção de ferramentas automáticas para extrair estas representações a partir de artefatos de código (por exemplo, JavaML [1][14][37], BeautyJ [28], DoctorJ [26], MultiJava [4], CppML [37], OOML [37], etc.).

A implementação das operações de análise de código ainda não tem aproveitado as vantagens e as ferramentas disponíveis para a manipulação de XML, onde este constitui apenas um mecanismo de importação e exportação de dados.

1.3 Objetivos

Neste trabalho, propomos um *framework* para o desenvolvimento de ferramentas de análise de código baseado no uso de padrões e tecnologias XML. Este framework, denominado XCARE (XML-based Code Analysis and Reverse Engineering), utiliza tecnologias de consulta e manipulação de dados XML (em particular a linguagem de consulta XQuery[63]) para implementar operações de análise sobre informações de código fonte representadas em algum padrão XML proposto para este fim (por exemplo, JavaML[1][14][37], CppML[37], OOML[37], etc.).

Entre as operações de análise que foram implementadas neste trabalho destacam-se métricas, críticas de projeto, e operações de engenharia reversa.

As operações baseadas em tecnologia XML podem ser mais facilmente especificadas, estendidas e reutilizadas em diferentes contextos. Por exemplo, uma operação de engenharia reversa para recuperação das dependências entre classes de um sistema, uma vez especificada como uma consulta XQuery para um padrão XML específico de representação de código Java, poderia ser reutilizada para o uso

em qualquer sistema Java cujo código fonte esteja representado neste padrão XML, independentemente das ferramentas utilizadas para extração desta representação de código, e para processamento da consulta XQuery, respectivamente.

O framework XCARE possibilita a construção de ferramentas para a análise de código baseadas na tecnologia XML. Este possui classes e interfaces que possibilitam a construção de instâncias de aplicações para a análise de código e engenharia reversa. Para a construção destas instâncias faz-se necessária a implementação de interfaces contidas no framework para execução das atividades de conversão do código fonte em XML, armazenamento dos dados XML, carregamento ou inserção de consultas aos dados e execução das consultas sobre os dados XML.

Uma instância do framework, chamada JCARE – XCARE para Java, foi avaliada com relação ao desempenho das ferramentas que esta instância utilizava, com o objetivo de comprovar a viabilidade da execução das tarefas e, conseqüentemente, do framework. Para a avaliação de desempenho, utilizaram-se aplicações de tamanho real com ferramentas nativas de banco de dados XML contendo a linguagem de consulta XQuery e também com protótipos de ferramentas XQuery que armazenam e acessam os dados em memória.

1.4 Trabalhos Relacionados

Em Riva e Yaojin[3] é mostrado um ambiente de recuperação de arquitetura no qual os modelos do código fonte são extraídos por analisadores e armazenados em formato GXL. Em seguida, um sistema em Prolog recebe os modelos em GXL, processa-os e gera o modelo de arquitetura, também em GXL. A conversão dos dados para outro formato, para fins de visualização ou processamento, é feita através de scripts XSLT. GXL tem o papel de repositório de dados relacionais.

O ambiente ISME (Integrated Software Maintenance Environment) [37] permite a representação de código fonte num formato de árvores DOM e provê mecanismos de registro de ferramentas CASE e de tipos de representação de código baseados em XML. Com isso este ambiente pode facilmente criar ferramentas para determinada linguagem trocando as peças (ferramentas) na extração de dados e nas ferramentas CASE de manipulação. Numa instanciação do ambiente, o usuário

entra com um código JavaML ou CppML, que então é convertido para o padrão OOML e armazenado em um banco relacional DB2 através de um script conversor de XML para DB2. A ferramenta CASE realiza métricas sobre os dados. Os resultados podem ser visualizados através de Regi ou Together C++.

O trabalho mais próximo do nosso é mostrado em [36]. Os autores desenvolveram uma ferramenta para extração de de partes de informações de código C++ usando uma representação em XML do mesmo em srcML, e como linguagem de consulta XPath. Contudo, essa abordagem se diferencia da nossa na medida em que os autores se limitaram apenas à extração de fatos, e nossa ferramenta realiza, além disso, tarefas de engenharia reversa, métricas e críticas. Outra diferença é a utilização, por nossa parte, da linguagem de consulta que é a padrão da W3C, XQUERY.

Embora a utilização de XML por essas ferramentas sela um passo para tornar o ambiente mais flexível e extensível, a estrutura de dados interna continua como uma caixa preta para os usuários, ficando estes impossibilitados de fazer qualquer modificação a fim de customizar a ferramenta.

1.5 Organização do Documento

Este documento está organizado da seguinte forma: no primeiro capítulo, apresentamos o problema, objetivos gerais e a motivação da dissertação. O segundo capítulo abordará padrões de representação em XML e representações de algumas linguagens de programação nesta tecnologia. Ao final do segundo capítulo serão discutidas as vantagens e desvantagens de se utilizar XML para a representação de código.

O terceiro capítulo apresentará tecnologias de consulta e manipulação de dados XML tendo em vista a utilização das representações de código, pela mesma tecnologia, na análise e engenharia reversa. O quarto capítulo descreverá o framework XCARE, que se baseia na tecnologia XML e que possibilita a construção de ferramentas para auxílio na análise de código e engenharia reversa. Também serão mostrados a arquitetura, seus papéis e serviços, como criar novas instâncias do framework, JCARE, uma instância do framework para Java e, por fim, as limitações do framework.

O quinto capítulo mostrará as considerações de desempenho sobre as ferramentas que são utilizadas na instância JCARE e comparativos com duas ferramentas de armazenamento em banco XML nativos e outras duas de armazenamento em memória.

Finalmente, no sexto capítulo serão apresentados a conclusão, as contribuições e os trabalhos futuros.

CAPÍTULO 2 - Padrões para Representação de Código em XML

2.1 Introdução

Um dos problemas encontrados em projetos de reengenharia de software é o fato de que o código fonte sempre é representado e armazenado no formato textual ASCII. Este formato é de difícil manipulação para a realização da análise de código sem o uso de ferramentas adequadas.

Como forma de deixar o código fonte portável e mais fácil de ser manipulado por ferramentas de análise, surgiu a necessidade de se definir representações de código padronizadas. Sabendo que a troca de informações se torna mais rápida e objetiva através de XML, uma vez que as informações ficam contidas dentro de marcações que dão mais semântica aos dados do documento, muitas representações têm sido propostas baseadas nessa linguagem.

2.2 Tipos de Representações

Nesta seção, apresentaremos algumas das representações em XML que foram propostas, algumas das quais já se tornaram padrões reconhecidos pelos engenheiros de software.

As representações de código podem ser divididas em generalistas e específicas. As representações generalistas são aquelas onde se procura agrupar numa única representação XML um universo de características comuns de linguagens de programação. As específicas representam a seu modo as estruturas de uma determinada linguagem de programação.

Existem também outros tipos de representação em XML como as representações de projeto (XMI [44]) e as representações de grafos, que podem ser interpretadas como representação de código em alto nível.

Nas subseções seguintes, para exemplificar as representações de código, estaremos usando um pequeno trecho de código Java, conforme mostra a figura 1.

```
public class Car{
     int color;
     public int getColor(){
            return color;
     }
}
```

Figura 1: Exemplo de código fonte em Java.

2.2.1 Representações Específicas

CppML

CppML [37] é uma representação da estrutura sintática de C++ em XML proposta por Evangelo Mamas. Alguns dos objetivos desta representação foram melhorar e incrementar a compreensão do código facilitar o processamento automático do código já que muitas ferramentas estão disponíveis publicamente para a manipulação de dados XML.

JavaML de Evangelo Mamas e Kontogiannis

Mamas e Kontogiannis propõem em [37] três formatos para representação de código: JavaML, CppML e OOML. Os dois primeiros representam as informações de códigos escritos em Java e C++, respectivamente, em forma uma árvore sintática abstrata (AST), sendo esta estrutura representada em um formato XML.

O terceiro formato generaliza os outros dois com os elementos comuns entre as duas linguagens. A figura 2 mostra a representação JavaML correspondente ao código Java da figura 1. As ferramentas utilizadas para a geração da representação JavaML são baseadas no gerador de parser JavaCC[25].

```
<ClassDeclaration Identifier="Car">
   <FieldDeclaration>
         <PrimitiveType Type="int" />
         <VaiableDeclaratorId Identifier="color" />
   </FieldDeclaration>
   <MethodDeclaration Identifier="getColor">
     <ResultType>
         <PrimitiveType Type="int" />
     </ResultType>
     <Block>
         <ReturnStatement>
              <PrimaryExpression>
                   <Name Identifier="color" />
              </PrimaryExpression>
         </ReturnStatement>
     </Block>
   </MethodDeclaration>
</ClassDeclaration>
```

Figura 2: Representação de Car.java em JavaML de Mamas e Kontogiannis.

JavaML de Badros

Badros apresenta um outro padrão para representar código Java em um formato XML, também chamado JavaML [1]. Assim como os citados acima, JavaML de Badros se preocupa em manter informações da AST do código fonte no formato XML. Contudo, esse formato, em comparação especificamente aos seus homônimos, representa o código de forma mais simples e completa, facilitando a especificação das operações de consulta e manipulação sobre os elementos de código representados em XML.

A figura 3 mostra o mesmo código da figura 1 representado no formato JavaML de Badros. O conversor de Java para JavaML foi implementado pelo próprio Badros modificando o compilador Jikes da IBM.

```
<java-source-program>
<java-class-file name="Car.java">
<class name="Car" visibility="public" line="1" col="0" end-
   line="7" end-col="0">
<superclass name="Object" />
<field name="color" line="3" col="0" end-line="3" end-
   col="9">
<type name="int" primitive="true" />
</field>
<method name="getColor" visibility="public" id="Car_mth-9"
   line="5" col="0" end-line="6" end-col="15">
<type name="int" primitive="true" />
<formal-arguments />
<block line="5" col="21" end-line="6" end-col="15">
<return>
<var-ref name="color" />
</return>
</block>
</method>
</class>
</java-class-file>
</java-source-program>
```

Figura 3: Representação de Car.java em JavaML de Badros.

MultiJava

MultiJava[4] é uma extensão de um projeto de JDE em XML, onde ambos foram projetados por Curtis Clifton e Gary T. Leavens na Iowa State University. MultiJava é um parser Java que tem capacidade de gerar arquivos XML. Foi escrito em Java baseado na API Xerces do Apache e na ferramenta de tradução-geração ANTLR. Um *Schema* XML é também proposto para a codificação da linguagem Java em XML. Na figura 4, podemos observar o código Java da figura 1 representado em MultiJava.

```
public class Car <classBody>
 public int getColor() <statement>
  <statBlock>
   <statement>
    <statReturn>
     <expression>
      <fieldRefExpression>
       <fieldRefExpr>
        <expression>
         <thisExpression>
         </thisExpression>
        </expression>
        <identifier>color</identifier>
       </fieldRefExpr>
      </fieldRefExpression>
     </expression>
    </statReturn>
   </statement>
  </statBlock>
 </statement>
 int color;</classBody>
```

Figura 4: Representação de Car.java MultiJava.

DoctorJ e BeautyJ

DoctorJ[26] é uma ferramenta de análise de código aberto desenvolvida por Jeff Pace, com o objetivo de analisar o código fonte Java. Esta ferramenta gera a AST de um programa Java em um arquivo de saída XML criado pelo analisador sintático.

BeautyJ[28] é outra ferramenta de transformação e análise de código desenvolvida por Jens Gulden. Esta possui um parser Java que transforma o código fonte em diferentes formatos de saída. O código fonte pode ser automaticamente normalizado a um formato estrutural limpo (“Beautified”) ou ser convertido num formato XML denominado XJava.

Este arquivo XML possui as classes ou interfaces e seus membros associados com o código fonte e seu comentário. A figura 5 mostra a representação do código fonte Java da figura 1 em BeautyJ.

```
<?xml version="1.0" encoding="UTF-8"?>
<xjava>
<class name="Car" public="yes" unqualifiedName="Car">
<extends class="java.lang.Object"/>
<field name="Car.color" packageprivate="yes"
unqualifiedName="color">
<type dimension="0" fullName="int" name="int"
unqualifiedName="int"/>
</field>
<method name="Car.getColor" public="yes"
unqualifiedName="getColor">
<type dimension="0"  fullName="int" name="int"
unqualifiedName="int"/>
<code>return color;</code>
</method></class></xjava>
```

Figura 5: Representação de Car.java em BeautyJ.

2.2.2 Representações Generalistas

OOML – Object Oriented Markup Language (OOML)

Conforme descrito na seção anterior esta proposta surgiu de uma pesquisa de Evangelo Mamas e Kontogiannis[37] sobre uma representação comum para duas linguagens orientadas a objeto: C++ e Java. Nessa pesquisa, Mamas procurou unir as características destas duas linguagens e propôs uma linguagem de marcação com representações para seus elementos em comum.

Na tabela 1 podemos observar alguns elementos em comum das duas linguagens e a sua representação correspondente em OOML.

Java	C++	OOML
CompilationUnit	Program	Program
CompilationUnit.source	PrimarySource, IncludedSource	Source
ImportDeclaration	Include	Include
ClassDeclaration	Class	Class
MethodDeclaration	Function	Method
FieldDeclaration	Variable	VariableDeclaration
Block	LexicalBlockStatement	Body
SwitchStatement, IfStatement	SwitchStatement, IfStatement	ConditionalStatement
DoStatement, ForStatement, WhileStatement	DoStatement, ForStatement, WhileStatement	Loop
Name	NameExpression	VariableUse
Name	FunctionCallExpression	MethodCall
Type	TypeDescriptor	Type
FormalParameter		Parameter

Tabela 1: Mapeamento de alguns elementos de Java e C++ para OOML.

srcML – Source Markup Language

Maletic [32] propôs uma representação baseada na inclusão de marcações (tags) XML ao próprio código fonte, denominada srcML. A ferramenta desenvolvida em sua pesquisa transforma o código fonte da linguagem C para o formato srcML.

Segundo o autor, enquanto o código fonte está intrinsecamente estruturado, a maneira na qual ele é armazenado quase não tem estrutura, isto é, o código é armazenado simplesmente como texto. Uma solução para o problema vista pelo autor era adicionar informação estrutural dentro do texto, inserindo, para isso, caracteres especiais ou marcações. Com isto, o texto poderia ser facilmente consultado, analisado gramaticalmente e transformado com a ajuda destas marcações.

Diferentemente dos formatos vistos anteriormente, srcML não representa a AST do código, mas sim o "anota" de forma estruturada, preservando informações como espaços em branco, comentários e indentações. Sua vantagem é que não é necessário fazer um *parsing* do código para convertê-lo para XML. A desvantagem é que a representação não é tão precisa sintaticamente quanto àquelas que representam diretamente a AST.

2.2.3 Representações de Alto Nível

GXL – Representação de Grafos

GXL (Graph eXchange Language) [21][12] é um padrão baseado em XML para a troca e compartilhamento de informações de código fonte entre ferramentas. Formalmente, GXL representa grafos que possuem tipos, atributos, direção e ordenação e que podem ser estendidos para representar hiper-grafos e grafos hierárquicos.

Este padrão foi desenvolvido num esforço entre grupos de pesquisa internacionais de várias áreas, incluindo reengenharia de software e transformação de grafos. Além de ser um formato genérico para a representação de estruturas de grafos, GXL pode ser utilizado para representar dados do tipo objeto-relacional. Conseqüentemente, GXL pode ser usado para representar dados de uma série de aplicações incluindo

as de repositório de dados e as bases de partes de informações de código das ferramentas de engenharia reversa.

Diferentemente da estratégia assumida pelo XML Meta Data Interchange(XMI) [44], que será apresentado a seguir, os esquemas GXL não são traduzidos em documentos XML baseados em MOF (Meta Object Facility). A abordagem MOF de representação de diagramas de classe UML cria um novo DTD (Document Type Definition é o arquivo que contém as regras que definem quais as tags que podem ser usadas em um documento XML e quais os valores válidos) para cada tipo de diagrama. Ao contrário, GXL usa o mesmo DTD para todos os esquemas. Conseqüentemente, este fato permite que haja uma grande compatibilidade entre as ferramentas que manipulam GXL.

Existem várias ferramentas que utilizam o formato GXL de alguma maneira, tais como: conversores (FAMIX [11] e GraLab [17]), análise de grafos e ferramentas de transformação (GenSet [16] e GRAS [19]), ferramentas de visualização (Graph Tool [18] e Jgraph [29]), editores de grafos (Graphpad [20] e Jgraphpad [30]), extratores de código fonte (cppx [6] e Source Navigator [54]) e ferramentas de reengenharia de software (Fujaba [13], Rigi [50] e Venice [60][59]).

XMI – Representação de Diagramas de Projeto UML

O principal objetivo da XMI [44] é facilitar o intercâmbio de metadados, termo genérico para qualquer dado que de alguma forma descreve uma informação, entre ferramentas de modelagem (baseadas na UML - Unified Modeling Language [58]) e repositórios de metadados (baseados no MOF - Meta Object Facility [38]) em um ambiente heterogêneo. O XMI integra três padrões chaves da indústria de software: XML, padrão do W3C; UML – padrão da OMG, que define uma linguagem de modelagem orientada a objetos, suportada por inúmeras ferramentas CASEs; e MOF – um padrão da OMG, que é uma estrutura de definição de modelos de metadados e fornece ferramentas para interfaces programáveis de armazenamento e acesso de metadados em repositórios.

A integração destes três padrões dentro do XMI une o que há de melhor das tecnologias de metadados e modelagem da W3C e da OMG, permitindo que desenvolvedores de sistemas compartilhem modelos de objetos e outros metadados.

Com a utilização da UML, diferentes definições de modelos podem interoperar utilizando o padrão XMI.

2.3 Vantagens e Desvantagens das Representações

Um dos benefícios principais da representação das linguagens de programação em XML é pelo fato da existência de uma enorme gama de ferramentas para as tarefas de visualização, análise e manipulação. Algumas ferramentas como Internet Explorer, XML Spy[70], XML Notepad[69], IDE Eclipe para Java[22], podem ser usadas para a visualização da estrutura ou busca de elementos dentro da árvore XML.

Um exemplo simples da importância do uso da representação das linguagens em XML é quando desejamos renomear o nome de um determinado método Java e todas as suas referências, tais como chamadas a este método por outras classes. Se fôssemos tentar realizar tal tarefa no código fonte (de forma textual) através do uso do "localizar e substituir" das ferramentas textuais, poderíamos cometer enganos caso o nome do método coincidisse com um nome de uma variável local ou um comentário da documentação do código.

Podemos descrever algumas vantagens do uso de XML para a representação de código:

• XML já é um padrão aberto independente de plataformas ou empresas.

• XML suporta os padrões ISO Unicode para a escrita de caracteres.

• Existem várias APIS (DOM, SAX, WebDAV) para a criação, visualização e integração de XML.

• O custo de disponibilizar informações em XML é baixo.

• Os documentos XML podem ser criados facilmente usando um editor de texto comum.

• A estrutura de marcação e a sintaxe textual dão uma maior facilidade de leitura.

• Uma série de ferramentas gratuitas para a consulta, armazenamento e manipulação de XML já pode ser encontrada na Internet.

• XML possui uma linguagem de consulta em vias de padronização pelo W3C, onde vários fabricantes já lançaram ferramentas com protótipos desse padrão.

• A representação de código fonte XML permite a construção de consultas, a migração de linguagens (transformações) e refatoração.

Podemos também explicitar algumas desvantagens do uso de XML para a representação de código:

• Dependendo da representação de código fonte XML, esta pode gerar arquivos muito maiores do que os arquivos fonte originais.

• O tempo gasto na conversão do código fonte para XML é considerável com relação as outras etapas realizadas pelas ferramentas de análise de código. As ferramentas tradicionais apenas recuperam o código fonte e o analisam sem a necessidade de nenhuma conversão.

• Algumas ferramentas que consultam XML ainda possuem problemas de performance.

• A principal linguagem de consulta a XML (XQuery[63]) ainda não está padronizada totalmente e várias empresas já implementaram diferentes versões desta linguagem.

Sobre as representações específicas e generalistas, podemos apontar ainda que há uma vantagem em procurar englobar o maior número de atributos comuns entre linguagens, como foi o caso da OOML, pois isto permite que se possa construir uma única ferramenta de análise de código para diversas linguagens as quais utilizem a mesma representação generalista.

Caso utilizássemos representações específicas para cada linguagem teríamos que construir uma ferramenta de análise de código para cada representação. Entretanto à medida que as linguagens se diferenciam, fica difícil construir um meta modelo que represente todas as características de uma linguagem, dificultando portanto consultas específicas em um determinado paradigma da linguagem.

Para exemplificar essa dificuldade, podemos citar que a OOML procura agregar características de linguagens orientadas a objeto, como é o caso de Java e C++, não estabelecendo suporte a outras linguagens como, por exemplo, COBOL, Natural, Visual Basic.

Em virtude dessa dificuldade, as representações específicas atendem a maior parte das consultas, pois estas conseguem representar as características de uma determinada linguagem.

Nas linguagens de representação de alto nível, o objetivo é apresentar diagramas e modelos da solução de desenvolvimento, sem necessariamente especificar uma linguagem para isso.

Além disso, as consultas de alto nível permitem apenas uma visão de projeto sobre a solução a ser desenvolvida, não permitindo a busca por elementos relativos a aspectos específicos de implementação.

2.4 Conclusão

Vimos neste capítulo as principais linguagens e formatos existentes para a representação de código em XML e suas caracterísitcas. No próximo capítulo, serão apresentadas algumas das principais ferramentas de consulta e manipulação de dados XML disponíveis atualmente, e que poderão ser utilizadas no processo de instanciação do *framework* proposto nesta dissertação.

CAPÍTULO 3 - Tecnologias de Consulta e Armazenamento de Dados XML

3.1 Introdução

Com a transformação do código para o formato XML surge, portanto, a necessidade de mecanismos que acessem e manipulem esses dados. A seguir apresentamos algumas tecnologias que manipulam dados XML, mostrando suas principais características e ilustrando seu uso através de um exemplo comum.

3.2 Linguagens e Ferramentas de Consulta

XML se tornou um formato padrão para a troca de dados. Conseqüentemente, o desenvolvimento de linguagens para consulta a XML é de interesse de toda a comunidade que usa dados em XML, tanto pesquisadores como a indústria de software.

Algumas linguagens para a consulta de dados XML, tais como, Quilt[51], XML-QL[8], XQL[24], evoluíram para a linguagem XQuery[63] que está em vias de padronização pela W3C. Outras linguagens como XUPDATE[71], XPATH[62] e XSLT[64][65] são utilizadas para a manipulação de dados XML.

3.2.1 Linguagens de Consulta

XPath

XPath[62] é uma linguagem de consulta a dados XML utilizada dentro de outras tecnologias, como XSLT [64], ou usado como uma API para extrair partes de documentos XML para processamento posterior. Sua principal característica, que também é sua maior desvantagem é realizar apenas consultas simples, não permitindo ao usuário fazer consultas mais complexas, como consultas aninhadas ou condicionais. Além de descrever um caminho para percorrer a árvore XML, XPath também oferece manipulação com strings, números e expressões booleanas.

A figura 6 mostra um exemplo de consulta XPath para acessar todos os métodos de todas as classes de um programa Java representado no padrão JavaML de Badros[1]. A figura 7, por sua vez, mostra um exemplo que retorna o nome do primeiro atributo de uma classe “Car” representada no mesmo padrão.

```
/java/java-source-program/java-class-file/class//method
```

Figura 6: Consulta XPath que retorna todos os métodos de todas as classes de um programa Java.

```
/java-source-program/java-class-file/class[@name='Car']
//field[1]/@name
```

Figura 7: Consulta XPath que retorna o nome do primeiro atributo da classe “Car”.

XQuery

XML Query Language – XQuery [63] é uma linguagem de consulta projetada para ser concisa e ter uma sintaxe de fácil entendimento por parte dos usuários. É também flexível o bastante para consultar diferentes fontes de dados XML, estejam eles armazenados em arquivos ou em bancos de dados nativos. XQuery é uma extensão de XPath. Sendo assim, também utiliza o conceito de expressão de caminho para navegar em árvores XML.

Contudo, XQuery se diferencia por suas consultas serem compostas por expressões baseadas nas cláusulas FOR, LET, WHERE e RETURN (FLWR, onde se lê *flower*), que permitem que várias expressões sejam aninhadas dentro da mesma consulta, tornando-a mais legível, simples, e poderosa. Além disso, XQuery oferece operadores para ordenar os nós resultantes de uma consulta, operadores matemáticos como count e avg, permite realizar consultas condicionais através dos operadores IF/THEN/ELSE, e também encapsular consultas em funções, o que possibilita fazer consultas recursivas.

Com XQuery, é possível formatar o resultado de uma consulta de maneira que ele seja um documento XML totalmente diferente do de entrada. O mesmo não ocorre com XPath, já que o retorno de uma consulta é sempre um conjunto de nós da árvore XML ou o valor desses nós ou de algum de seus atributos, não permitindo que os mesmos sejam manipulados na própria linguagem. As crescentes aceitação e utilização de XQuery podem ser comprovadas através do desenvolvimento de protótipos que implementem a linguagem por grandes corporações, como Microsoft, Oracle e Software AG.

A figura 8 mostra um exemplo de consulta XQuery que não poderia ser facilmente expressa com XPath. A consulta define uma função que retorna o nome de todos os atributos da classe passada como parâmetro que são utilizados dentro de algum método, tomando como base a representação JavaML de Badros.

```
declare function get_used_fields($class as item()*) as
item()*
{
  FOR $field in $class//field,
      $var_ref in $class//method//block//var-ref
  WHERE $field/@name = $var_ref/@name
  RETURN
    $f/@name}
```

Figura 8: Função XQuery que retorna todos os atributos de uma classe que são utilizados dentro de algum método.

3.2.2 Ferramentas de Consulta

DERBY

Derby[7] é uma ferramenta que auxilia na análise de dados e em suas representações físicas. Esta é uma ferramenta de consulta baseada na linguagem XQuery que ajuda na seleção, na checagem e na transformação de dados XML.

GALAX

Galax[15] é uma implementação de código aberto de XQuery (versão 1.0). Galax se aproxima da definição de XQuery 1.0 como especificado pela W3C e também implementa XPath 2.0, que é um subconjunto de XQuery 1.0.

IPSI-XQ

IPSI-XQ[23] é outra implementação de XQuery. Esta ferramenta foi implementada o mais fielmente possível à especificação mais recente de XQuery disponível pela W3C. A ferramenta acompanha uma série de interfaces como ferramenta gráfica, linha de comando e interface Web. Possui também uma API Java para a integração com outras aplicações Java.

MSXQUERYDEMO

MSXQUERYDEMO[39] é uma ferramenta de demonstração da Microsoft que implementa XQuery. Esta consiste de uma série de bibliotecas de classes do framework.NET que implementam a linguagem de consulta da W3C, estão disponíveis através de uma interface Web.

QEXO

Qexo[46] é uma outra implementação parcial da linguagem XQuery. Esta ferramenta compila as consultas XQuery em bytecodes Java usando para isso o framework Kawa[34]. Qexo também inclui conceitos de XSLT implementados.

QIZX/OPEN

Qizx/Open[47] é uma implementação Java de código aberto da especificação XQuery de maio de 2003. Esta ferramenta suporta quase todas as consultas exemplo da W3C, com exceção de um exemplo que usa schemas. Algumas funcionalidades implementadas: módulo de importação, checagem estática de tipos, funções de extensão para saída SAX2, integração com JAXP e tratamento de erros.

QUIP

Quip[53] é um protótipo XQuery gratuito da Software AG. Quip foi projetado para tornar mais fácil o aprendizado e uso da linguagem XQuery. Este possui uma ferramenta gráfica para a escrita das consultas e a visualização dos resultados, além de uma interface com o servidor de banco de dados Tamino XML Server e uma API para Java. A especificação da linguagem usada na implementação é de Abril de 2002.

XQENGINE

XQEngine[72] (XML Query Engine) é uma ferramenta de consulta para documentos XML. Utilizando XQuery da W3C, este permite pesquisar coleções de documentos XML. XQEngine é disponível na forma de uma API para ser usado em aplicações Java.

3.3 Armazenamento de Dados XML

Apache Xindice

Apache Xindice[68] é um banco de dados XML nativo, de código aberto, desenvolvido pela Apache Software Foundation. O benefício de se usar um banco de dados nativo é que não há a necessidade de mapeamento dos dados XML com algum modelo de dados, principalmente em termos do armazenamento de documentos muito complexos. Da mesma forma que os documentos XML inseridos, estes são lidos ainda como XML. As linguagens com que este banco de dados trabalha são: XPATH[62] como linguagem de consulta e XML:DB XUpdate[71] como linguagem de atualização. Esta ferramenta possui uma API (XML:DB API) Java para o desenvolvimento de aplicações em Java. Existe uma outra forma de acesso ao banco, através de um plug-in para serviços Web, para outras linguagens não Java.

XStreamDB

XStreamDB™ 3.0[73] é um servidor de banco de dados XML nativo comercial para armazenamento e recuperação de documentos XML e com funcionalidades ideais para aplicações de gerenciamento de conteúdo, publicação na web e integração de aplicações distribuídas. Este possui uma arquitetura baseada em transações com suporte para XQuery com modificações para atualizações(update extensions), XML Schemas, índices, gerênciamento de recursos e compartilhamento WebDAV. O banco de dados pode ser acessado usando-se uma API Java, ou uma ferramenta de consulta e navegação de dados chamada XStreamDB Explorer.

Tamino – Software AG

O servidor Tamino XML Server[55] é um banco de dados XML nativo comercial que oferece uma série de funcionalidades relacionadas ao padrão de XML, tais como, XML Schema, XML namespaces e XQuery. Outras funcionalidades não relacionadas a XML como autenticação, replicação e clusterização, segurança, serviços de interface (APIs para manipulação) e suporte a transações distribuídas também estão disponíveis através do servidor.

X-Hive

X-Hive/DB[67] é um banco de dados que possibilita o armazenamento e a manipulação de alta velocidade e para uma grande quantidade de documentos XML. X-Hive/DB armazena documentos XML num banco de dados orientado a objetos integrado, de alta escalabilidade e performance. X-Hive/DB possui uma API em Java para a manipulação de seus dados via aplicação Java. X-Hive/DB implementa e estende as seguintes recomendações do W3C para consulta, recuperação, manipulação e escrita de documentos XML: Document Object Model (DOM), XSLT, XQuery, XPath, XLink e XPointer.

3.4 Conclusão

Neste capítulo apresentamos as principais tecnologias disponíveis atualmente para consulta e armazenamento de dados XML.

No próximo capítulo, será apresentado o framework XCARE, que possibilita a criação de ferramentas de análise de código de fácil customização utilizando as tecnologias descritas neste capítulo e no capítulo anterior.

CAPÍTULO 4 - O Framework XCARE

4.1 Introdução

Este capítulo apresenta o framework XCARE (acrônimo de "XML based Code Analysis and Reverse Engineering" [42]). Inicialmente é feita uma introdução sobre o que é o framework, incluindo aspectos de sua arquitetura, os padrões de projeto usados na sua implementação, e os papéis existentes para o reuso do framework. Em seguida, será mostrado como criar ferramentas de análise de código usando o framework, incluindo a descrição de uma instância do framework para análise de código na linguagem Java. O capítulo encerra com uma discussão das principais limitações do framework.

4.2 Visão Geral do Framework XCARE

XCARE é uma implementação em Java, organizada em conjunto de pacotes contendo classes e interfaces que compõem uma API em Java. Esta biblioteca Java é destinada especificamente para dar suporte às tarefas de análise de código e engenharia reversa de aplicações.

A API XCARE funciona como um modelo reusável onde o desenvolvedor de ambientes de análise de código pode selecionar quais ferramentas estará utilizando para suas tarefas.

As tarefas de análise de código e reengenharia envolvem:

- A conversão do código fonte para um formato de representação de código baseado em XML;
- Armazenamento dos documentos XML gerados em banco de dados nativo ou em arquivos;
- Consultas sobre os dados armazenados;
- Visualização dos dados e de suas dependências;

Para as tarefas de consulta utilizou-se a linguagem XQuery, descrita no capítulo 3. A razão da utilização de XQuery foi devido a dois aspectos. Primeiro por ser um padrão de consulta oficial do W3C. E segundo pelo número expressivo de ferramentas de software livre e também ferramentas comerciais que trabalham com esta linguagem.

Um ponto contrário ao uso da XQuery é a diversidade de implementações para essa linguagem, o que dificulta o desenvolvimento de bibliotecas de consultas que possam ser executadas em todas as ferramentas.

Para agrupar as consultas Xquery, o framework define o conceito de categorias de consultas. Estas podem ser divididas em métricas, críticas e consultas de engenharia reversa. Tais categorias são incorporadas num arquivo XML, chamado arquivo de categorias, onde ficam registradas todas as consultas por categoria. As consultas são armazenadas em arquivos separados e o endereço físico destas são encontrados no arquivo XML de categorias.

Um trecho do arquivo de categorias pode ser visto na figura 9. Apesar deste arquivo ter sido criado com apenas três categorias, o framework permite a criação de novas categorias a critério do gerador de ferramentas de análise de código. Isto será explicado nas seções seguintes.

Para o desenvolvimento do framework foram utilizadas as seguintes ferramentas: Java (versão J2SDK1.4.1_02), Eclipse (versão SDK-2.1.2-win32) e ArgoUML (versão.0.14), que é responsável pela geração dos diagramas de classe e de pacotes.

```
<Categories>
        <revengs>
          <reveng>
            <name>Get Methods name of Classes</name>
            <description>Get all methods name of classes
            </description>
            <path>../reveng/getMethodsname.xquery</path>
          </reveng>
                ...
        </revengs>
        <metrics>
                <metric>
                <name>WMC</name>
                <description>Weighted Methods per Class
                </description>
                <path>e:/temp/teste/wmc.xquery</path>
                </metric>
                ...
        </metrics>
        <critics>
                <critic>
                <name>Uninstantiable public</name>
                <description>Classes identificadas por essa critica nao podem ser
instanciadas externamente por outras classes, o que indica um potencial erro de projeto
                </description>
                <path>e:/categories/critics/critic1.xquery</path>
                </critic>
                ...
        </critics>
</Categories>
```

Figura 9: Arquivo XML que descreve as categorias de consultas no framework XCARE.

Na próxima seção serão abordados os aspectos da arquitetura e os padrões utilizados para o desenvolvimento do framework.

4.3 Arquitetura e Padrões Utilizados

4.3.1 Componentes de Arquitetura

O framework é dividido em quatro componentes principais. O primeiro cuida da conversão do código fonte para algum formato especificado em XML, tal como JavaML ou CppML. A decisão pelo uso de um formato ou outro ficará a critério da forma como a instância do framework for implementada.

O segundo componente lida com o armazenamento ou registro dos documentos XML gerados ou não pelo primeiro componente. A implementação

desse componente por uma instância do framework poderá ser feita utilizando um banco de dados nativo XML ou uma ferramenta que carregue em memória os dados XML que estejam armazenados em algum diretório do disco rígido.

O terceiro componente trata do carregamento das consultas chamados de "itens de consulta". Este carregamento é feito através da leitura do arquivo de categorias onde ficam registradas as consultas XQuery das três categorias descritas anteriormente.

O quarto e último componente trata da execução das consultas propriamente dita, fornecendo tanto os resultados das consultas como o tempo gasto em seu processamento.

A figura 10 mostra o diagrama de atividades do framework com relação aos seus quatro componentes.

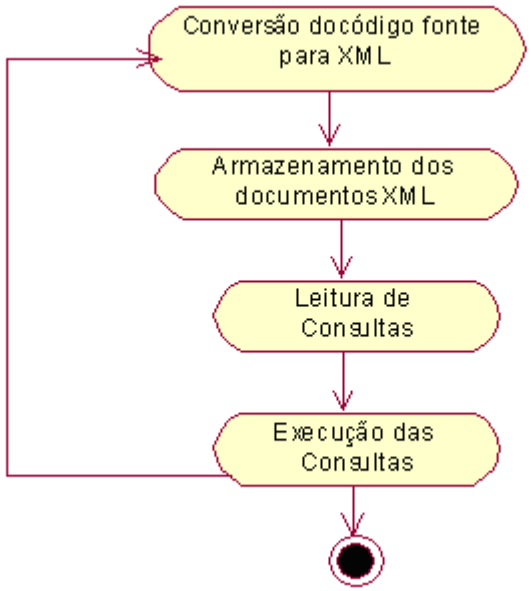

Figura 10 - Diagrama de atividades do framework XCARE

4.3.2 Pacotes de Implementação

Detalharemos nos próximos parágrafos os pacotes e as classes que compõem o framework bem como as suas funcionalidades e os padrões de projetos utilizados na sua implementação.

XCareConversor

XCareConversor é um pacote que contém classes que realizam a conversão ou transformação de arquivos de uma determinada linguagem de programação para XML. Este pacote contém algumas classes e interfaces utilizando como metodologia de construção do pacote alguns padrões de projeto de GoF ("The Gang of Four") [10].

Para este pacote o padrão usado foi o "Factory Method". Este padrão foi usado para que o pacote possa ser flexível quanto a inclusão de classes que implementem novos conversores.

Uma das principais classes desse pacote é a interface *Conversor*. Esta é responsável pela primeira etapa da análise de código e tem por objetivo converter um arquivo de um formato X para um formato Y, por exemplo, de Java para JavaML[1], de Java para MultiJava[40], etc.

Na figura 11 observamos o padrão Factory Method que define uma interface para criar objetos de forma a deixar que as subclasses decidam qual classe instanciar. Este padrão de projeto permite que as subclasses façam a instanciação.

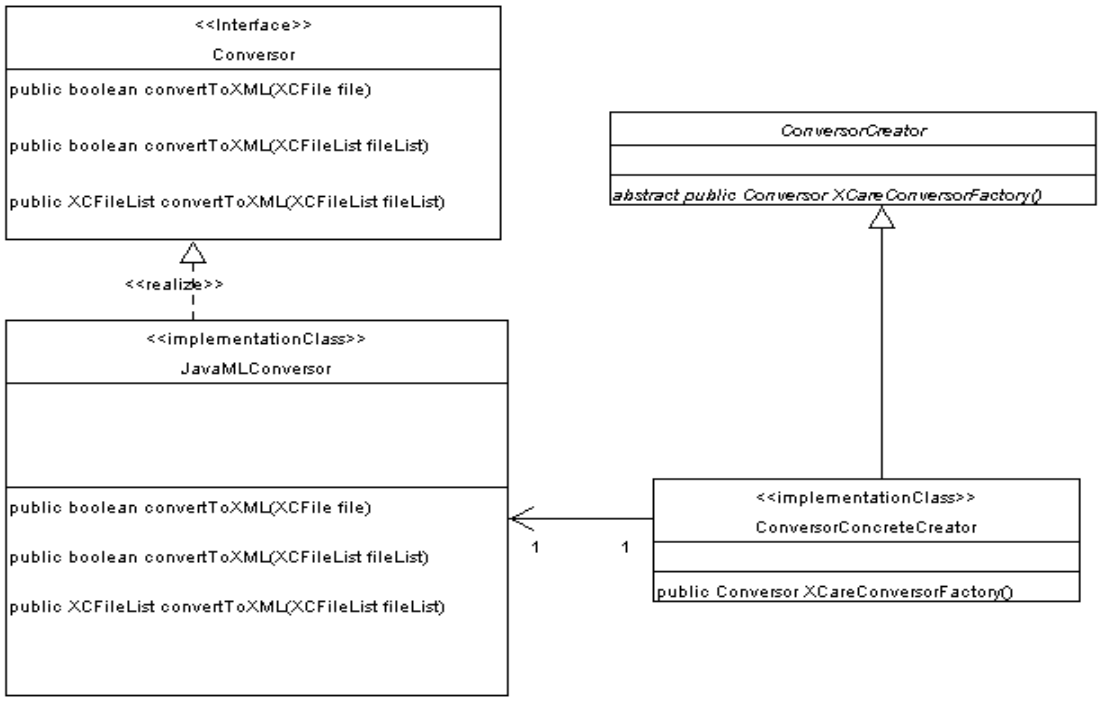

Figura 11: Pattern Factory Method do XCareConversor

A idéia é que ao invés de um cliente que precise de um objeto tenha que especificar a classe concreta que ele instancia, o cliente deve chamar um método abstrato (Factory Method) especificado em alguma classe abstrata ou interface (ConversorCreator), e a subclasse concreta (ConversorConcreteCreator) vai decidir que tipo exato de objeto criar e retornar.

Além disso, mudar a subclasse concreta que cria o objeto permite mudar a classe do objeto criado sem que o cliente saiba, chegando ao nosso objetivo de estender a funcionalidade através da construção de subclasses sem afetar os clientes.

A interface Conversor do framework possibilitará o reuso nas atividades de conversão através de classes que implementem os seus métodos.

Alguns exemplos de classes que possam implementar algum tipo de conversor são: ConversorJavaToJavaML, ConversorJavaToMultiJava, ConversorCppToCppML, etc.

Por exemplo, para instanciar o framework para o padrão JavaML de Badros é necessário criar uma classe chamada JavaMLConversor que terá a responsabilidade de conversão de arquivos .Java para documentos XML definidos de acordo com esse padrão.

Caso algum usuário queira utilizar o framework com outro conversor, este deverá apenas criar as classes que implementem as interfaces Conversor e ConversorCreator, assim como na figura 12, onde está sendo criado um novo conversor chamado MultiJavaConversor.

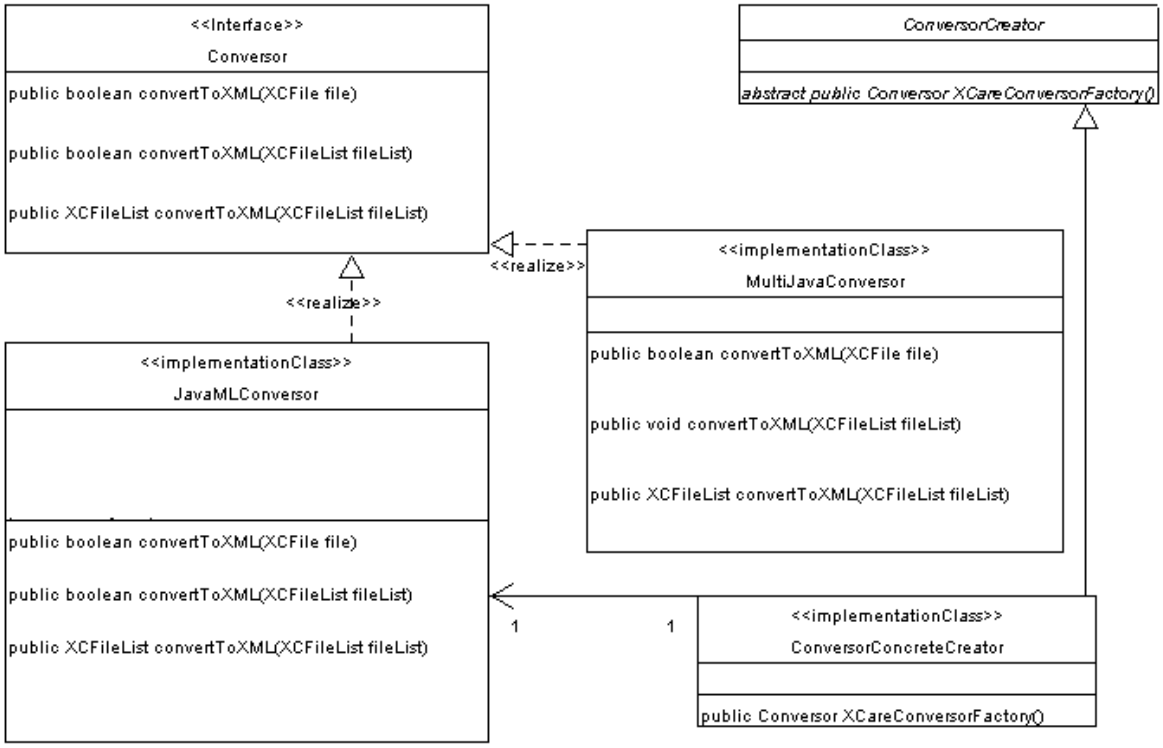

Figura 12: Exemplo de Extensão do framework XCare.

A decisão de usar uma classe JavaMLConversor ou MultiJavaConversor ficará a critério do método ConversorFactory da classe ConversorConcreteCreator. Neste método, uma condição será incluída para que possa ser escolhida a classe que se deseje trabalhar. Podemos observar um exemplo de implementação desse método no código seguinte:

```
public class XCareConversorConcreteCreator extends XCareConversorCreator {
   ...
   public XCareConversor XCareConversorFactory() {
      if (cond) {return new XCareJavaMLConversor();}
      else{return new XCareMultiJavaConversor();  }
   }
   ...}
```

O desejado é que os usuários do framework utilizem a classe ConversorFacade, que na figura 13 possui apenas os métodos para a conversão de arquivos do pacote Conversor.

Figura 13: ConversorFacade com os métodos para conversão.

Observamos três métodos da ConversorFacade que retornam o tipo boolean e outros dois métodos de conversão que retornam os tipos File e um vetor de Files. Estes últimos métodos serão utilizados em conjunto com as classes de persistência no pacote xCareStorage e servirão para retornar os arquivos gerados em XML para que estes possam ser armazenados.

xCareFiles

Este pacote é responsável pela criação de objetos que representem os arquivos onde o pacote de conversão irá tratar. O código fonte é um arquivo que será representado em forma de objeto pelo pacote xCareFiles e será utilizado pelo pacote de conversão.

Este conjunto de classes permite a manipulação de coleções de objetos para que seja possível a conversão de um único arquivo ou uma lista de arquivos. Para isso utilizou-se o padrão de projeto "*Iterator*".

O conversor recebe uma lista de XCareFiles e com isso há uma possibilidade de navegação na lista através de uma instância de "Iterator". A finalidade desta passagem da lista para o pacote de conversão é justamente com o intuito de ampliar a conversão de um único arquivo para uma coletânea de arquivos ou até mesmo um diretório de arquivos fonte.

Podemos observar na figura 14 as classes que implementam o Iterator e que possuem métodos para adição de novos elementos, remoção, navegação da lista e recuperação do elemento corrente da lista.

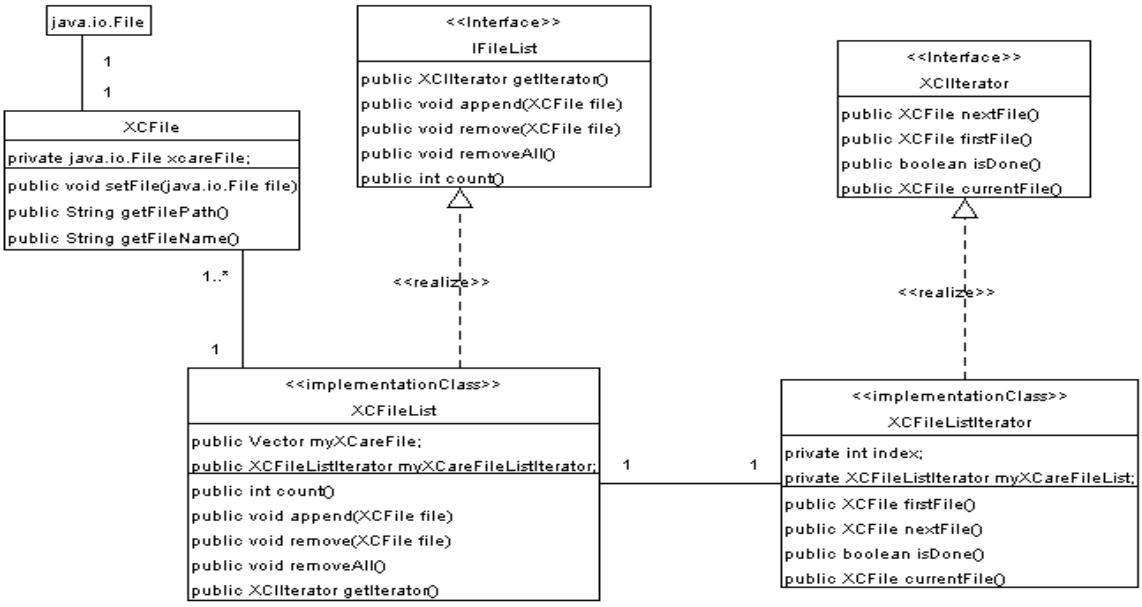

Figura 14: Pacote xCareFiles com suas classes.

xCareStorage

Este é o pacote responsável pela persistência de dados, ou seja, o registro dos dados em bancos XML nativos ou em ferramentas que armazenem XML em memória para o posterior tratamento.

O pacote utiliza a mesma idéia do padrão de método *factory*, onde este método retorna um objeto da implementação da interface DBManager. Isto pode ser visto na figura 15.

Desta forma, é possível que vários bancos nativos XML possam ser utilizados para os fins de persistência dos dados convertidos. Entretanto, esse pacote possui uma limitação por não armazenar outras informações importantes, tal como, o resultado de consultas realizadas na base de documentos. Essa limitação faz com que as instâncias do framework tenham que implementar a persistência dessas informações de uma outra forma.

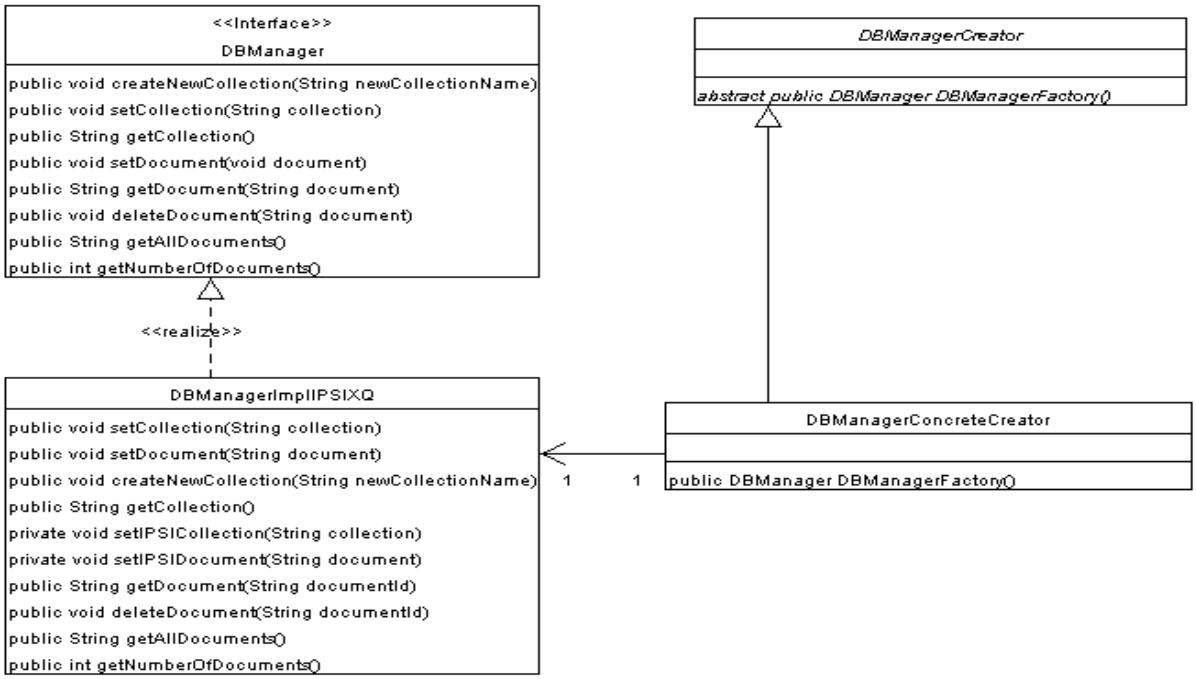

Figura 15: Classes do pacote xCareStorage.

A implementação de alguns métodos da interface DBManager permite a criação de coleções de dados, a adição e deleção de documentos a coleções, a obtenção de um ou mais documentos, a recuperação do número de documentos registrados na coleção corrente e a obtenção do nome da coleção corrente.

xCareCategories

Este pacote é responsável por recuperar os itens de consulta das categorias e adicionar novos itens para as categorias. Estes itens de consulta são obtidos através de um arquivo XML de categorias, onde são registradas as informações sobre os itens de consulta. Os dados registrados no arquivo de categorias possui informações sobre o nome da consulta, a descrição sobre o que essa consulta faz e o caminho onde está armazenado o arquivo XQuery que realiza a consulta.

A organização deste pacote também é feita através do padrão de projetos de método fábrica. Como principal elemento do padrão destaca-se a interface Category que define apenas três métodos: o primeiro é destinado a recuperação de um XQueryItem (que será visto no pacote xQueryFiles), o

segundo a obtenção de um objeto da classe XQueryItemList (lista de XQueryItem) e o terceiro método adiciona itens no arquivo de categoria.

Observa-se na figura 16 que três classes implementam a interface Category: Critic, Metric e ReverseEngineering. Estas classes utilizam a classe xCareCategoryReaderWriter.

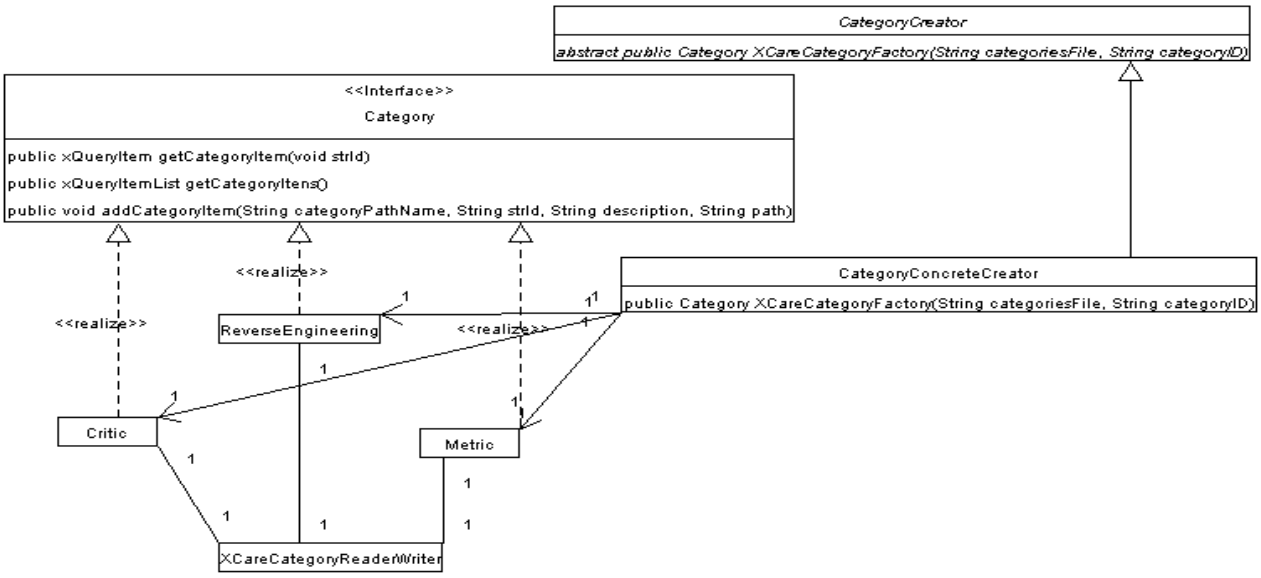

Figura 16: Classes do pacote xCareCategories.

Esta classe, que é compartilhada com as implementações da interface Category, possui métodos para a leitura e gravação do arquivo XML de categorias como ilustrado na figura 17.

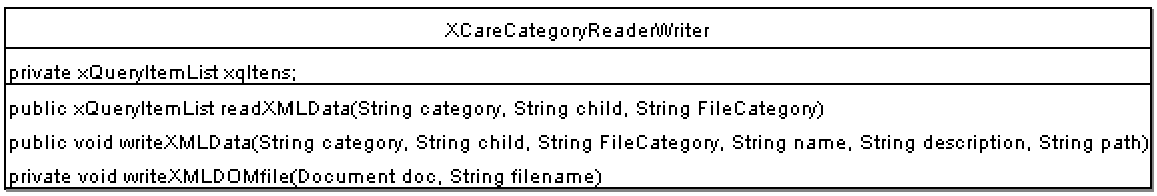

Figura 17: XCareCategoryReaderWriter e seus métodos.

xCareQueryFiles

Este pacote possui as funcionalidades para o tratamento das informações sobre os arquivos de categorias (xQueryItemList – lista de xQueryItem) que foram recuperados através da classe XCareCategoryReaderWriter. Este foi

modelado com o padrão de projeto Iterator, como podemos observar pela figura 18.

A classe XqueryItem modela um objeto que pode passar informações sobre o nome do item lido no arquivo de categorias, a descrição e o caminho que estão também armazenados neste arquivo.

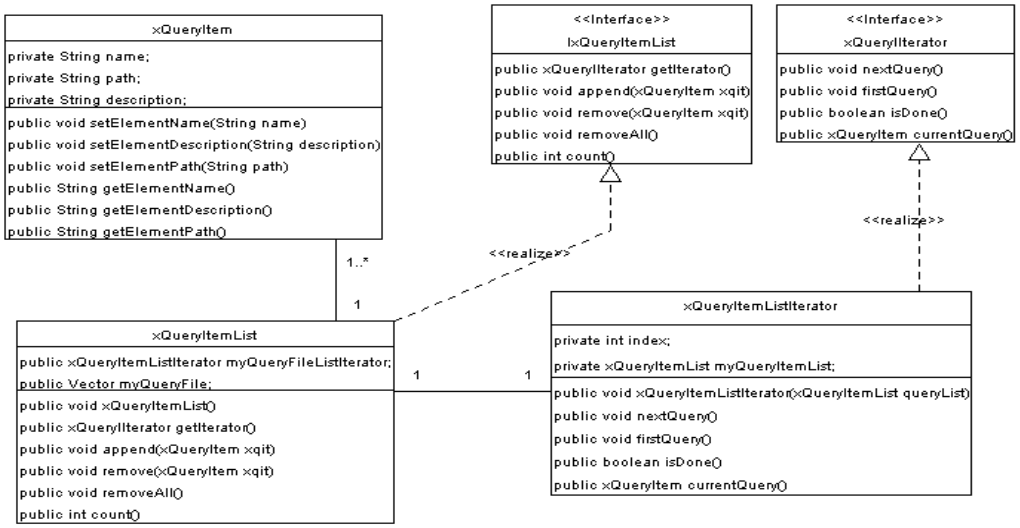

Figura 18: Pacote xCareQueryFiles e suas classes.

xCareQuery

A principal funcionalidade deste pacote é a execução e obtenção dos resultados dos itens XQuery. Este pacote utiliza o padrão de projeto método fábrica. Isto permite, como comentado em outros pacotes, uma maior flexibilidade para a inclusão de novas implementações de XQuery.

A interface XQuery deste pacote possui cinco métodos. Dois dos métodos são responsáveis pela conexão e a inicialização através da configuração de propriedades. Outros dois métodos retornam o resultado e o tempo gasto para a execução por meio de Strings. O último método realiza a execução da consulta.

Podemos observar na figura 19 a implementação da interface XQuery pela classe IPSIXQXquery. Esta classe utiliza como máquina de execução de consultas a ferramenta IPSI-XQ[23]. Esta ferramenta poderia ser trocada por outra ferramenta Xquery como, por exemplo, QUIP[53], da Software A.G.

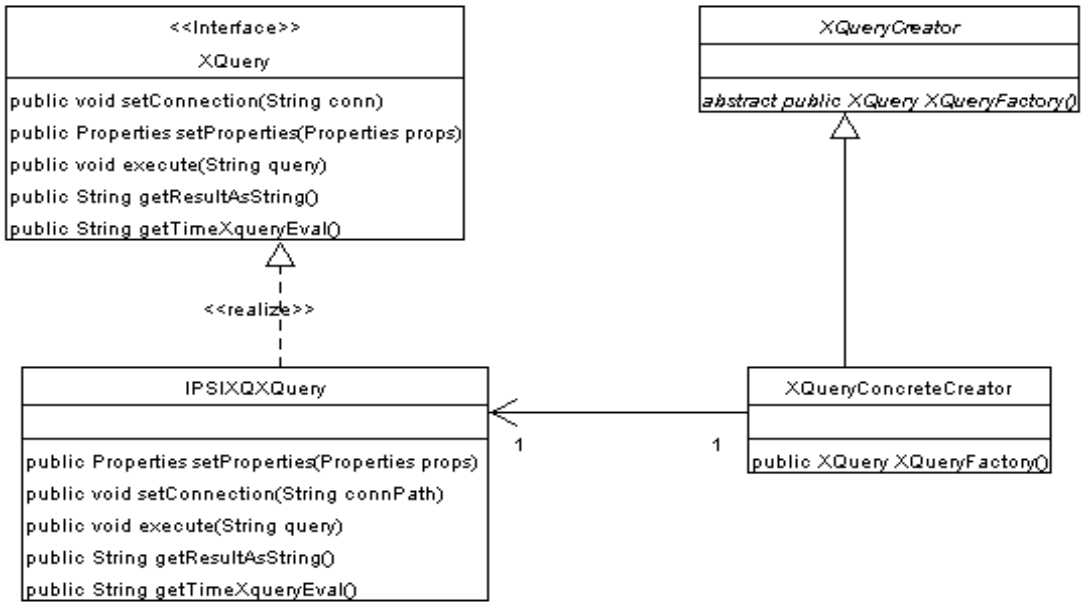

Figura19: Pacote xCareQuery e suas classes.

xCareManager

Este pacote é responsável pela fachada dos outros pacotes sendo este uma porta de entrada para a execução das funcionalidades. Pela figura 20 podemos observar as classes fachada que fornecem os serviços do framework XCARE.

As classes que pertencem a este pacote fornecem métodos que podem ser utilizados por aplicações externas para a criação de ambientes de análise de código.

Na seção 4.5 será explicado mais detalhadamente o processo de criação de ferramentas usando o conjunto de pacotes visto até aqui. Nas próximas sub-seções os serviços e os papéis que fazem parte do framework.

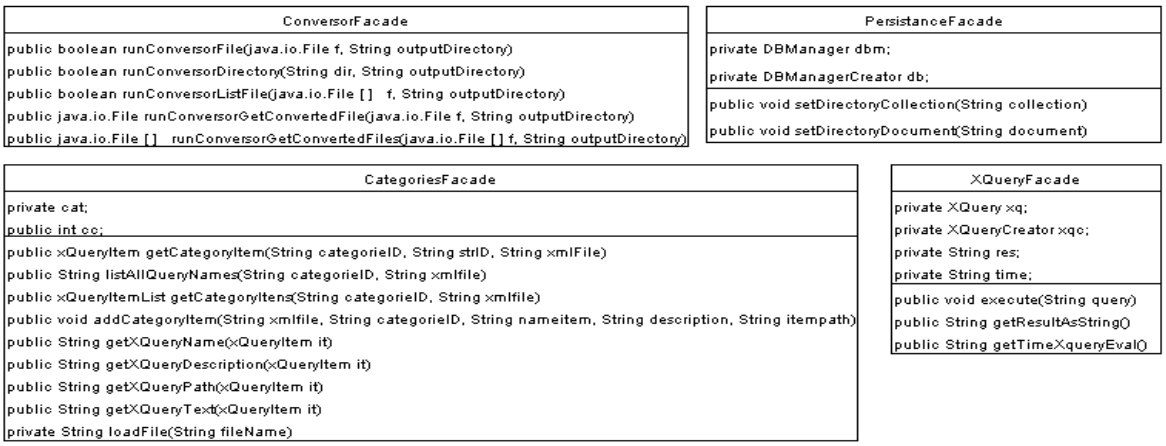

Figura 20: As classes fachadas.

4.4 Serviços e Papéis

Esta seção descreve os serviços e papéis necessários para o funcionamento, a criação e a manutenção de ambientes de análise de código e engenharia reversa utilizando o framework XCARE.

4.4.1 Serviços

Alguns serviços como conversão, armazenamento e execução de consultas que pertencem aos pacotes de xCareConversor, xCareStorage, xCareXQuery, respectivamente, devem ter as suas interfaces implementadas com as escolhas do conversor, banco para a persistência de dados, e executor de XQuery para que os serviços funcionem.

A classe ConversorFacade é responsável por uma lista de serviços para a conversão de código fonte para XML. Esta classe poderá ser estendida com a inclusão de novos métodos para que outras aplicações possam utilizá-la. Os principais serviços desta classe são apresentados na tabela 2.

A classe PersistanceFacade garante a persistência através do armazenamento dos documentos para que estes possam ser consultados posteriormente.

Classe Fachada: xCareManager.ConversorFacade	
Serviço	Método
Conversão de um código fonte para um formato XML. O retorno é do tipo boolean para a identificação se a conversão foi executada com sucesso.	runConversorFile
Conversão de um diretório de arquivos de código fonte para documentos XML. O retorno é do tipo boolean para a identificação se a conversão foi executada com sucesso.	runConversorDirectory
Conversão de uma lista de arquivos de código fonte para documentos XML. O retorno é do tipo boolean para a identificação se a conversão foi executada com sucesso.	runConversorListFile
Conversão de um código fonte para um formato XML. O tipo retornado é o documento XML gerado.	runConversorGetConvertedFile
Conversão de uma lista de arquivos de código fonte para um formato XML. O retorno é uma lista de documentos XML gerados.	runConversorGetConvertedFiles

Tabela 2 – Serviços disponíveis para conversão pela classe ConversorFacade

Esta classe possui apenas dois métodos que fornecem o endereço físico de onde se encontram o diretório de dados ou onde se encontra o caminho de diretório de um arquivo, como podemos observar pela tabela 3.

Dependendo de como for implementado esse serviço, ele irá armazenar em banco de dados nativo ou em memória.

Classe Fachada: xCareManager. CategoriesFacade	
Serviço	Método
Recupera do arquivo de categorias o item de consulta selecionado(categorieID que é uma das categorias:critic, metric, reverseengineering. strId é o nome da consulta, o elemento <name> do arquivo de categorias) e retorna um objeto do tipo xQueryItem.	getCategoryItem
Liste todos os nomes das consultas que estão armazenadas no arquivo de categorias.	listAllQueryNames
Recupera uma lista de xQueryItem dependendo do categorieId entrado.	getCategoryItens
Adiciona um item numa determinada categoria.	addCategoryItem
Recupera o atributo Name de um xQueryItem.	getXQueryName
Recupera o atributo Description de um xQueryItem.	getXQueryDescription
Recupera o atributo Path de um xQueryItem.	getXQueryPath
Recupera o texto xquery de um xQueryItem.	getXQueryText

Tabela 4 - Serviços disponíveis para a gerência das categorias de consulta.

Classe Fachada: xCareManager.PersistanceFacade	
Serviço	Método
Armazena os arquivos XML de um diretório em uma coleção corrente para que esta possa ser manipulada posteriormente.	setDirectoryCollection
Armazena um documento XML em documento corrente para que este possa ser manipulado posteriormente.	setDirectoryDocument

Tabela 3 - Serviços disponíveis para armazenamento de dados

Os serviços da classe CategoriesFacade permitem a manipulação do arquivo de categorias, adicionando ou recuperando os itens de consulta. Esta classe pode ser ampliada permitindo uma manipulação maior dos itens como, por exemplo, a alteração de partes de um item de consulta ou a exclusão de itens. A classe XQueryFacade possui serviços de execução e recuperação do resultado da execução. Entretanto, existe a ordem em que os métodos possam ser invocados sem que haja algum erro de execução Java. Por exemplo, não será possível recuperar o resultado de uma consulta se antes o método execute não tiver sido chamado.

Esta classe também possui enormes possibilidades de ampliação de seus serviços. Por exemplo, poderia ser criado um método que recebesse uma lista de consultas a serem executadas tal como um escalonador paralelo.

A tabela 5 nos mostra os serviços disponíveis para uso da classe XQueryFacade.

Classe Fachada: xCareManager.XQueryFacade	
Serviço	Método
Executa a consulta xquery.	Execute
Recupera o resultado da consulta.	getResult
Recupera o tempo de execução da consulta.	getTimeXQueryEval

Tabela 5 - Serviços disponíveis para execução das consultas.

4.4.2 Papéis

Podemos vislumbrar pelo menos três papéis no uso e na pesquisa do framework, como mostra a figura 21.

- O pesquisador gerador de novas extensões e ferramentas para análise de código.
- O programador de novas categorias e consultas de análise de código.
- O usuário da aplicação de análise de código que realiza consultas examina suas saídas e seu tempo de resposta.

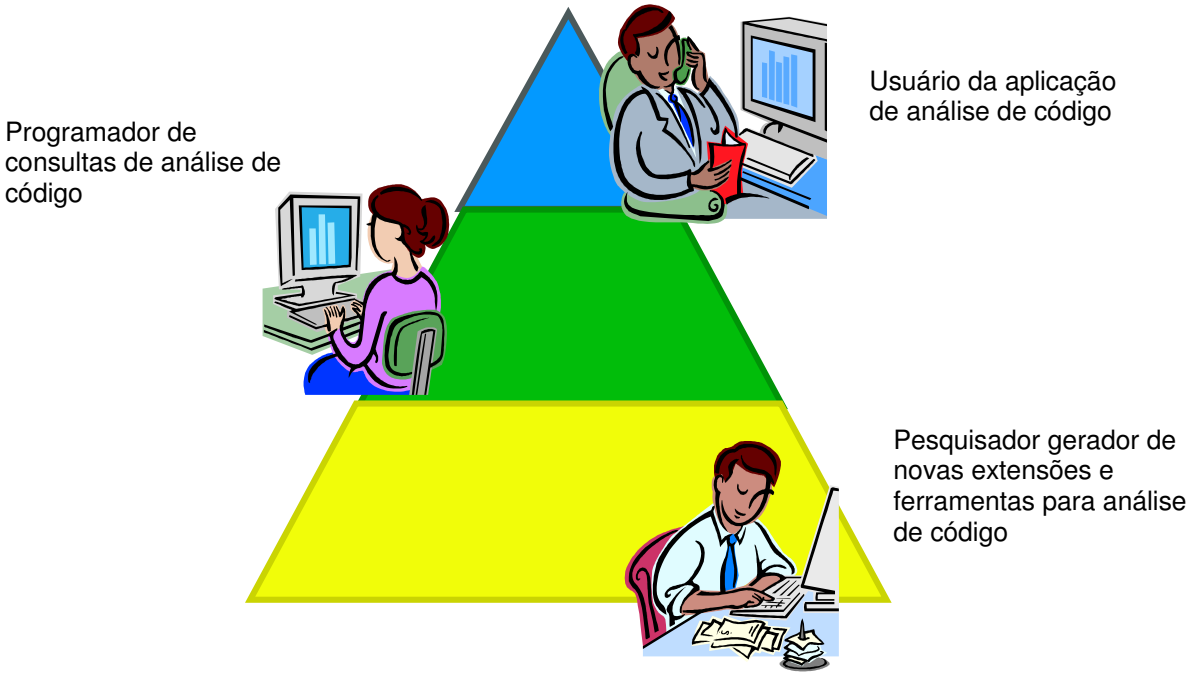

Figura 21: Pirâmide de utilização do framework.

O pesquisador gerador de novas extensões compõe a base da pirâmide que dá o sustentáculo para os demais papéis. Este usuário determina para quais linguagens o ambiente de análise de código será implementado.

Uma vez definidas as linguagens, este acopla os conversores, ou o conversor, caso este ambiente seja para apenas uma linguagem específica, determina também quais ferramentas de persistência, de execução de consulta xquery, e define e constrói as consultas xquery que serão utilizadas.

Um outro papel é o programador de novas categorias e consultas de análise de código. Esse é a essência da pirâmide, pois possui a sua importância na inserção de novas métricas, críticas e consultas de engenharia reversa.

Por fim, encontra-se o usuário que utiliza aplicações de engenharia reversa que tenham o framework como base. Este usuário poderá realizar consultas aos dados do código fonte das aplicações de seu interesse, com base nas diversas categorias cadastradas.

4.5 Criando Ferramentas de Análise de Código com XCARE

Esta é a atividade exercida pelo usuário gerador de novas extensões. Para a criação de ferramentas de análise de código este usuário deverá cumprir as seguintes tarefas:

- Escolher com qual linguagem o ambiente irá tratar.
- Escolher ou implementar uma ferramenta de conversão do código fonte da linguagem escolhida para um formato em XML, onde este possa ser consultado via XQuery.
- Implementar a interface Conversor utilizando a ferramenta de conversão.
- Implementar nova classe ConversorConcreteCreator onde esta retorne o objeto da classe que implementa Conversor.
- Implementar a interface DBManager para a persistência dos dados, escolhendo para isso ferramenta que armazene XML e possa aceitar

consultas XQuery; ou, caso as ferramentas de armazenamento e de consulta XQuery sejam distintas, que estas possam ser integradas.

- Implementar uma nova classe DBManagerConcreteCreator para que esta retorne o objeto da classe que implementa DBManager.

- Escolher a ferramenta de execução de consultas XQuery onde é pré-requisito que esta possa se comunicar com a implementação da interface DBManager.

- Implementar a interface XQuery utilizando a ferramenta escolhida para a consulta em XQuery.

- Implementar uma nova classe XQueryConcreteCreator para que esta retorne o objeto da classe que realiza a implementação da interface.

- Criar quais as consultas irão pertencer às categorias e destas construir as XQuerys correspondentes.

- Armazenar as informações destas consultas num único arquivo XML onde este tenha estrutura semelhante a da figura 9.

Com isso o gerador de extensões terá um conjunto de classes que perfazem uma API Java permitindo que seja geradas aplicações utilizando essa API, como é o exemplo do JCare mostrado a seguir.

4.6 JCARE: Uma Instância de XCARE para Java

JCare é uma API construída com base nas classes e serviços do framework XCARE. Esta API é específica para a análise de código fonte Java, pois utiliza o conversor JavaML de Badros que converte do código fonte para um formato XML, formato este definido pela ferramenta.

Para a lista de consultas usamos a linguagem para consulta XQuery especificada pela W3C, onde todas as consultas foram construídas com base no formato especificado pela representação JavaML.

Para o armazenamento optou-se pela utilização da ferramenta IPSI-XQ, que realiza o armazenamento em memória dos documentos ou coleções e posteriormente pode executar consultas com base nos arquivos XML armazenados.

A opção IPSI-XQ nas classes de armazenamento deveu-se principalmente pela facilidade de se usar um pacote gratuito, não comercial, e também pela maior atualização da implementação da especificação de XQuery da W3C (versão IPSI-XQ v.1.3.1 datada do dia 22 de Agosto de 2003).

A figura 22 nos mostra a divisão dos pacotes de serviços que a instância JCare utiliza para a realização da análise de código baseado nas tecnologias XML.

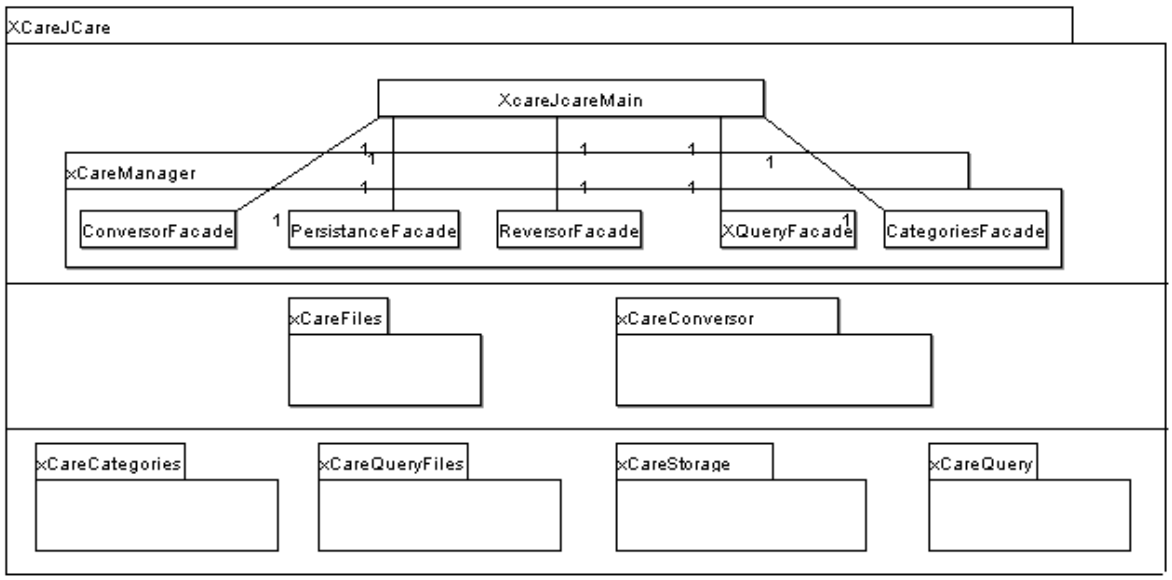

Figura 22: Pacotes que compõem o framework instanciados para Java (JCare) .

Finalmente, para fins de experimentação foram utilizados bancos nativos XML para testes de performance onde estes serviriam de comparativos com as ferramentas de armazenamento XML em memória.

4.7 Limitações do Framework

Como principais limitações do framework XCARE podemos citar:

- Para cada linguagem escolhida para análise de código, faz-se necessário a construção das funções base específicas da linguagem, ou

seja, a estrutura sintática da linguagem de programação deverá ser analisada para descobrirmos quais funcionalidades serão implementadas nas funções base.

- Não há um padrão XML único que represente todas as linguagens. Isto implica no desenvolvimento de novos conversores para linguagens de programação ainda sem representação em XML definida.

- As ferramentas escolhidas para consulta são dependentes do fabricante. Isto implica no risco de que o fabricante não acompanhe o amadurecimento da linguagem XQuery tal como especificada pela W3C.

- O tempo de conversão dos arquivos fontes para XML deve ser também analisado para o efetivo uso da tecnologia XML (não medimos os tempos de conversão dos arquivos fonte em XML).

- O tamanho do arquivo XML gerado também pode ser um problema para as análises do código, quando usamos uma ferramenta de consulta a XML em memória.

4.8 Conclusão

Neste capítulo descrevemos o framework XCARE, proposto nesta dissertação, que utiliza padrões e tecnologias baseadas em XML para facilitar o desenvolvimento de ferramentas de análise de código.

No próximo capítulo serão apresentados os experimentos realizados e seus resultados, e serão feitas considerações sobre o desempenho e a escalabilidade das tecnologias envolvidas na instância do framework XCARE.

CAPÍTULO 5 - Considerações de Desempenho das Ferramentas Desenvolvidas com o Framework

5.1 Introdução

A seguir apresentamos resultados dos experimentos realizados em algumas tecnologias que manipulam dados XML que foram utilizados numa instância do *framework* XCARE.

5.2 Objetivo dos Experimentos

Os experimentos reportados neste capítulo tiveram como objetivo avaliar a real possibilidade de se utilizar algumas das ferramentas de consulta a dados XML disponíveis atualmente, que implementem a linguagem XQuery, para a atividade de análise de código e engenharia reversa.

Para a avaliação de desempenho das ferramentas, utilizaram-se aplicações reais de tamanhos variados, cujo código fonte fora convertido para uma representação correspondente em XML. O código fonte foi então consultado utilizando ferramentas de banco de dados nativo XML e ferramentas que trabalham apenas em memória.

5.3 Metodologia

Os experimentos foram realizados através de uma aplicação Java que contabilizou o tempo gasto nas consultas feitas ao código utilizando as API´s das ferramentas para a execução das consultas. Os resultados obtidos foram comparados em relação a cada ferramenta e a cada tipo de consulta XQuery considerada. Para as ferramentas que armazenam os dados em memória não foi considerado o tempo de carregamento em memória dos arquivos XML, mas apenas a duração das consultas uma vez que arquivos já estavam em memória. Além da duração das consultas, também foram medidos: o número de arquivos consultados, o tamanho deste conjunto de arquivos e o tempo consumido na consulta desse conjunto de arquivos.

O equipamento de hardware utilizado para a realização dos experimentos foi um micro-computador Pentium 4, com processador de 2 GHz, 512 Mb de Memória DRAM e HD de 40 GB. O software utilizado para o desenvolvimento da aplicação Java foi o Eclipse SDK-2.1.2-win32, utilizando o Java Jdk 1.4, rodando em um Sistema Operacional Windows XP.

5.3.1 Escolha da ferramenta de conversão de dados XML

Para a escolha da ferramenta de conversão de dados XML, foi realizado um experimento utilizando-se conversores para cinco representações de código fonte Java em XML: JavaML de Badros[1], JavaML de Mamas[37], JavaSRCML[5], BeautyJ[2] e MultiJava[40].

A aplicação cujo código fonte foi convertido neste experimento foi o Java Development Kit - JDK versão 1.4.0_01. Esta aplicação possui 3883 arquivos fonte (“java”), totalizando de 39.659.815 bytes (39 Mbytes).

Pelas tabelas a seguir podemos observar o número de arquivos e seus respectivos tamanhos, bem como as estratégias de conversão que cada conversor utiliza. Por exemplo, a ferramenta BeautyJ converte um diretório inteiro de arquivo fonte para um único arquivo XML. Já as ferramentas JavaML de Badros, MultiJava, JavaSRCML e JavaML de Mamas convertem cada arquivo fonte para um arquivo XML em separado.

	JavaML Badros	JavaML Mamas	JavaSRCML	BeautyJ	MultiJava
	.xml	.xml	.xml	.xml	.xml
Jdk 1.4	3877	3883	3809	151	3781

Tabela 6 - Quantidade de arquivos XML gerados.

Na tabela 6, podemos observar que o JavaML de Mamas é a ferramenta que consegue converter todos os arquivos solicitados. A segunda ferramenta, em número de arquivos convertidos com sucesso, foi a JavaML de Badros. Entretanto, não podemos descartar que a ferramenta BeautyJ converteu um número menor de arquivos em virtude da sua estratégia de conversão.

Outro aspecto importante a ser considerado foi o tamanho do XML gerado. Podemos observar na tabela 7 que o JavaML de Mamas é a ferramenta que apresenta um maior número de bytes convertidos em XML, 6 vezes maior do que o segundo maior valor, e 46 vezes maior do que o menor valor entre as ferramentas.

Podemos comprovar, também através da tabela 7, que o JavaML de Mamas é bastante prolixo na sua representação, onerando cada arquivo gerado em cerca de 15 vezes o seu tamanho, em relação a representação de Badros.

	JavaML Badros	JavaML Mamas	JavaSRCML	BeautyJ	MultiJava
	MBytes	MBytes	MBytes	MBytes	MBytes
Jdk 1.4	89,67	1391,76	40,45	29,88	217,82

Tabela 7 – Tamanho total em Bytes dos arquivos XML gerados

Comparando as demais ferramentas de conversão, podemos observar que as ferramentas BeautyJ, JavaSRCML e JavaML de Badros ficaram com menos de 100 Mbytes de arquivos convertidos no total.

No trabalho de Tommy Chu[57], onde este realizou uma avaliação de conversores Java para XML, foi observado que a ferramenta que mais se destacou na conversão pela eficiência, tamanho, confiabilidade e completeza foi JavaML de Badros[1].

Como a intenção da pesquisa não é investigar a melhor alternativa para representar a Java em XML, escolhemos JavaML de Badros, com base nos

experimentos iniciais realizados, sobre o tamanho do código convertido e na quantidade de respostas que teríamos da representação XML escolhida, convergindo também com a idéia do trabalho de Tommy Chu, como sendo a representação a ser utilizada para a conversão de Java para XML no contexto dos experimentos.

5.3.2 Aplicações Java escolhidas como amostra

Cinco aplicações Java foram escolhidas para serem utilizadas nos experimentos. São elas: JWAM 1.6, JhotDraw 5.2, Jdk 1.4.0_01 AWT, JDK 1.4.0_01 e Eclipse 2.02.

A escolha das aplicações foi baseada nas aplicações utilizadas com estudos de caso no trabalho de Dirk Beyer[9], onde este propõe uma ferramenta chamada CrocoPat para a manipulação e consulta de relações entre entidades de código através de cálculos de predicados.

JWAM[33] é um framework orientado a objeto baseado em Java que possui componentes pré-fabricados e extensíveis para o desenvolvimento rápido de sistemas. É uma ferramenta pequena com apenas 90 arquivos fonte totalizando 518 kbytes.

JHotDraw[31] é um framework Java para gráficos técnicos e estruturados. Foi desenvolvido incialmente como "um exercício de projeto" mas já é hoje bastante poderoso. Os autores originais de JHotDraw foram Erich Gamma e Thomas Eggenschwiler. Esta ferramenta possui 160 arquivos fonte, totalizando 484 kbytes.

Jdk 1.4.0_01[35] é a aplicação para o desenvolvimento Java da Sun Microsystems. O Jdk 1.4.0_01 possui um pacote chamado AWT, que é responsável pelos componentes gráficos de formulários, botões, caixas de texto, etc. Este pacote contem cerca de 345 arquivos, destes 384 são classes, perfazendo um total de 141.267 linhas de código e cerca de 5Mbytes de tamanho.

Eclipse 2.02[22] é uma IDE de desenvolvimento Java que é bastante conhecida pela comunidade acadêmica e comercial de desenvolvimento de sistemas em

Java. Esta aplicação é a maior de todas as cinco aplicações consideradas, totalizando 7920 arquivos e 56 Mbytes de tamanho.

Aplicação	**Classes**	**Linhas de Código LOC**
JhotDraw 5.2	168	17.819
JDK 1.4.0 AWT	384	141.267
JWAM 1.6	2.397	284.818
JDK1.4.0	5.312	1.179.576
Eclipse2.02	8.925	1.181.270

Tabela 8 - Número de classes e número linhas de código das aplicações Java.

Amostra	Aplicação	Número de Arquivos Fonte Java	Num Bytes	Arquivos Java – XML Gerados com Sucesso	Num Bytes
1	JWAM 1.6	90	518.997	89	746.248
2	JhotDraw 5.2	160	484.425	159	1.867.526
3	JDK 1.4.0_01 AWT	345	5.073.574	330	9.397.333
4	JDK1.4.0_01	3883	39.659.815	3745	82.127.519
5	Eclipse2.02	7920	56.402.895	5403	129.470.509

Tabela 9 – Comparativo do número de arquivos fonte com o número de arquivos XML gerados, com seus respectivos tamanhos em bytes.

Podemos observar nas tabelas 8 e 9 a quantidade de classes, arquivos, linhas de código, arquivos XML gerados, tamanho em bytes do código fonte e tamanho em bytes do código XML gerado para cada uma das cinco aplicações.

Amostra	**Aplicação**	**Arquivos Java Sem Erros**	**Num Bytes**	**Arquivos Java Não repetidos**	**Num Bytes**
1	JWAM 1.6	89	746.248	81	730.647
2	JhotDraw 5.2	159	1.867.526	159	1.867.526
3	JDK 1.4.0_01 AWT	330	9.397.333	329	9.396.432
4	JDK1.4.0_01	3745	82.127.519	3606	79.369.467
5	Eclipse2.02	5403	129.470.509	5079	121.862.851

Tabela 10 - Tabela contendo o número de arquivos Java (não repetidos em seu nome) utilizados.

A tabela 10 demonstra nas colunas de arquivos Java não repetidos, o número de arquivos utilizados nos experimentos, bem como seu respectivo tamanho. Os arquivos de cada uma das cinco aplicações foram agrupados em diretórios

em separado e denominados conjuntamente de amostra 1, 2, 3, 4 e 5, respectivamente.

5.3.3 Consultas utilizadas nos experimentos

As consultas que foram utilizadas nos experimentos levaram em conta o seu uso prático no contexto de operações típicas de análise de código. Muitas outras consultas poderiam ter sido utilizadas para a experimentação; entretanto, estas foram escolhidas pela facilidade de seu uso e de implementação em XQuery. Consultas com um grau maior de complexidade não foram testadas. As implementações detalhadas das consultas escolhidas encontram-se na seção de Anexos.

Seis consultas foram selecionadas, como segue:

Q1 - Retorna o nome de todos os atributos de classe que sejam do tipo “int”.

A consulta Q1 tem por objetivo recuperar, dos arquivos XML gerados, o nome de todos os atributos encontrados dentro das classes, onde estes atributos sejam do tipo “int” (Integer).

Esta consulta é bastante comum para o desenvolvedor, pois este algumas vezes necessita entrar numa classe e procurar atributos de um determinado tipo, ou até mesmo modificar estes atributos para um novo tipo, realizando para isso uma operação de “refactoring” [41].

Q2 – Retorne a métrica WMC da aplicação.

A consulta correspondente à métrica WMC (Weighted Methods per Class) simplesmente conta o número de métodos implementados por cada classe da aplicação. Esta métrica é útil para medir o potencial de reuso de uma determinada classe. Quanto maior for o número de métodos numa classe, maior será o impacto dessa classe nas suas eventuais sub-classes. Isto porque as sub-classes herdam todos os métodos definidos na sua classe pai. Classes que possuem um número grande de métodos podem ser mais específicas para uma aplicação do que outras, limitando assim a possibilidade de seu reuso.

A consulta que implementa esta métrica utiliza “count”, uma função primitiva da linguagem XQuery, para contar o número de filhos do tipo “method” de um determinado elemento da árvore XML do tipo “class” passado como parâmetro.

Q3 – Verifique a crítica de projeto “uninstantiable public”.

A função XQuery que implementa a crítica Uninstantiable public é utilizada para detectar a existência de classes públicas que não possuam um construtor. Classes identificadas por essa crítica não podem ser instanciadas externamente por outras classes, o que indica um potencial erro de projeto.

Q4 – Verifique a crítica de projeto “class with only attributes and no methods suggest inadequate behavior distribution”.

Esta consulta pode ser utilizada pelo desenvolvedor para identificar falhas de projeto na especificação de classes sem nenhum método definido.

Q5 – Verifique a crítica de projeto “public class with no public features don't present anything useful”

Outra consulta que evidencia erro de projeto é a criação de uma classe sem métodos públicos. Isto demonstra uma falha na implementação, pois sabemos que toda classe deveria precisar de pelo menos um método público.

Q6 - Retorne a hierarquia de classes da aplicação.

Esta consulta é bastante comum para o desenvolvedor, pois este algumas vezes necessita conhecer a hierarquia de classes da aplicação desenvolvida. Identificar as dependências de herança existentes entre as classes de um programa é um importante recurso para avaliar o impacto de mudanças, entre vários outros usos. Classes muito abaixo na hierarquia, isto é, que dependam excessivamente de outras classes, tendem a exigir um esforço maior para alteração e reutilização.

5.3.4 Ferramentas de consulta utilizadas

A escolha das ferramentas de consulta e armazenamento de dados XML utilizadas nos experimentos foi baseada no fato de precisarmos realizar

medições envolvendo tanto ferramentas de Banco de Dados Nativo XML como também ferramentas que utilizassem apenas a memória para o armazenamento dos dados.

Além disso, outra característica determinante para a escolha foi que as ferramentas deveriam possuir uma API Java, através da qual pudéssemos executar as consultas a partir de uma aplicação Java, contabilizando assim o tempo de execução destas em arquivos texto.

As ferramentas de consulta XML em memória escolhidas foram QUIP versão 2.2.1 [53] e IPSI-XQ versão 1.3.1 [23], já descritas no Capítulo 3, e as que utilizam um banco de dados XML nativo foram XStreamDB versão 3.0 [73] e XHive versão 4.1 [67], também descritas naquele capítulo.

Os experimentos foram executados através de quatro programas Java, que possuíam acesso as API´s Java das ferramentas escolhidas. Cada consulta foi executada repetidas vezes, sendo então contabilizado a média das execuções(19 vezes foi o número escolhido, poderia ter sido um número menor de vezes).

Na próxima seção apresentaremos os resultados dos experimentos realizados com as ferramentas escolhidas.

5.4 Resultados dos Experimentos

5.4.1 Ferramentas de Armazenamento em Memória

5.4.1.1 QUIP

As consultas foram executadas através de uma aplicação Java que invocava os serviços de processamento de consulta da API do QUIP. Esta aplicação registrava em um arquivo texto o tempo de resposta de cada consulta.

Amostra	System	Q1	Q2	Q3	Q4	Q5	Q6
1	JWAM 1.6	1848	38028	55	53	54	2488
2	JhotDraw 5.2	5234	133650	67	49	53	6773
3	JDK 1.4.0_01 AWT	*	*	*	*	*	*
4	JDK1.4.0_01	*	*	*	*	*	*
5	Eclipse2.02	*	*	*	*	*	*

Tabela 11 – Média de tempo de execução da ferramenta QUIP das 6 consultas nas 5 amostras (em milisegundos).

Como mostram os resultados da tabela 11, o QUIP não suportou a execução das amostras 3, 4 e 5, sendo estas finalizadas com uma mensagem de erro informando que a capacidade de memória da ferramenta (256MB de heap) havia sido excedida.

Também podemos observar que não necessariamente o tempo de execução das consultas varia conforme o tamanho da amostra. Por exemplo, para as consultas Q4 e Q5 o tempo de execução da amostra 1 foi muito semelhante ao da amostra 2. Já para as demais consultas houve uma proporção do tempo de execução com relação ao tamanho da amostra.

A figura 23 mostra que a proporção de aumento da amostra 1 para a amostra 2 é de 2.55 vezes com relação ao tamanho da amostra, sendo que para o tempo de execução das consultas Q1, Q2 e Q3 a proporção de aumento entre as amostras foi ainda maior.

Também na figura 23 notamos que para as consultas Q3, Q4 e Q5 a proporção de aumento de tempo de execução foi bastante inferior a proporcionalidade do tamanho.

Em uma análise primária, podemos afirmar que com o QUIP nem todas as consultas sofrem degradação de desempenho proporcional ao aumento de tamanho das amostras.

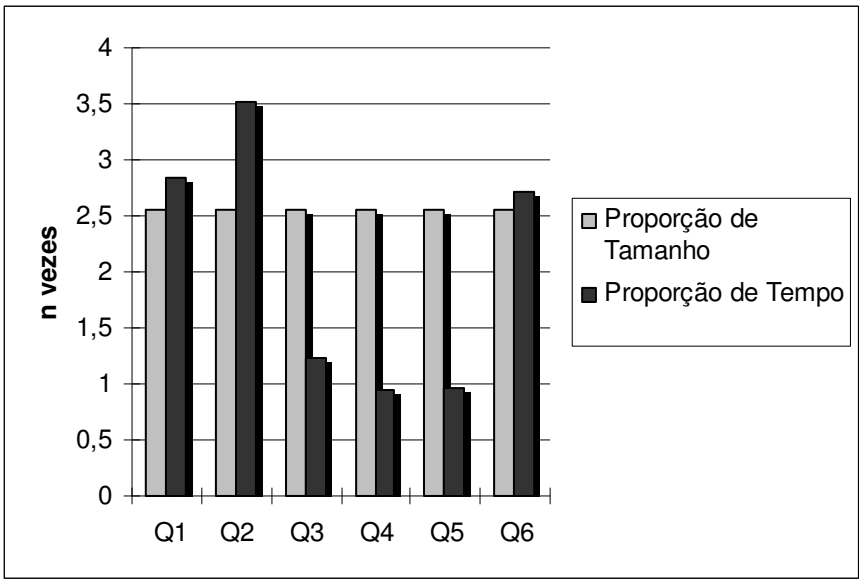

Figura 23: Proporção do tempo x tamanho das amostras 1 e 2.

5.4.1.2 IPSI-XQ

A ferramenta IPSI-XQ também produziu uma mensagem de erro durante a execução dos experimentos. Por razão similar à da ferramenta QUIP, a IPSI-XQ excedeu seu limite de memória na execução das amostras 4 e 5.

Pela tabela 12, podemos observar que, em relação à ferramenta anterior, IPSI-XQ executou com sucesso a amostra 3. Observa-se também que a consulta Q2 foi a consulta que mais tempo em média levou para completar a sua execução.

Amostra	System	Q1	Q2	Q3	Q4	Q5	Q6
1	JWAM 1.6	187	406	294	229	165	173
2	JhotDraw 5.2	366	882	382	400	225	242
3	JDK 1.4.0_01 AWT	4148	6966	2183	2155	1141	1177
4	JDK1.4.0_01	*	*	*	*	*	*
5	Eclipse2.02	*	*	*	*	*	*

Tabela 12 – Média de tempo de execução da ferramenta IPSI-XQ das 6 consultas nas 5 amostras (em milisegundos).

Execução Nr.	Q1	Q2	Q3	Q4	Q5	Q6
1	8656	10843	1640	2203	1140	1172
2	4062	6766	1656	2172	1141	1172
3	3984	6703	1500	2187	1141	1171
4	3875	6734	1485	2203	1140	1172
5	3922	6735	3578	2219	1156	1156
6	3921	6734	1484	2156	1157	1172
7	4000	6703	3563	2125	1140	1172
8	3907	6703	3781	2156	1141	1156
9	3906	6734	1500	2125	1156	1140
10	3875	6672	1484	2141	1141	1141
11	3875	6719	3594	2140	1125	1156
12	3844	6875	1516	2141	1141	1187
13	3844	6829	1516	2140	1141	1218
14	3860	6813	3593	2157	1140	1188
15	3875	6750	1500	2156	1125	1234
16	3859	6750	1562	2125	1140	1188
17	3844	6719	3547	2141	1141	1187
18	3859	6719	1485	2125	1125	1203
19	3844	6844	1484	2140	1141	1187
Média	4148	6966	2183	2155	1141	1177
Mediana	3875	6734	1516	2141	1141	1172

Tabela 13 - Tempo em milesegundos das 19 execuções na amostra 3 usando IPSI-XQ.

A execução das consultas na ferramenta IPSI-XQ foi realizada após o carregamento dos arquivos XML em memória. Isto implica que com o equipamento utilizado, nas amostras 4 e 5, que possuem, respectivamente, 79Mb e 121Mb de tamanho, o IPSI-XQ não foi capaz de carregar todo o tamanho das amostras em memória. Entretanto, em outros equipamentos com uma configuração superior à utilizada, provavelmente estes erros não aconteceriam.

Na tabela 13, mostramos todos os tempos de execução coletadas para as seis consultas na amostra 3. Através destas informações notamos algumas

particularidades da execução da ferramenta IPSI-XQ. Na primeira execução das consultas Q1 e Q2, há uma considerável divergência com relação a outras execuções. Vale salientar que as execuções foram realizadas em seqüência, sem qualquer interrupção da aplicação que controlava os experimentos.

Na figura 24, observamos o comportamento comentado no parágrafo anterior sobre a execução inicial de algumas consultas. Esse fenômeno é interessante de ser notado em virtude de que, quando se roda uma consulta esta é executada, muito provavelmente, uma única vez.

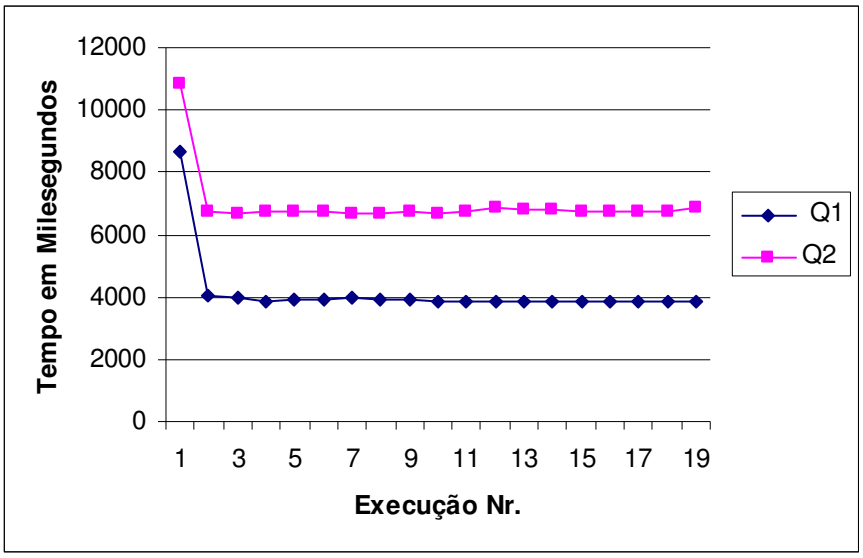

Figura 24: Tempo das execuções para a consulta Q1 e Q2 na amostra 3, usando IPSI-XQ.

Uma outra estratégia poderia ser usada para contornar o problema do carregamento dos dados em memória. As consultas poderiam ser executadas em partes das informações armazenadas nos arquivos XML, ou seja, em pacotes ou conjunto de classes da aplicação de maneira separada, contanto que estes pacotes ao se transformarem em arquivos XML, não extrapolassem o limite de memória da ferramenta.

Entretanto, é preciso salientar que, em algumas consultas, como é o caso da consulta Q6, sobre a hierarquia de classes, ou as que necessitem demonstrar todas as dependências entre classes, faz-se necessário a execução desta em toda a aplicação.

5.4.2 Bancos de Dados XML Nativos

5.4.2.1 XStreamDB

O XStreamDB teve um desempenho bastante superior com relação as ferramentas de carregamento em memória. Entretanto, de forma semelhante àquelas, esta ferramenta teve problemas de execução a partir da amostra 4.

Nas amostras 1, 2 e 3, todas as médias obtidas para as consultas ficaram abaixo de 1 segundo, conforme mostra a tabela 14. Podemos observar, também, que o tamanho das amostras não influenciou de forma significativa, nem é proporcional ao tempo de execução das consultas.

Amostra	System	Q1	Q2	Q3	Q4	Q5	Q6
1	JWAM 1.6	50	36	41	39	165	29
2	JhotDraw 5.2	55	30	39	34	101	27
3	JDK 1.4.0_01 AWT	49	35	47	38	86	27
4	JDK1.4.0_01	645	*	*	*	*	*
5	Eclipse2.02	*	*	*	*	*	*

Tabela 14 – Média de tempo de execução na ferramenta XStreamDB das 6 consultas nas 5 amostras (em milisegundos).

Os experimentos com o XStreamDB foram realizados através da criação de um Banco de dados XML e da inserção dos arquivos XML de cada amostra para esse banco.

O XStreamDB é um servidor de banco de dados que pode ser utilizado por um programa Java executando as suas operações através da API da ferramenta. No entanto, o tempo de rede não foi computado nos experimentos em virtude da ferramenta de banco de dados estar no mesmo computador que rodou o programa Java para a captura do tempo de resposta das consultas.

Podemos observar ainda, através da figura 25, que o tempo de execução para as amostras foi semelhante, com pequenas variações. O XStreamDB possui a opção da criação de índices para a otimização das consultas aos dado XML; entretanto, para os experimento não foi realizada nenhuma comparação de resultados entre consultas com índice e sem índice.

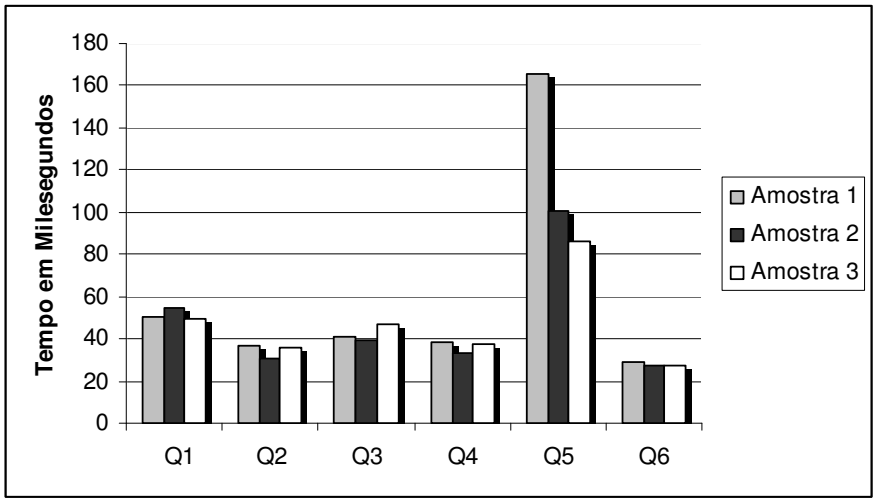

Figura 25: Tempo médio de execução das consultas para as amostras 1, 2 e 3.

Por fim, vale salientar que na amostra 4, as consultas Q2, Q3, Q4, Q5 e Q6 não conseguiram ser realizadas com sucesso. Podemos concluir então o tamanho também pode ser uma fator limitante, mesmo para ferramentas de banco de dados XML nativo.

5.4.2.2 X-HIVE

A ferramenta X-Hive foi a única dentre as quatro ferramentas que conseguiu rodar as consultas nos dados das 5 amostras. Essa ferramenta, assim como a XStreamDB, possui um objeto do tipo Iterator (resultset) que pode ser navegado. Neste objeto encontram-se os resultados das consultas.

Na tabela 15, podemos observar os resultados coletados para as 5 amostras. Notamos que ocorreu um fenômeno nas execuções das consultas Q1, Q2 e Q3. Para estas consultas, a ferramenta realizou uma espécie de “cache”, já que nas 19 execuções a primeira destas eram resolvidas em 16, 15 e 16 milesegundos e as demais execuções em zero milesegundos.

Amostra	System	Q1*	Q2*	Q3*	Q4	Q5	Q6
1	JWAM 1.6	16	15	16	260	262	165
2	JhotDraw 5.2	16	15	16	417	421	163
3	JDK 1.4.0_01 AWT	16	15	16	1168	1145	160
4	JDK1.4.0_01	16	15	16	10009	11271	185
5	Eclipse2.02	16	15	16	24584	23705	234

* Observação:
Nas consultas Q1, Q2 e Q3 a ferramenta X-Hive executou uma consulta com 16 milesegundos e as demais 18 em 0 milesegundos, dando uma falsa média.

Tabela 15 – Média de tempo de execução na ferramenta X-HIVE das 6 consultas nas amostras (em milisegundos).

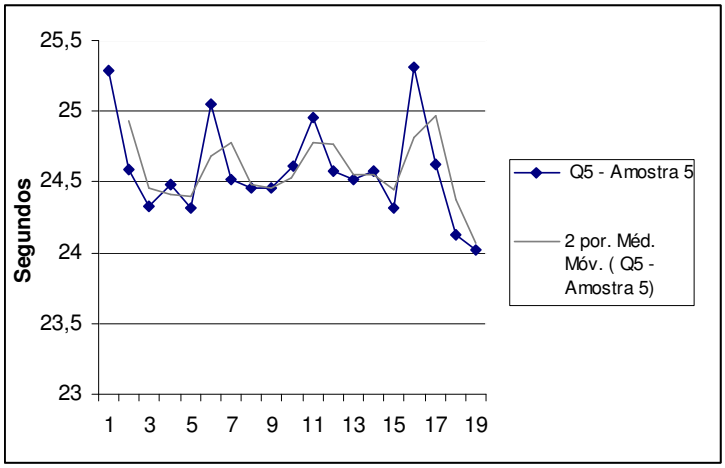

Figura 26: Gráfico das 19 execuções da consulta Q5 na amostra 5.

Outra informação interessante é com relação a variação encontrada nas 19 execuções das consultas, como, por exemplo, a apresentada pela consulta Q5 na amostra 5, mostrada na figura 26.

Já na consulta Q6 essa proporção é tendente a ser uma função polinomial, como mostra a figura 27.

Pela figura 28 fica visível que em determinadas consultas a proporção do tempo de execução com o tamanho da amostra é exponencial, por exemplo, nas consultas Q4 e Q5. Nessa figura vemos o gráfico dos tempos das consultas Q4, Q5 e a linha de tendência de crescimento exponencial.

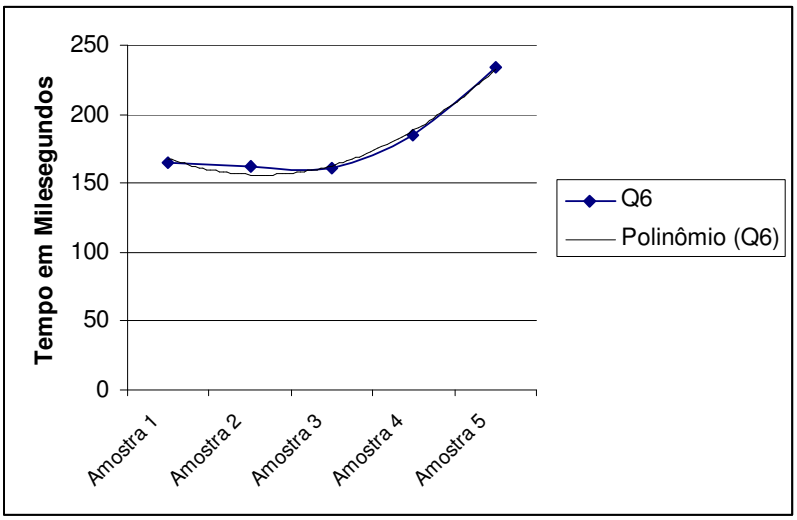

Figura 27: Gráfico do tempo de resposta da consulta Q6 nas 5 amostras.

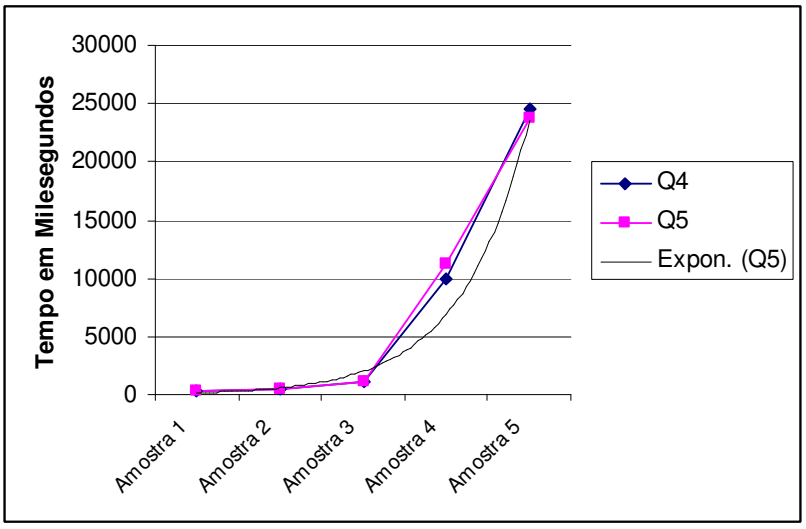

Figura 28: Gráfico do tempo de resposta das consultas Q4 e Q5 nas 5 amostras

5.4.3 Comparativos

5.4.3.1 Comparativo das Ferramentas de Armazenamento em Memória (QUIP e IPSI-XQ)

A figura 29 mostra os resultados comparativos para a amostra 1 entre o QUIP e o IPSI-XQ. Pela figura, podemos observar que nas consultas Q1, Q2 e Q6 o tempo de execução do QUIP é superior ao IPSI-XQ. Apesar desse resultado, em algumas consultas o QUIP apresentou um tempo de resposta superior ao IPSI-XQ, como no caso de Q3, Q4 e Q5. Estas últimas são consultas da

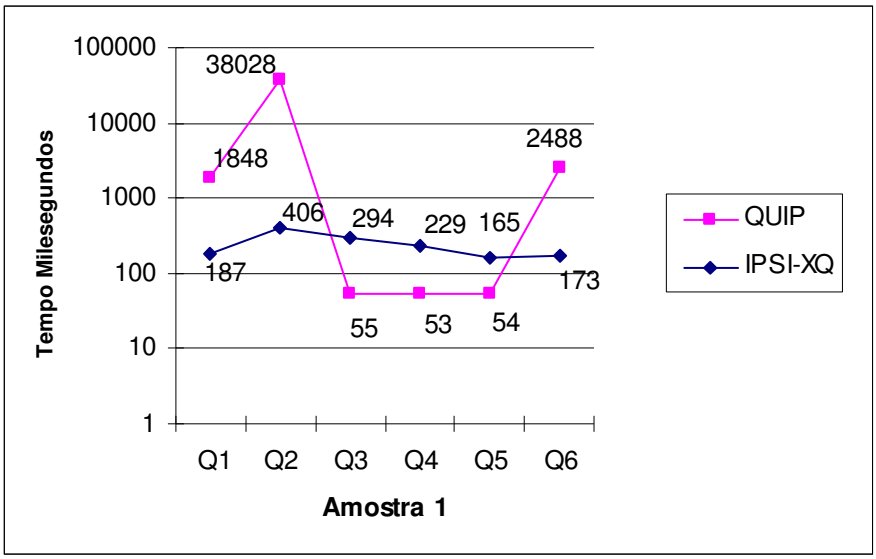

Figura 29: Gráfico comparativo do tempo de execução das consultas para a amostra 1 entre QUIP e IPSI-XQ.

categoria críticas de projeto (críticas de projeto são sugestões de melhorias ou erros apontados no projeto ou no código fonte de desenvolvimento de software.).

Observamos também que para a ferramenta IPSI-XQ todas as consultas realizadas na amostra 1 levaram menos de 0,5 segundos para a execução.

O QUIP variou o seu tempo de execução, com algumas consultas sendo bem mais rápidas do que outras. Notamos assim uma variação de tempo de execução das consultas muito maior na ferramenta QUIP.

A figura 30 apresenta os tempos de execução das mesmas consultas pelas duas ferramentas na amostra 2.

Os valores são muito parecidos com aqueles obtidos para a amostra 1. Novamente, nas consultas Q3, Q4 e Q5 a ferramenta QUIP oferece um melhor tempo de execução que o IPSI-XQ.

Entretanto, novamente, comprovamos uma variação de resultados maior no QUIP. Nota-se, ainda, que com a ferramenta IPSI-XQ todas as consultas levaram menos de 1 segundo para serem executadas.

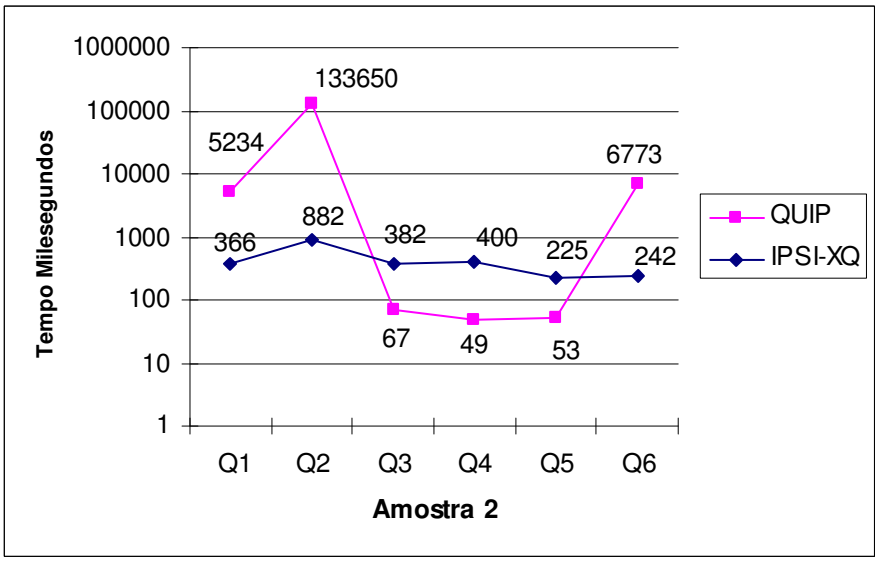

Figura 30: Gráfico comparativo do tempo de execução das consultas para a amostra 2 entre QUIP e IPSI-XQ.

Pela figura 31 observamos que à medida que o tamanho da amostra aumenta, o tempo de execução das consultas com o IPSI-XQ também acompanha essa tendência. Para amostras de tamanho menor do que 2Mb o tempo de execução de todas as consultas ficou abaixo de 1 segundo. Na terceira amostra, com cerca de 9 Mb, esse tempo variou entre 1 segundo no melhor caso (consulta Q6), e 7 segundos no pior caso (consulta Q2).

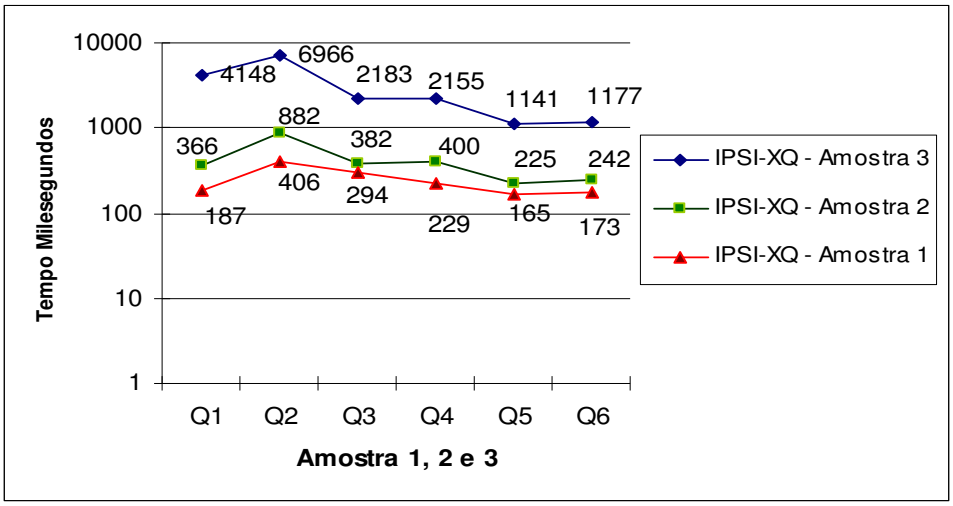

Figura 31: Gráfico comparativo do tempo de execução das consultas para a amostra 1, 2 e 3 do IPSI-XQ.

5.4.3.2 Comparativo dos Bancos de Dados XML Nativos (X-HIVE e XStreamDB)

A figura 32 mostra o comparativo entre os resultados obtidos para as seis consultas com as ferramentas X-HIVE e Blue-Stream, na amostra 1.

Pela figura observamos que nas consultas Q1, Q2 e Q3 o X-Hive se saiu um pouco melhor do que o Blue-Stream, com relação ao tempo de execução. Entretanto, nas consultas Q4, Q5 e Q6 o XStreamDB teve o tempo de execução consideravelmente menor do que o X-HIVE.

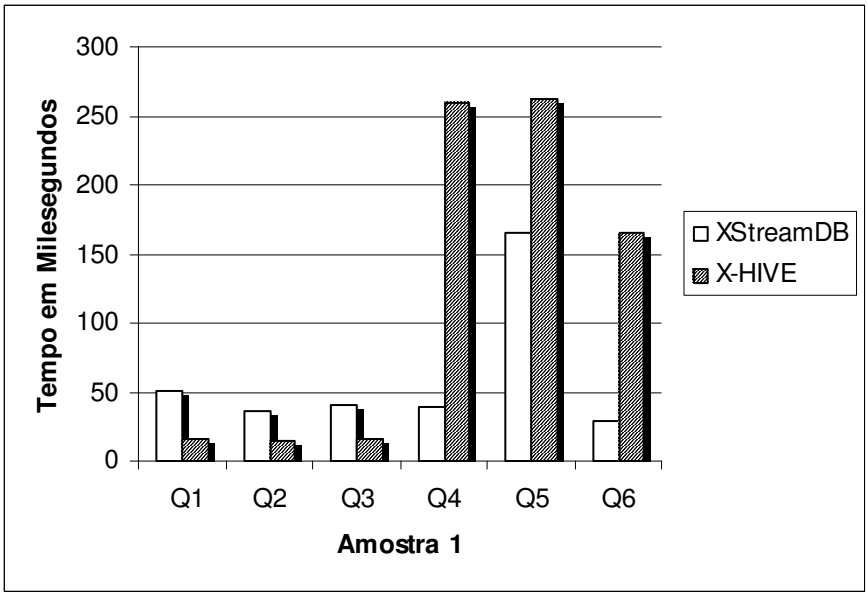

Figura 32: Gráfico comparativo do tempo de execução das consultas nas ferramentas XStreamDB e X-HIVE para amostra 1.

Analisaremos agora a tendência para as amostras 2 e 3. Intuitivamente, os valores devem ser semelhantes ao da amostra 1 com o tempo de execução das consultas sendo superior ao alcançado na primeira amostra.

Os gráficos das figuras 33 e 34 mostram a evolução dos tempos de execução das consultas para as amostras 2 e 3. Pela figura 33 observamos um gráfico muito semelhante ao apresentado nas figuras anteriores. A tendência se repetiu, modificando apenas o tempo máximo de execução de cada consulta.

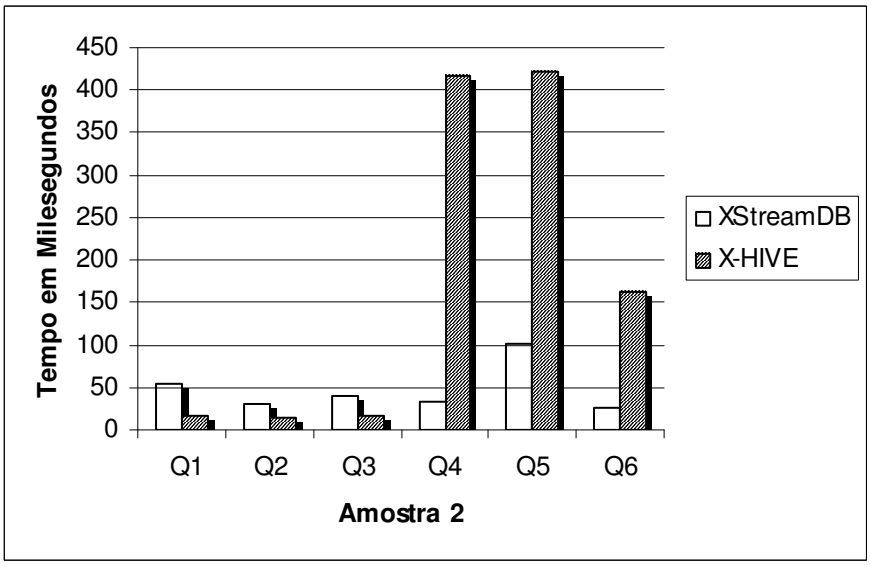

Figura 33: Gráfico comparativo do tempo de execução das consultas nas ferramentas XStreamDB e X-HIVE para amostra 2.

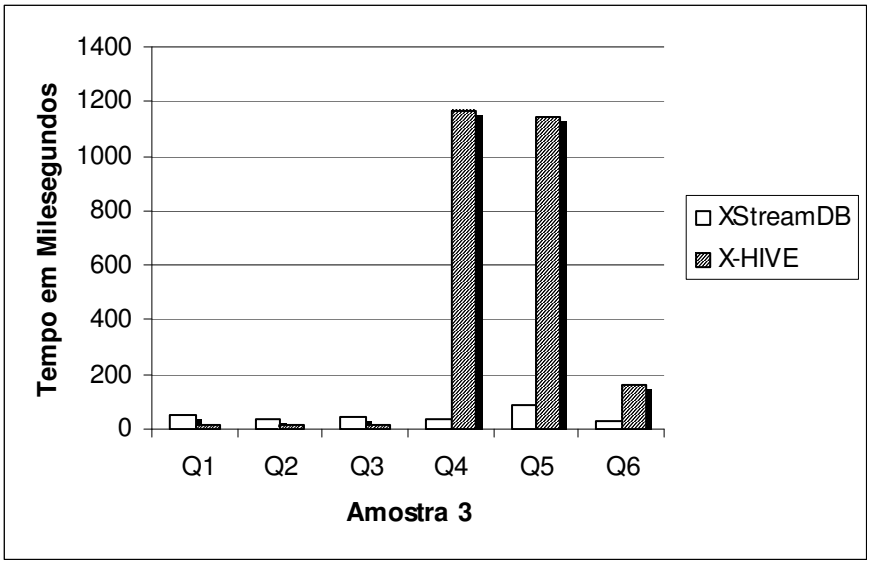

Figura 34: Gráfico comparativo do tempo da execução das consultas nas ferramentas XStreamDB e X-HIVE para amostra 3.

Na amostra 3 (figura 34), a tendência foi similar; entretanto, a proporção entre os tempos de resposta das consultas com a ferramenta X-HIVE foi notadamente desigual em comparação ao XStreamDB no tempo de execução das consultas Q4, Q5 e Q6. Para as consultas Q4, Q5 e Q6 a ferramenta X-HIVE teve o tempo de execução entre 1 segundo e 1,2 segundo. Entretanto, o XStreamDB manteve-se abaixo dos 0,2 segundos.

Nas 3 amostras a ferramenta XStreamDB foi a que teve os melhores tempos de execução, mantendo praticamente o tempo para todas as seis consultas de forma linear. Vale ressaltar que a ferramenta X-HIVE foi a única que conseguiu executar as amostras 4 e 5.

5.4.3.3 Comparativo entre os Bancos de Dados Nativos e as Ferramentas de Execução em Memória

Comparando as ferramentas de Banco de Dados nativos XML e as ferramentas de execução em memória, podemos observar algumas situações interessantes.

Em algumas consultas, as ferramentas de memória executam em um tempo menor do que as ferramentas de Banco de Dados, sobretudo nas menores amostras (amostras 1 e 2).

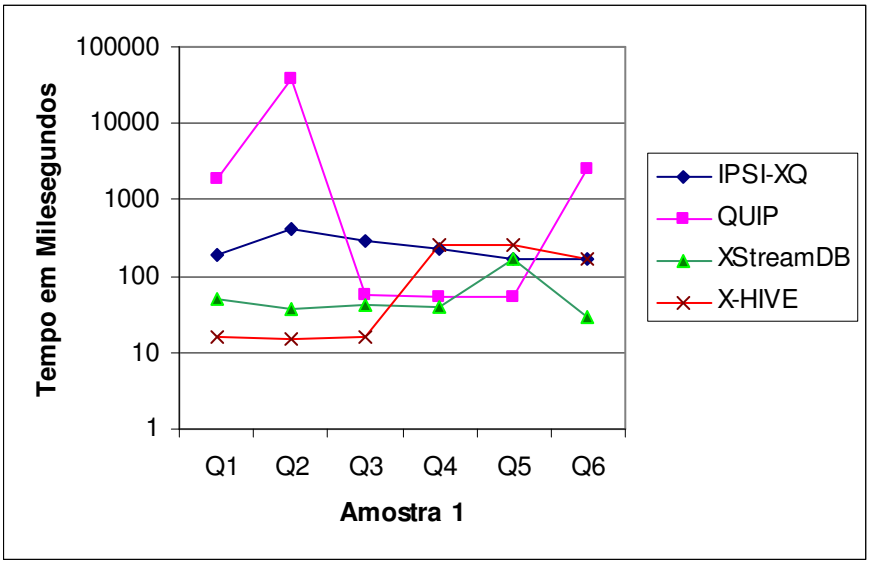

Figura 35: Gráfico comparativo do tempo de execução das consultas na amostra 1, pelas ferramentas IPSI-XQ, QUIP, XStreamDB e X-HIVE.

Podemos observar isto através das figuras 35, 36 e 37.

É importante frisar que, para as ferramentas de banco de dados nativo, o carregamento dos arquivos XML é realizado antes das execuções das consultas, assim como para as ferramentas de memória.

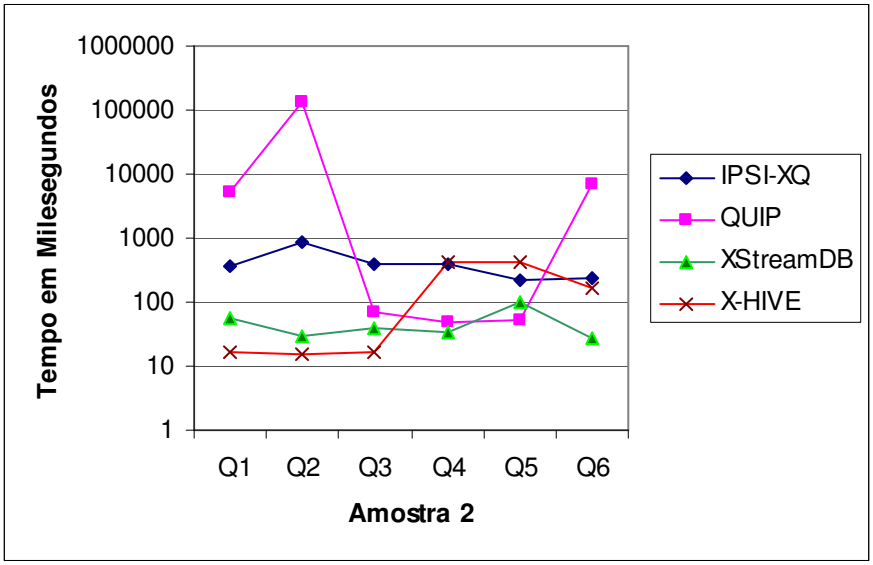

Figura 36: Gráfico comparativo do tempo de execução das consultas na amostra 2, pelas ferramentas IPSI-XQ, QUIP, XStreamDB e X-HIVE.

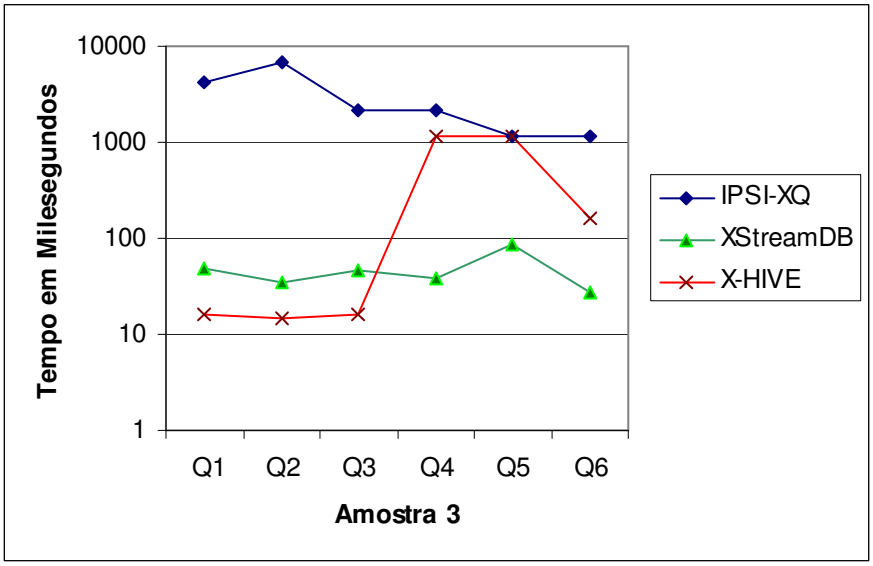

Figura 37: Gráfico comparativo do tempo de execução das consultas na amostra 3, pelas ferramentas IPSI-XQ, QUIP, XStreamDB e X-HIVE.

Nota-se pelos gráficos que as duas ferramentas de banco de dados e a IPSI-XQ, para as amostras 1 e 2, executam as consultas em um tempo abaixo de 1 segundo. Na amostra 3 este quadro é modificado, com apenas as ferramentas de banco, sobretudo a ferramenta XStreamDB, ficando com tempos de execução próximos de 1 segundo.

Pelas figuras 36 e 37 fica visível que à medida que aumenta o tamanho da amostra, o tempo de execução da ferramenta XStreamDB, nas consultas Q5 e Q6, é praticamente constante. Observamos ainda, na figura 38, que para a ferramenta X-HIVE, o tempo de execução da consulta Q5 aumenta consideravelmente para as amostras 4 e 5 em comparação com as outras amostras.

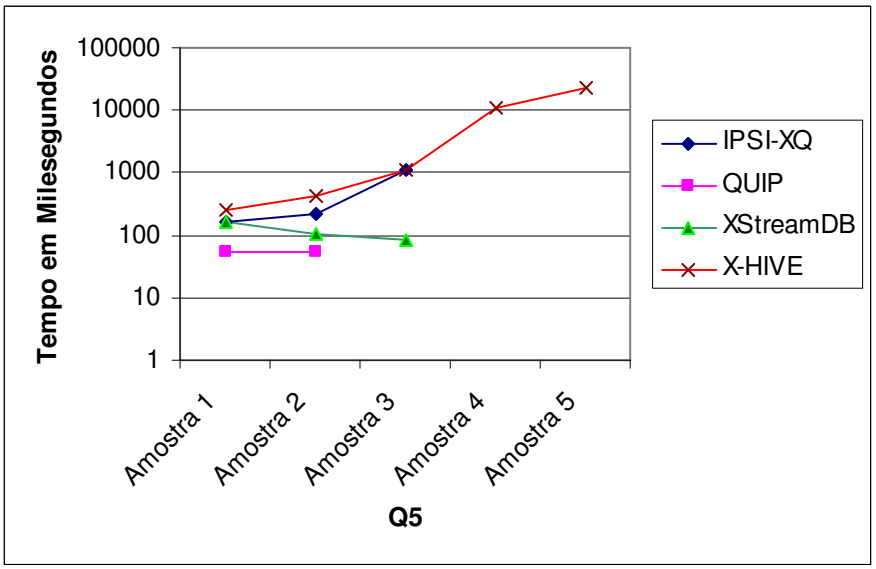

Figura 38: Gráfico comparativo do tempo de execução da consulta Q5 nas 5 amostras, pelas ferramentas IPSI-XQ, QUIP, XStreamDB e X-HIVE.

Na figura 39 fica claro que o tempo de execução das ferramentas de armazenamento em memória aumenta de forma proporcional ao aumento no tamanho da amostra.

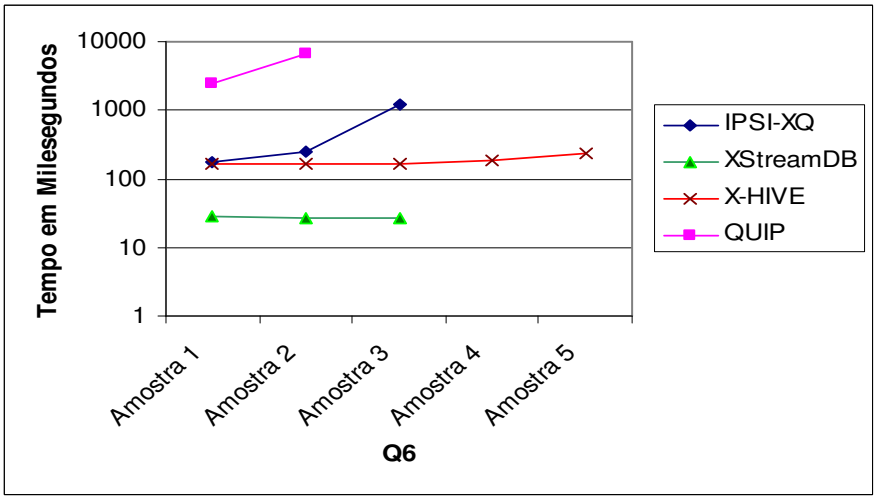

Figura 39: Gráfico comparativo do tempo de execução da consulta Q6 nas 5 amostras, pelas ferramentas IPSI-XQ, QUIP, XStreamDB e X-HIVE.

Já para as ferramentas de banco de dados XML nativos, a tendência, na consulta Q6, é um tempo de resposta praticamente constante, independente do tamanho da amostra.

5.5 Discussões sobre os Experimentos

Os experimentos realizados neste trabalho mostraram que as ferramentas existentes para a consulta de dados XML em memória podem ser suficientes para realizarem operações de análise de código em pequenas partes ou módulos das aplicações.

Para amostras pequenas, em consultas do tipo críticas de projeto (onde estas críticas se refiram a cada arquivo fonte de forma separada), o tempo de resolução das consultas é muito próximo entre as ferramentas de banco de dados nativo XML e das ferramentas de processamento em memória.

Para consultas do tipo “recuperar um atributo”, as ferramentas XQuery de banco de dados XML nativo possuem um tempo melhor do que as de memória, assim como para consultas do tipo métrica de projeto e para as do tipo “recuperar a hierarquia de classes”.

Demonstra-se ainda que as ferramentas de memória possuem limitações na execução de amostras de maior porte. Isto indica que uma configuração de máquina superior poderia ser necessária para que essas ferramentas possam ser usadas em aplicações de grande porte.

Vale ressaltar que se amostras fossem consultadas de forma diluída, ou seja, em pequenas porções de dados (como o tamanho das amostras 1 e 2), possivelmente a execução seria bem sucedida.

Para os resultados nas ferramentas de banco de dados, mostrou-se que de fato elas oferecem uma escalabilidade muito superior com pelo menos uma ferramenta sendo capaz de suportar todas as consultas em todas as amostras consideradas.

5.6 Conclusão

Este capítulo descreveu os resultados preliminares da análise de desempenho de algumas ferramentas para consulta de dados XML as quais podem ser utilizadas como parte do framework XCARE, na implementação de vários tipos de operações de análise de código.

Demonstrou também os limites de cada ferramenta utilizada para diferentes tamanhos de aplicações uma determinada configuração de máquina.

No próximo capítulo apresentaremos as conclusões deste trabalho, descrevendo suas principais contribuições e algumas possíveis linhas de pesquisa para trabalhos futuros.

CAPÍTULO 6 - Conclusão e Trabalhos Futuros

6.1 Benefícios do Trabalho

As tarefas de análise de código durante algum tempo foram dominadas por ferramentas proprietárias, de difícil implementação, customização e reuso.

O uso de padrões abertos XML oferece aos desenvolvedores possibilidades de escolha entre diversas ferramentas e tecnologias para a realização das tarefas de análise de código.

Este trabalho demonstrou a viabilidade de se usar tecnologias abertas e padronizadas, baseadas em XML, para facilitar a implementação de ferramentas de análise de código.

O framework XCARE permite uma flexibilização na escolha das ferramentas que manipulam XML, na implementação das consultas e na utilização em várias linguagens de programação, para as tarefas de análise de código.

Entretanto vale ressaltar que o framework XCARE foi desenvolvido em Java e as ferramentas utilizadas precisam ter compatibilidade com esta tecnologia. Além disso as linguagens de programação indicadas para a utilização do framework dependem da modelagem XML criada para a representação da linguagem.

Comparadas às ferramentas tradicionais, estas ferramentas oferecem vários benefícios:

- utilização de modelos de dados e mecanismos de consulta padronizados;
- independência de plataformas específicas de desenvolvimento e processamento de consultas XML;
- possibilidade de reuso das operações de análise em diferentes contextos e para diferentes linguagens de programação.

6.2 Contribuições

Apresentamos as principais contribuições desse trabalho:

- Mostrar a viabilidade de se utilizar padrões e tecnologias baseadas em XML para construir ferramentas de análise de código que sejam mais facilmente portáveis e configuráveis por parte do usuário.
- Avaliar o desempenho das tecnologias de consulta a dados XML mais recentes para viabilizar a análise de informações de código extraídas de sistemas “reais” de variados portes.
- Promover o desenvolvimento de novas pesquisas na área de engenharia reversa e análise de código, oferecendo um repositório aberto para o desenvolvimento e compartilhamento de diversos tipos de ferramentas por parte de pesquisadores e fabricantes de produtos comerciais.

6.3 Trabalhos Futuros

Como trabalhos futuros podemos vislumbrar a realização de mais experimentos com novas aplicações e novos tipos de operações de análise de código, onde serão considerados os tempos de conversão dos arquivos fonte em XML.

Definir um padrão XML único que represente todas as linguagens, desenvolvendo novos conversores para linguagens de programação ainda sem representação em XML definida, tendo a preocupação com o tamanho do arquivo XML gerado, pois este também pode ser um problema para as análises do código, quando usamos uma ferramenta de consulta a XML em memória.

O tempo de conversão dos arquivos fontes para XML deve ser também pesquisado e analisado para o efetivo uso da tecnologia XML.

Outra linha de pesquisa é a geração para cada linguagem escolhida na análise de código, das funções bases específicas da linguagem, ou seja, a estrutura sintática da linguagem de programação deverá ser analisada para descobrirmos quais funcionalidades serão implementadas nas funções bases.

A integração de uma instância deste framework, por exemplo, para a linguagem Java, a uma IDE de desenvolvimento (por exemplo, Eclipse) através da implementação de um plug-in, seria mais uma linha de pesquisa futura.

Outro trabalho futuro seria integrar o framework XCARE com o framework RefaX [41][43], que utiliza tecnologias XML para as tarefas de refactoring de código, produzindo um único framework para as atividades de análise de código, engenharia reversa e refactoring.

Referências Bibliográficas

[1] Badros, G. J. “JavaML: A Markup Language for Java Source Code”, In Proceedings of the 9 th International World Wide Web Conference (WWW9), Amsterdam, The Netherlands, May 13-15, 2000.

[2] BeautyJ. Disponível em http://beautyj.berlios.de/. Acessado em 01/05/2003.

[3] Riva, Claudio; Yaojin, Yang. “An Environment for Architecture Reconstruction”. SWARM - Software Architecture Recovery and Modelling. WCRE 2001 Discussion Forum. Stuttgart, 2 October, 2001.

[4] Clifton, G. T. Leavens; Chambers,C.; Millstein, T. “MultiJava: Modular Open Classes and Symetric Multiple Dispatch for Java”. In Proceedings of OOPSLA 2000, volume 35, pages 130—145, October 2000.

[5] Collard, M.L; Maletic, J.I; Marcus, A. “Source Code Files as Strutctured Documents”. in Proceedings of 10th IEEE International Workshop on Program Comprehension (IWPC'02) (Paris, France, June 27-29, 2002), 289-292.

[6] Cppx – Disponível em http://www.swag.uwaterloo.ca/~cppx. Acessado em 07/03/2005.

[7] Derby. Disponível em http://www.gael.fr/derby/index.php?id=home. Acessado em 01/05/2003.

[8] Deutsch, M. Fernandez, D. Florescu, Alon Levy; D.Suciu. “XML-QL: A Query Language for XML”. In Proc. of the Query Languages workshop(QL98), Cambridge, Mass., December 1998, http:/ /www.w3.org/TR/1998/NOTE-xml-q1-19980819/.

[9] Beyer, Dirk; Noack, Andreas; Lewerentz, Claus. “Simple and Efficient Relational Querying of Software Structures”. Proceedings of the 10th Working Conference on Reverse Engineering (WCRE’03).

[10] Gamma, Erich; Sides, John Vli; Jonhson, Ralph; Helm, Richard. "Design Patterns". Addison-Wesley Pub Co; 1st edition (January 15, 1995).

[11] FAMIX - Disponível em http://iamwww.unibe.ch/~famoos/FAMIX/. Acessado em 07/03/2005.

[12] Ferramentas GXL (1.0) Tools – Disponível em http://www.gupro.de/GXL/tools/tools.html. Acessado em 07/03/2005.

[13] Fujaba – Disponível em http://www.uni-koblenz.de/~ist/gupro.en.html. Acessado em 07/03/2005.

[14] McArthur, G.; Mylopoulos, J.; Keith, S. K. Ng. “An Extensible Tool for Source Code Representation Using XML”, In Proceedings of the 9 th Working Conference on Reverse Engineering (WCRE’02). October 29 - November 01, 2002 Richmond, Virginia.

[15] Galax. Disponível em http://db.bell-labs.com/galax/. Acessado em 01/05/2003.

[16] GenSet – Disponível em http://www.cs.uoregon.edu/research/perpetual/Software/GenSet/index.html. Acessado em 07/03/2005.

[17] GraLab – Disponível em http://www.uni-koblenz.de/~ems/GraLab4/. Acessado em 07/03/2005.

[18] Graph Tool – Disponível em http://vrg.dur.ac.uk/. Acessado em 07/03/2005.

[19] GRAS Graph Database – Disponível em http://www-i3.informatik.rwth-aachen.de/research/projects/gras/index.html. Acessado em 07/03/2005.

[20] GXL Graphpad – Disponível em http://gxl.sourceforge.net/. Acessado em 07/03/2005.

[21] Holt, R.C.; Winter, A; Schür, A.“GXL: Toward a Standard Exchange Format”. In Proceedings of the 7th Working Conference on Reverse

Engineering (WCRE'00), pp. 162-171, Brisbane, Queensland, Australia, November 23-25, 2000.

[22] IDE Eclipse para Java. Disponível em http://www.eclipse.org/. Acessado em 01/05/2003.

[23] IPSI-XQ. Disponível em http://www.ipsi.fraunhofer.de/oasys/projects/ipsi-xq/. Acessado em 01/05/2003.

[24] J. Robie, J. Lapp, D. Schach. "XML Query Language (XQL)". http://www.w3.org/TandS/QL/QL98/.

[25] JavaCC. Disponível em https://javacc.dev.java.net/. Acessado em 01/12/2004.

[26] Pace, Jeff. "DoctorJ – Diagnoses error in JavaDoc and Java Code", http://doctorj.sourceforge.net/

[27] Korn, Jeffrey L.; Chen, Yih-Farn; Koutsofios, Eleftherios. "Chava: Reverse Engineering and Tracking of Java Applets", In Proceedings of 7 th Working Conference on Reverse Engineering (WCRE'00), Brisbane, Austrália, pp. 314-325, November 23-25, 2000.

[28] Gulden, Jens. "BeautyJ- Customizable Java Source Code Transformer & Sourclet API with Java Source Code Parser API", http://beautyj.berlios.de/beutyj.html/

[29] JGraph – Disponível em http://sourceforge.net/projects/jgraph/. Acessado em 07/03/2005.

[30] JGraphpad – Disponível em http://www.jgraph.com/jgraphpad.shtml. Acessado em 07/03/2005.

[31] JHotDraw – Aplicação para construção de gráficos técnicos e estruturados. Acessado em 10/11/2004. http://www.jhotdraw.org/.

[32] Jonathan I. Maletic, Michael L. Collard and Andrian Marcus, "Source Code Files as Structured Documents", In Proceedings of the 10 th

International Workshop on Program Comprehension (IWPC '02), Paris, France, pp. 289-292, June 27-29, 2002.

[33] JWAM - Java framework for Swing. Acessado em 10/11/2004 http://www.jwam.de/engl/e_index.htm.

[34] Kawa. Disponível em http://www.gnu.org/software/kawa. Acessado em 01/05/2003.

[35] Linguagem Java. Sun Microsystems. Disponível em java.sun.com/. Acessado em 01/05/2003.

[36] Maletic, J.I; Collard, M. L.; Kagdi, H.H., "A XML-Based Lightweight C++ Fact Extractor", In Proceedings of the 11 th IEEE International Workshop on Program Comprehension (IWPC'03), Portland, USA, May 10-11, 2003.

[37] Mammas, E. and Kontogiannis, C., "Towards Portable Source Code Representations Using XML", In Proceedings of the 7 th Working Conference on Reverse Engineering (WCRE'00), Brisbane, Queenlsand, Australia, pp. 171-182, November 23 –25, 2000.

[38] MOF - Meta Object Facility – Disponível em http://www.omg.org/mof. Acessado em 07/03/2005.

[39] MsXquery Demo. Disponível em http://www.topxml.com/xquery/xquery_demo.asp. Acessado em 01/05/2003.

[40] MultiJava. Disponível em http://multijava.sourceforge.net/index.shtml. Acessado em 01/05/2003.

[41] Mendonça, Nabor C.; Maia, Paulo Henrique M.; Fonseca, Leonardo A.; Andrade, Rossana M. C. "RefaX: A Refactoring Framework Based on XML". 20th IEEE International Conference on Software Maintenance (ICSM'04). September 11 - 14, 2004. Chicago, Illinois.

[42] Mendonça, Nabor C.; Fonseca, Leonardo A.; Maia, Paulo Henrique M. "Towards Reusable Code Analysis Tools Using Standard XML Technologies". Apresentado no 1º Workcomp Sul, Florianópolis, em Maio de 2004.

[43] Mendonça, Nabor C.; Maia, Paulo Henrique M.; Fonseca, Leonardo A.; Andrade, Rossana M. C. "Building Flexible Refactoring Tools with XML". 18º Simpósio Brasileiro de Engenharia de Software.SBES 2004. Brasília, 18 a 22 de outubro de 2004.

[44] OMG – "XML Metadata Interchange (XMI), version 1.2", Disponível em http://www.omg.org/technology/documents/formal/xmi.htm.

[45] Pressman, Roger S. "Software Engineering – A Practitioner's Approach". 5th Edition, McGraw Hill, 2001.

[46] Qexo. Disponível em http://www.gnu.org/software/qexo. Acessado em 01/05/2003.

[47] Qizx/open. Disponível em http://www.xfra.net/qizxopen/. Acessado em 01/05/003.

[48] Ferenc, R.; Beszédes, A.; Trakiainen, M.; Gyimóthy, T. "Columbus – Reverse Engineering Tool and Schema for C++", In Proceedings of the International Conference on Software Maintenance (ICSM'02), 2002.

[49] Rational Rose Home page. Disponível em http://www.rational.com/products/rose/. Acessado em 01/05/2003.

[50] Rigi – Disponível em http://www.rigi.csc.uvic.ca/. Acessado em 07/03/2005.

[51] Robie, J.; Chamberlin, D.; Florescu, D. 2000. "Quilt: an XML Query Language." http://www.almaden. ibm.corrgcslpeople/chamberlin/quilt euro.htmi.

[52] Burson, Scott; Kotik, Gordon B.; Markosian, Lawrence Z. "A Program Transformations Approach to Automating Software Re-Engineering", In

Proceedings of the 14 th Annual International Computer Software and Applications Conference (COMPSAC'90), Amsterdan, Holland, pp. 314-322, 1990.

[53] Software AG. QuiP XQuery Prototype. Disponível em http://developer.softwareag.com/tamino/quip/ . Acessado em 01/05/2003.

[54] Source Navigator – Cláudio Riva. Disponível via email - claudio.riva@nokia.com.

[55] Tamino. Disponível em http://www.softwareag.com/tamino. Acessado em 01/05/2003.

[56] TogetherSoft Corporation. Together Home page. Disponível em http://www.togethersoft.com . Acessado em 01/05/2003.

[57] Chu, Tommy. "Benchmarking on Java – XML Transformers". Proposta de Projeto da Disciplina CMPUT 664 - Software Reverse Engineering— Computing Science — University of Alberta – Canadá.

[58] UML - Unified Modeling Language – Disponível em http://www.uml.org/. Acessado em 07/03/2005.

[59] Venice – Disponível em http://www.cs.helsinki.fi/group/venice/. Acessado em 07/03/2005.

[60] VENICE UML Visualization Tool. Department of Computer Science, University of Helsinki, 2001. Disponível em http://www.cs.Helsinki.FI/group/venice/

[61] W3C. "Extensible Markup Language (XML)". Disponível em http://www.w3.org/TR/xml. Acessado em 01/05/2003.

[62] W3C. "XML Path Language (XPath) Version 1.0". W3C Recommendation November 1999. Disponível em http://www.w3.org/TR/xpath.Acessado em 01/05/2003.

[63] W3C. “XQuery 1.0: An XML Query Language”. W3C Working Draft 02, May 2003. Disponível em http://www.w3.org/TR/XQuery/ . Acessado em 07/05/2003.

[64] W3C. “XSL Transformations (XSLT) Version 1.0”. W3C Recommendation November 1999. Disponível em http://www.w3.org/TR/xslt. Acessado em 01/05/2003.

[65] W3C. XML Transformations (XSLT). http://www.w3.org/TR/xslt.

[66] World Wide Web Consortium, W3C Home page. Disponível em http://www.w3.org. Acessado em 01/05/2003.

[67] XHive. Disponível em http://www.x-hive.com/xquery/. Acessado em 01/05/2003.

[68] Xindice. Disponível em http://www.xindice.org/. Acessado em 13/01/2004.

[69] XML Notepad. Disponível em http://www.xml.com/pub/r/453. Acessado em 01/05/2003.

[70] XML Spy Home page. Disponível em http://www.xmlspy.com . Acessado em 01/05/2003.

[71] XML-DB. “Xupdate – XML Update Language Working Draft”. Disponível em http://www.xmldb.org/xupdate/ . Acessado em 01/05/2003.

[72] XQEngine. Disponível em http://www.fatdog.com/. Acessado em 01/05/2003.

[73] XStreamDB. Disponível em http://www.bluestream.com/. Acessado em 01/05/2003.

[74] Yih-Farn, R. Chen et al., “Ciao: A Graphical Navigator for Software and Document Repositories”. In Proceedings of the International Conference on Software Mantainance (ICSM’99), pp. 3-14, San Antonio, Texas.

Anexos

Consultas Implementadas em XQuery

CLASSIFICAÇÃO: ANÁLISE DE CÓDIGO

Q1. RETORNA O NOME DE TODOS OS ATRIBUTOS DA CLASSE QUE SEJAM DO TIPO "INT"

JAVAML - BADROS

```
<Nome_do_atributo>
{
  let $doc := document("Car.xml")
  for $a in $doc/java-source-program/java-class-file/class//field
  where $a/type/@name = "int"
  return
    <atributo name= "{$a/@name}"></atributo>
}</Nome_do_atributo>
```

JAVAML – MAMAS

```
<Nome_do_atributo>
{
  let $doc := document("Car.xml")
  for $a in $doc/CompilationUnit/TypeDeclaration/ClassDeclaration/
UnmodifiedClassDeclaration/ClassBody/FieldDeclaration
  where $a/type/PrimitiveType/@Type = "int"
  return
    <atributo name= "{$a/VariableDeclarator/VariableDeclaratorId/@Identifier}"></atributo>

}</Nome_do_atributo>
```

BEAUTYJ

```
<Nome_do_atributo>
{
  let $doc := document("Car.xml")
  for $a in $doc/xjava/package//class//field
  where $a/type/@name = "int"
  return
    <atributo name= "{$a/@name}"></atributo>
}</Nome_do_atributo>
```

CLASSIFICAÇÃO: MÉTRICA

Q2. A MÉTRICA WMC IMPLEMENTADA EM XQUERY

JAVAML - BADROS

```
<WMC>{
        let $doc := document("Car.xml")
        FOR $class in $doc//java-source-program/java-class-file/class
        RETURN
                count($class//method)
}</WMC>
```

JAVAML – MAMAS

```
<WMC>{
  let $doc := document("Car.xml")
  FOR $class in $doc/CompilationUnit/TypeDeclaration/ClassDeclaration/  UnmodifiedClassDeclaration/ClassBody
  RETURN
                count($class//MethodDeclaration)
}</WMC>
```

BEAUTYJ

```
<WMC>{
        let $doc := document("Car.xml")
        FOR $class in $doc/xjava/package//class
        RETURN
                count($class//method) }</WMC>
```

CLASSIFICAÇÃO: CRÍTICA

Q3. A CRÍTICA DE PROJETO UNINSTANTIABLE PUBLIC IMPLEMENTADA EM XQUERY

JAVAML - BADROS

```
let $doc := document("Car.xml")
for $class in $doc//java-source-program/java-class-file/class
where (($class/@visibility = "public") and
        not(exists($class/constructor))
return $class
```

JAVAML – MAMAS

```
let $doc := document("Car.xml")
for $class in $doc/CompilationUnit/TypeDeclaration/ClassDeclaration
  where (($class/@isPublic= "True") and
        not(exists($class/UnmodifiedClassDeclaration/ClassBody/ConstructorDeclaration))
return $class
```

BEAUTYJ

```
let $doc := document("Car.xml")
for $class in $doc/xjava/package//class
where (($class/@public = "yes") and
        not(exists($class/constructor))
return $class
```

Q4. A CRÍTICA DE PROJETO CLASS WITH ONLY ATTRIBUTES AND NO METHODS SUGGEST INADEQUATE BEHAVIOR DISTRIBUTION

JAVAML - BADROS

```
let $doc := document("Car.xml")
for $c in $doc//java-source-program/java-class-file/class
where exists($c//:attribute) and
     (not(exists($c//method)))
return
  <critic class="{$c/@name}">Class with only attributes and no methods
    suggest inadequate behavior distribution</critic>
```

JAVAML – MAMAS

```
let $doc := document("Car.xml")
```

```
for $c in $doc/CompilationUnit/TypeDeclaration/ClassDeclaration/UnmodifiedClassDeclaration/ClassBody
where exists($c//FieldDeclaration) and
      (not(exists($c//MethodDeclaration)))
return
  <critic class="{$c/../@Identifier}">Class with only attributes and no methods
     suggest inadequate behavior distribution</critic>
```

BEAUTYJ

```
let $doc := document("Car.xml")
for $c in $doc/xjava/package//class
where exists($c//field) and
      (not(exists($c//method)))
return
  <critic class="{$c/@name}">Class with only attributes and no methods
     suggest inadequate behavior distribution</critic>
```

Q5. A CRÍTICA DE PROJETO PUBLIC CLASS WITH NO PUBLIC FEATURES DON'T PRESENT ANYTHING USEFUL

JAVAML - BADROS

```
let $doc := document("Car.xml")
for $c in $doc//java-source-program/java-class-file/class
where (($c/@scope = "public")or ($c/@scope = ""))
and
  not(exists(($c//:attribute/@scope="public") or    ($c//:attribute/@scope="")))
and
  not(exists(($c//method/@scope="public") or ($c//method/@scope="")))
return
 <critic class="{$c/@name}"> Public class with no public features
    don't present anything useful </critic>
```

JAVAML – MAMAS

```
let $doc := document("Car.xml")
for $c in $doc/CompilationUnit/TypeDeclaration/ClassDeclaration/UnmodifiedClassDeclaration/ClassBody
where ($c/../../@isPublic = "True")
and
  not(exists($c//FieldDeclaration /@isPublic="True"))
and
  not(exists(($c//MethodDeclaration/@isPublic="True"))
return
 <critic class="{$c/@name}"> Public class with no public features
    don't present anything useful </critic>
```

BEAUTYJ

```
let $doc := document("Car.xml")
for $c in $doc/xjava/package//class
where ($c/@public = "yes")
and
  not(exists($c//field/@public="yes"))
and
  not(exists(($c//method/@public="yes"))
return
 <critic class="{$c/@name}"> Public class with no public features
    don't present anything useful </critic>
```

CLASSIFICAÇÃO: ENGENHARIA REVERSA

Q6. CONSULTA XQUERY QUE RECUPERA A HIERARQUIA DE CLASSES.

JAVAML - BADROS

```
let $doc := document("Car.xml")
for $class in $doc//java-source-program/java-class-file/class
return
<inherit from="{string($class/@name)}"
        to="{string($class/superclass/@name)}"/>
```

JAVAML – MAMAS

```
let $doc := document("Car.xml")
for $class in $doc/CompilationUnit/TypeDeclaration/ClassDeclaration/UnmodifiedClassDeclaration
Where
          $class/@Extends=”True”
return
<inherit from="{string($class/@Identifier)}"
        to="{string($class/name/@Identifier)}"/>
```

BEAUTYJ

```
let $doc := document("Car.xml")
for $class in $doc/xjava/package//class
return
<inherit from="{string($class/@name)}"
        to="{string($class/extends/@class)}"/>
```

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo