UMA FERRAMENTA DE INSPEÇÃO PARA APLICAÇÕES JAVA UTILIZANDO REFLEXÃO COMPUTACIONAL

André Luis Castro de Freitas[1,2]
afreitas@inf.ufrgs.br

Ana Maria de Alencar Price[1]
anaprice@inf.ufrgs.br

[1]UFRGS - Universidade Federal do Rio Grande do Sul
Porto Alegre – RS – BRASIL
[2]FURG – Universidade Federal do Rio Grande
Rio Grande – RS - BRASIL

RESUMO

É reconhecido que, para tornar-se um projetista especializado em técnicas de desenvolvimento de software Orientado a Objetos, exige-se um maior esforço de dedicação e treinamento comparando-as com o aprendizado de técnicas estruturadas tradicionais. Portanto, a manutenção de um software desta natureza torna-se também um processo delicado para um projetista não especializado na linguagem na qual o software fora descrito. Visando facilitar o entendimento de tal software este trabalho descreve uma ferramenta desenvolvida para auxiliar o processo de inspeção e visualização de aplicações construídas na linguagem Java. Acredita-se que esta ferramenta possa ajudar a reduzir a complexidade inerente do processo de compreensão de um software em Java propondo auxiliar o projetista na construção automática de modelos de projeto, através de mecanismos de análise, exploração e visualização da informação em diferentes níveis de abstração.

Palavras-chave: Engenharia de Software, Reflexão Computacional, Linguagem Java.

ABSTRACT

It is recognized that to become a specialized designer in techniques of object-oriented software development it is demanded a greater dedication and training comparing them with the learning of traditional structured techniques. Therefore, the maintenance of a software of this nature becomes also a delicate process for a designer not specialized in the language in the which the software had been described. Seeking to facilitate the understanding of such software this work describes a tool developed to aid the inspection process and visualization of applications built in the Java language. It is believed that this tool can help to reduce the inherent complexity of the process of understanding of a software in Java intending to aid the designer in the automatic construction of project models through analysis mechanisms, exploration and visualization of the information in different abstraction levels.

Key-words: Software Engineering, Computational Reflection, Java Language.

1 INTRODUÇÃO

As metodologias de desenvolvimento e as linguagens de programação evoluíram durante as últimas décadas, com o objetivo de minimizar os problemas de manutenção de software. Entretanto estas mesmas metodologias falham no fornecimento de suporte adequado à compreensão dos sistemas envolvidos. Segundo Arango et al. (1993), as metodologias de desenvolvimento de software orientado a objetos, em geral, "*falham na provisão de formalismos adequados para refletir o mapeamento entre o projeto e a implementação"*.

A reutilização de software, em contrapartida ao desenvolvimento de aplicações, desde o início é vista como um fator que pode levar ao aumento de produtividade de software, pois o uso de elementos de software já desenvolvidos e depurados, reduz o tempo de desenvolvimento, de testes e as possibilidades de introdução de erros. A reutilização de elementos de software pode ocorrer a nível de projeto ou de código. A reutilização de projeto consiste no reaproveitamento de concepções arquitetônicas de uma aplicação em outra aplicação, não necessariamente com a utilização da mesma implementação. A reutilização de código consiste na utilização direta de trechos de código já desenvolvidos (CAMPO, 1997).

Neste sentido, tanto para o processo de manutenção como para o de reaproveitamento de código, caracteriza-se de grande importância o conhecimento, por parte do projetista, das estruturas estáticas e dinâmicas da aplicação desenvolvida pois uma aplicação orientada a objetos constrói-se com classes que colaboram entre si, através da troca de mensagens, para realizar as tarefas do sistema.

Com o objetivo de auxiliar os processos de manutenção e reaproveitamento de código existente de software orientado a objetos, este trabalho propõe o desenvolvimento de uma ferramenta de inspeção e visualização automáticas para aplicações Java.

Este trabalho está organizado nas seções 2, 3 e 4. A seção 2 define e caracteriza a ferramenta em seu estado atual. Na seção 3 é construído e avaliado um exemplo de aplicação em Java. Por fim na seção 4 faz-se a demonstração da ferramenta sobre a aplicação Java considerada. Após procede-se a conclusão e trabalhos futuros.

2 FERRAMENTA

Esta seção apresenta o protótipo da ferramenta que tem por objetivo automatizar a identificação e classificação de colaborações entre classes e/ou objetos em aplicações Java. Esta identificação é feita por meio dos processos de: engenharia reversa e reflexão computacional.

A Engenharia Reversa tem como objetivos extrair informações da especificação de um software para posterior análise no intuito de identificar os componentes da aplicação e seus inter-relacionamentos. A partir do código fonte em Java faz-se a identificação dos componentes da aplicação através de um gerador de referência cruzada. Um gerador de referência cruzada descreve relações e referências para entidades como classes, métodos e atributos analisando gramaticalmente uma aplicação fonte em Java. Estas relações são apresentadas em um relatório descrevendo quais entidades referem-se a outras entidades. O gerador de referencia cruzada foi implementado utilizando a ferramenta ANTLR. ANTL, *ANother Tool for Language Recognition*, é uma ferramenta que provê um *framework* para construção de reconhecedores, compiladores, e tradutores a partir de gramáticas contendo descrições para aplicações C++ ou Java (PARR, 2001).

Após a construção do relatório de referência cruzada são identificados, no código fonte, os atributos a serem avaliados e selecionados para envio de informações para a ferramenta no posterior processo de execução. A aplicação do projetista passa a ser um pacote da ferramenta e como tal deve ser compilada. Após a compilação a ferramenta executa a aplicação do projetista e num processo conjunto com a execução caracteriza-se o processo de reflexão computacional.

A reflexão computacional define uma arquitetura em níveis, denominada arquitetura reflexiva. Em uma arquitetura reflexiva, um sistema computacional é visto como incorporando dois componentes: um representando o objeto, e outro a parte reflexiva. As computações do objeto, localizado no nível base, tem por objetivo resolver problemas e retornar informações sobre o domínio da aplicação, enquanto o nível reflexivo, localizado no meta-nível, resolve os problemas e retorna informações sobre as computações do objeto, podendo adicionar funcionalidades extras a este objeto (MAES, 1987).

O estado atual da ferramenta mostra como extrair características de objetos através de suas colaborações e apresenta a visualização dos objetos e/ou classes em análise. A partir de então são

apresentadas janelas com diagramas de classes e objetos bem como janelas de inspeção de conteúdos para os atributos envolvidos.

3 APLICAÇÃO JAVA

Considere-se a figura a seguir (Figura 1) composta por uma hierarquia de classes que representam o ativo patrimonial de uma determinada pessoa. *Ativo* implementa a classe que oferece a interface para a estrutura, *AtivoComposite* define o comportamento para os objetos compostos e *Garantia* para os objetos primitivos.

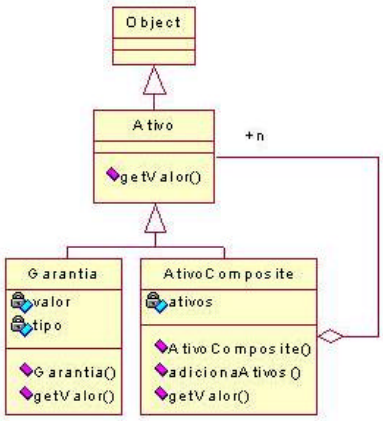

Figura 1: Aplicação Java

Faz-se a criação de um objeto *AtivoComposite*, denominado *listaMonetario* que inicializa o atributo *ativos* (do tipo *Vector*) através da mensagem *new*. Em seguida, é enviada a mensagem *adicionaAtivos*: para *listaMonetario* que procede a inserção dos elementos primitivos (*Garantia*) ao vetor. Os quatro elementos: *aplicacoes*, *conta corrente*, *poupanca* e *acoes* são inseridos no vetor como objetos da classe *Garantia*.

A estrutura utilizada para criação de *listaBens* é a mesma de *listaMonetario*. Através da mensagem *adionaAtivos*: para *listaBens* são inseridos no vetor *ativos* os objetos *imoveis* e *semoventes* da classe *Garantia*. O mecanismo de inserção em *AtivoComposite* permite a inserção de *listaMonetario* como elemento do vetor *listaBens*. A estrutura a seguir mostra uma árvore de objetos.

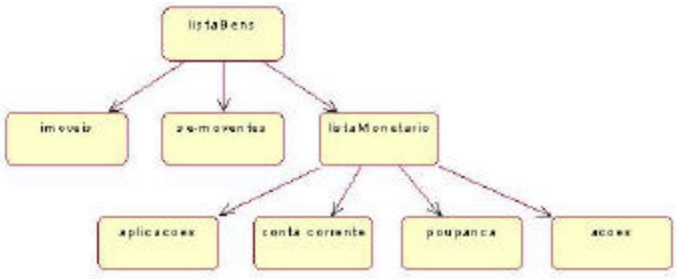

Figura 2: Diagrama de Objetos

Para a pesquisa de uma informação na árvore utiliza-se o método *getValor()* que executa o somatório junto aos nodos primitivos ou junto a possíveis vetores na estrutura, de maneira recursiva. O resultado do somatório é representado pela soma do atributo valor de todos os objetos primitivos.

4 FERRAMENTA EM EXECUÇÃO

Para mostrar o funcionamento da ferramenta, será usado o exemplo descrito na Seção 3. Através da janela principal o projetista pode selecionar a aplicação (código fonte em Java) a ser executado e inspecionado. No caso fora selecionado o arquivo CompositeOriginal.java.

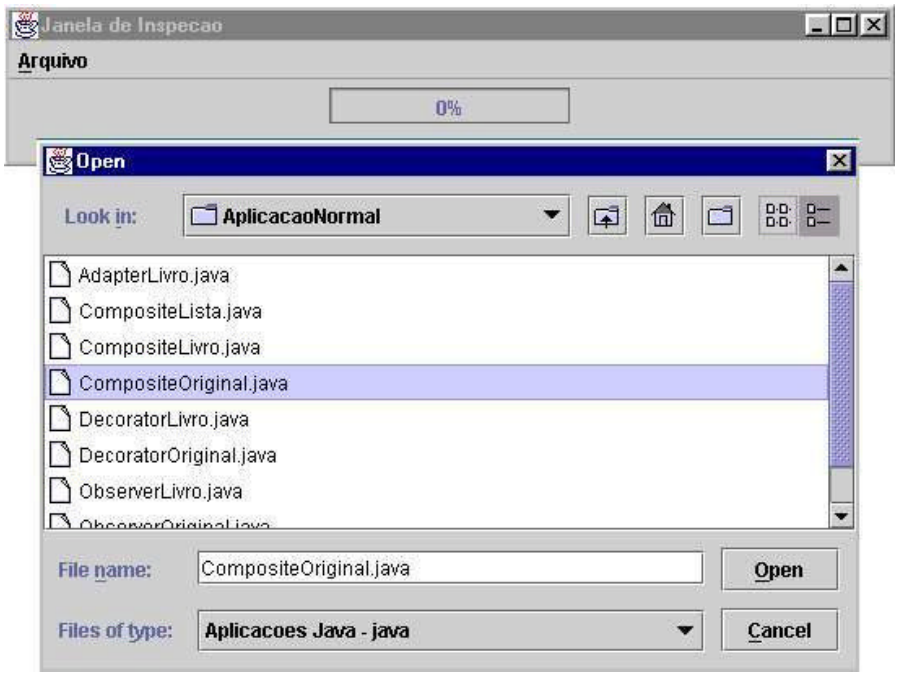

Figura 3: Interface inicial da aplicação

Após a identificação do arquivo é iniciado o módulo de verificação de referência cruzada que procede a investigação das classes, métodos e atributos existentes no arquivo. A Figura 4 mostra a execução do pacote de referência cruzada enquanto este procede a construção do relatório final. Após a conclusão da referência cruzada passa-se a transformação da aplicação do projetista onde o método *main()* é transformado para um método *runProcessing(),* pois em tempo de execução a aplicação do projetista comportar-se-á como um pacote trabalhado pela ferramenta de inspeção. Todos os objetos identificados na aplicação principal serão selecionados para captura de informações em posterior tempo de execução. Através do envio destes objetos para uma *Thread* de controle, a ferramenta pode, em tempo de execução, caracterizar a inspeção destes elementos.

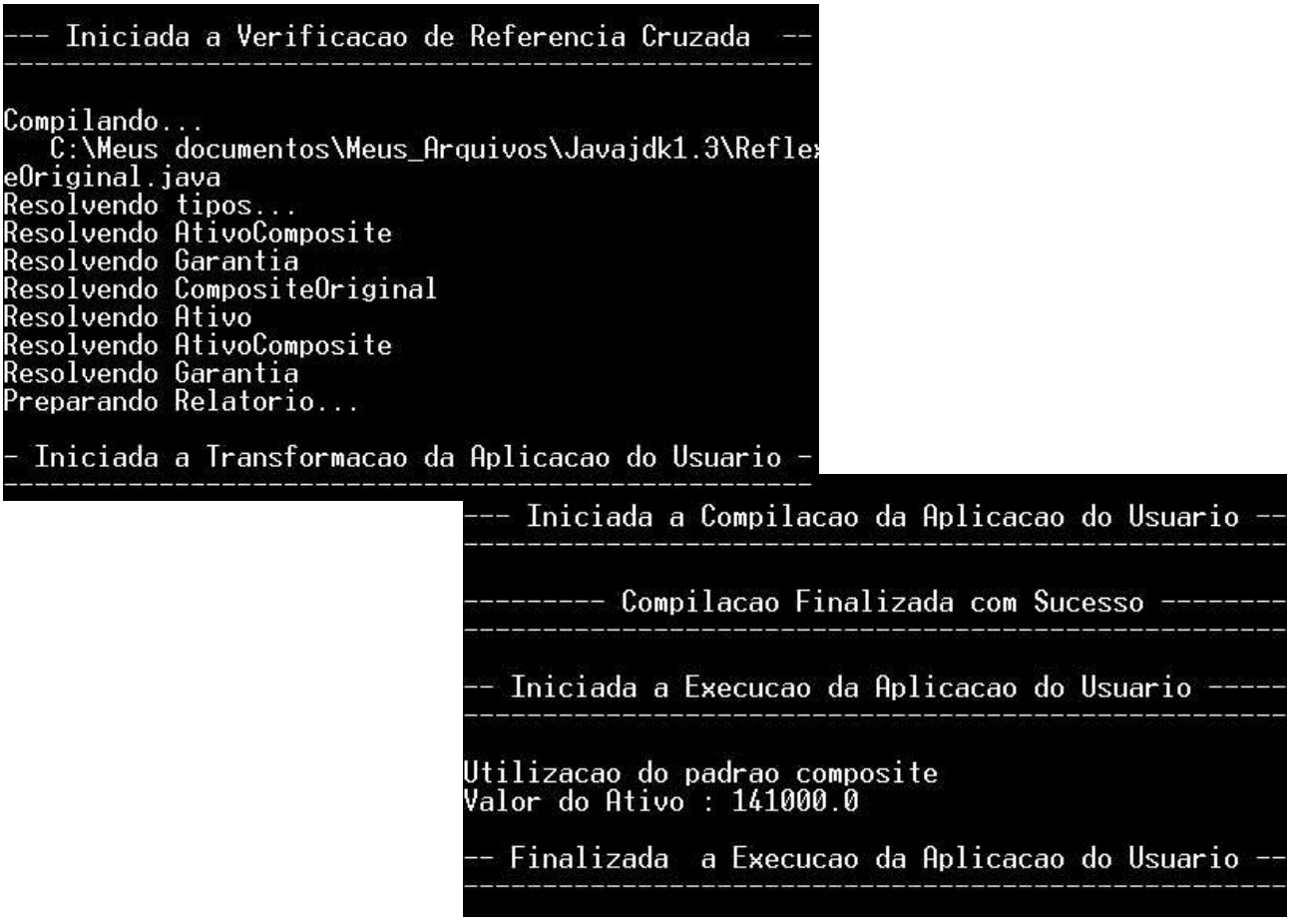

Figura 4: Janela de execução da aplicação

Após o processo de modificação é iniciada a compilação da aplicação do projetista. O processo de compilação ativa o compilador JDK através de uma chamada externa. Considerando a compilação finalizada sem problemas a ferramenta passa para a fase de execução do pacote do projetista. Durante a execução da aplicação a *Thread* de controle recebe, em intervalos de tempo, os objetos extraídos. A extração é feita para cada objeto que recebe mensagens diferentes na aplicação

do projetista e para tanto a ferramenta avalia os estados destes objetos. Se na avaliação, o objeto apresentar estado diferente do armazenado anteriormente este será considerado, caso contrário será descartado.

Após a finalização do pacote do projetista a ferramenta irá apresentar as interfaces: de referência cruzada, de estados dos objetos encontrados na aplicação e de classes dos objetos identificados, conforme Figura 5. Na referência cruzada o projetista pode proceder a procura na aplicação fonte das entidades identificadas. Neste caso a ferramenta aponta para o objeto *listaBens* da classe *AtivoComposite* conforme citado na seção 3. Para este objeto é apresentada uma janela para verificação de seus estados bem como uma janela para verificação de sua classe, superclasse e atributos existentes nestas.

Na interface de estados são apresentados os estados que o objeto caracterizou em tempo de execução. O projetista pode selecionar o que julga mais pertinente e proceder a inspeção destes estados, através da opção *VerificaEstado*.

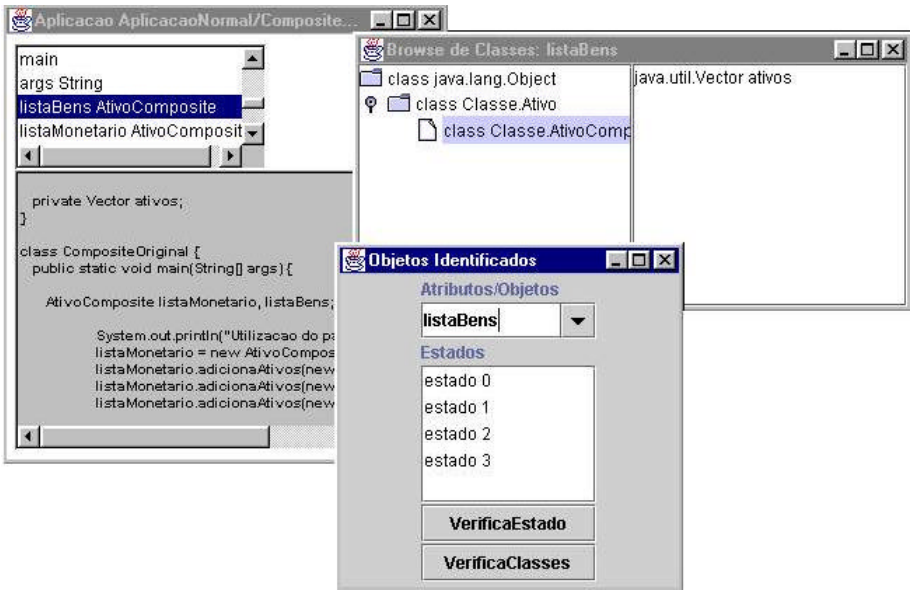

Figura 5: Interface de apresentação de estados

No caso a seguir fora selecionado o estado 3 para o objeto *listaBens*. A partir da identificação das colaborações deste com outros objetos e/ou classes faz-se a apresentação visual e textual de seus relacionamentos. A ferramenta distingue os objetos determinando para referência àqueles objetos não considerados como literais para o compilador. O atributo *ativos* da classe *AtivoComposite* apresenta uma relação com dois objetos da classe *Garantia* e um outro objeto de

AtivoComposite. O segundo *AtivoComposite* apresenta uma relação com quatro objetos da classe *Garantia.*

Figura 6: Diagrama de objetos

A janela de inspeção textual, Figura 7, apresenta o conteúdo dos atributos dos objetos identificando novos objetos associados pelos atributos. A janela, portanto, identifica o objeto da classe *AtivoComposite* (*listaBens* citado na seção três), demonstrando suas relações com objetos da classe *Garantia* (*imóveis* e *se-moventes*) o qual são devidamente inspecionados. A relação com outro objeto *AtivoComposite* (*listaMonetario*) também é demonstrada e este, por fim, relaciona-se com quatro objetos da classe *Garantia* (*aplicacoes*, *conta corrente*, *poupanca* e *acoe*).

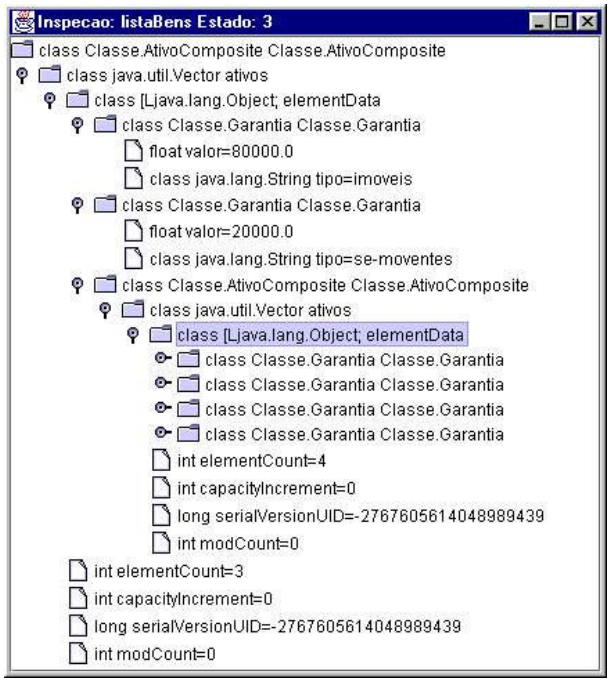

Figura 7: Inspeção de objetos selecionados

O diagrama de classes para a aplicação é construído a partir da seleção *VerificaClasses* na janela de estados. Conforme Figura 8, o diagrama apresenta a estrutura estática da aplicação caracterizando as colaborações envolvidas. A ferramenta constrói a estrutura que define as classes envolvidas na aplicação bem como verifica os indicadores de multiplicidade dos relacionamentos.

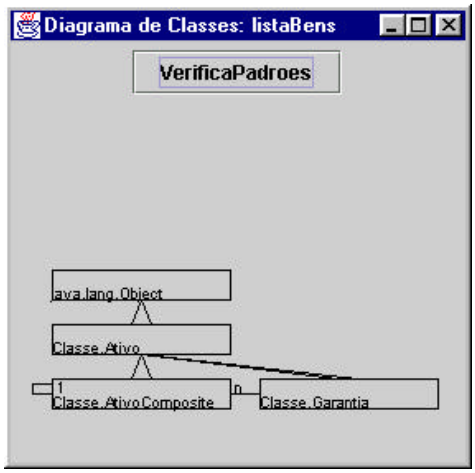

Figura 8: Diagrama de classes

A janela do diagrama de classes apresenta para o exemplo citado um relacionamento de 1:1 da classe *AtivoComposite* com ela mesma, identificado a partir do relacionamento de *listaBens* com *listaMonetario*, e um relacionamento de 1:N da classe *AtivoComposite* com *Garantia*, identificado a partir dos relacionamentos de *listaBens* com seus objetos *primitvos* e *listaMonetario*, também, com seus objetos primitivos.

5 CONCLUSÕES

Este trabalho tem como objetivo apresentar uma ferramenta para inspeção de aplicações Java em tempo de execução. A ferramenta ainda encontra-se em fase de construção mas já conta com todo o processo aqui mencionado. A ferramenta captura um ou mais objetos, a partir do código fonte, e identifica as colaborações destes objetos através da inspeção de seus atributos em tempo de execução. Estas informações são apresentadas, portanto, em janelas apropriadas onde o projetista pode diagnosticar e interagir com as informações contidas nos objetos.

O projetista pode visualizar, através de diagramas, as colaborações dos objetos com relação a outros objetos relacionados. As colaborações não se limitam somente ao objeto em teste mas sim a todos os objetos relacionados a este e assim sucessivamente. Através de um método com

chamada recursiva pesquisa-se todos os atributos dos objetos envolvidos nos relacionamentos. A ferramenta caracteriza, portanto, uma representação estática e dinâmica da aplicação em questão. Estática porque evidencia diagramas de classes dos objetos da aplicação e dinâmica porque representa as transições entre estados que um objeto pode caracterizar. O projetista pode portanto acompanhar e visualizar a execução da aplicação.

Acredita-se como trabalho futuro, que a ferramenta possa apresentar, também, diagramas de classes, com vistas a identificação de padrões de projeto GRAND (1998). Neste primeiro momento identifica-se um diagrama de classes simplificado e num segundo momento tem-se como meta prover a identificação de padrões nas aplicações e construir diagramas diferenciados pois estes possuem diferenças com relação aos diagramas originais.

Acredita-se que a validação da consistência lógica apresentada pela ferramenta será aumentada gradualmente com novos testes, portanto mais estudos de caso, envolvendo projetos reais, estão sendo preparados para certificar a ferramenta.

6 AGRADECIMENTOS

Agradeço a FAPERGS – Fundação de Amparo a Pesquisa do Estado do Rio Grande do Sul por proporcionar apoio financeiro necessário para a realização deste trabalho.

7 BIBLIOGRAFIA

ARANGO, G. et al. **A process for Consolidating and Reusing Design Knowledge**. In: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 13., 1993, Baltimore. Proceeding... California: IEEE Press, 1993.

CAMPO, M. **Compreensão Visual de Frameworks através da Introspecção de Exemplos.** Porto Alegre: CPGCC da UFRGS, 1997. Tese de Doutorado.

COAD, P. et al. **Object Models - Strategies, Patterns & Applications.** Yourdon Press: Prentice Hall Building, 1997.

GRAND, M. **Patterns in Java: A Catalog of Reusable Design Patterns with UML**. [S.l.]: John Wiley & Sons, 1998.

HORSTMANN, C; CORNELL, G. **Core Java. Volume I – Fundamentos**. Makron Books: São Paulo, 2001a.

HORSTMANN, C; CORNELL, G. **Core Java. Volume II – Avançado**. Makron Books: São Paulo, 2001b.

MAES, P. Concepts and Experiments in Computational Reflextion. In: OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES AND APPLICATIONS CONFERENCE, 1987, **Proceedings...** New York: ACM Press, 1987.

PARR, T. **What's An ANTLR?** Disponível em: <http://www.antlr.orgl>. Acesso em: 04 abr. 2001.

RIEL, A.J. **Object-Oriented Design Heuristics.** Reading: Addison-Wesley, 1996.