Simpósio Brasileiro de Sistemas de Informação

GeCA: Uma Ferramenta de Engenharia Reversa e Geração Automática de Código

Igor Steinmacher[1], Éderson Fernando Amorim[1], Flávio Luiz Schiavoni[1], Elisa Hatsue Moriya Huzita[1]

[1]Departamento de Informática
Universidade Estadual de Maringá
Av. Colombo, 5790 – CEP 87020-900 – Maringá – PR

{igor, éderson}@din.uem.br, fls@rendera.com.br, emhuzita@din.uem.br

Resumo. *Alterações durante o ciclo de vida de um software são inevitáveis. Isto pode trazer problemas quando a documentação é inexistente ou quando da necessidade de realização de constantes manutenções no sistema. Neste artigo propõe-se a ferramenta GeCA (Gerador de Códigos Auxiliares) que visa permitir aos analistas e desenvolvedores realizarem engenharia reversa e executar a manutenção dos sistemas de maneira facilitada. A ferramenta apresenta várias maneiras de obter os dados e a possibilidade de geração automática de artefatos: códigos fonte, estruturas de bancos de dados, diagramas UML e artefatos diversos (hibernate, formulários, xml).*

1. Introdução

A obsolescência de sistemas provoca a necessidade de ajustes nos sistemas para adaptá-los às novas tecnologias e, principalmente, às exigências do negócio ou dos usuários [PRESSMAN, 2002; QUINAIA; STADZISZ, 2004]. Tais adaptações podem ser referentes à correção de erros, melhora de desempenho, adição de funções, alteração do modelo ou adaptação a novas tecnologias.

Em grande parte das situações deste tipo a documentação está desatualizada (ou até mesmo inexistente) e o código é a única fonte de informação disponível para o sistema [VERONESE *et al.,* 2002]. Nestes casos, todo o conhecimento necessário sobre o software passa a estar em alguma linguagem de programação, estruturas de dados (bancos de dados) ou até mesmo apenas em códigos binários. É necessário, portanto, haver uma maneira automatizada de extrair tais informações.

Outra possível situação é a existência da documentação de um sistema, e esta estar em constante alteração. Isto demanda grande esforço de manutenção, uma vez que, em um sistema pode haver referência a uma determinada estrutura em diversos locais (replicação de código), bem como pode haver necessidade de alteração de estruturas relacionadas ao sistema principal: esquemas XML de armazenamento ou mapeamento, campos em formulários, colunas de tabelas. Neste sentido [BAXTER; MEHLICH, 1997] afirmam que sistemas maduros são muito difíceis de alterar e a validação das modificações é muito complexa.

Neste artigo propõe-se a ferramenta GeCA (Gerador de Códigos Auxiliares) que busca contemplar os dois casos, permitindo aos analistas e desenvolvedores realizar a engenharia reversa e executar a manutenção dos sistemas de maneira facilitada. No contexto do sistema consideram-se códigos auxiliares todos aqueles códigos que não são parte integrante do código relativo ao modelo, mas são essenciais ao sistema. São exemplos de códigos auxiliares: mapeamentos Hibernate (objeto-relacional), mapeamentos castor, construção de DAOs, formulários HTML.

A ferramenta apresenta como diferenciais diversas maneiras de obter os dados e a possibilidade de geração automática de artefatos relativos a diversas fases do ciclo de vida de software: códigos fonte, estruturas de bancos de dados, diagramas UML e códigos auxiliares.

Embora a ferramenta tenha sido desenvolvida para fins específicos, sua arquitetura baseada em *plugins* permite que esta seja estendida para atender a requisitos de outros domínios. O restante do artigo é organizado da seguinte maneira. A seção 2 apresenta uma contextualização com relação a engenharia reversa e geração automática de códigos. Na seção 3 é apresentada em detalhes a ferramenta GeCA. Finalmente, na seção 4 são apresentadas as conclusões e os trabalhos futuros.

2. Análise das Ferramentas Relacionadas

A fim de identificar algumas características importantes para a ferramenta, foram analisadas e estudadas algumas ferramentas que provêem engenharia reversa e geração automática de código conhecidas no meio acadêmico e no mercado. As ferramentas estudadas foram ArgoUML [2006], Poseidon [2006], Rational Rose [2006] , ARES [VERONESE *et al.,* 2002], FuJaba[2006] e Together [2006] e JQuerena [2006].

As características analisadas dizem respeito tanto à Engenharia Reversa de códigos Java quanto à geração automática de código. Estas características são apresentadas abaixo:

- *Engenharia Reversa de código Java*: Capacidade de a ferramenta extrair informações a partir de códigos *Java* (.java);
- *Engenharia Reversa de byte code Java*: Capacidade de a ferramenta extrair informações a partir de códigos *byte code* Java (.class);
- *Engenharia Reversa a partir de pacote Java*: Capacidade de a ferramenta extrair a estrutura de pacotes de um sistema, e as informações das classes pertencentes a este pacote;
- *Detecção e geração de associações entre classes*: Capacidade de a ferramenta detectar relacionamentos entre classes, bem como sua multiplicidade;
- *Criação automática de códigos auxiliares*: Possibilidade de gerar códigos de mapeamento, representação e tradução de dados de acordo com as classes geradas;
- *Manutenção automática de códigos:* Possibilidade de a ferramenta fornecer meios de atualização dos códigos fonte, modelo de banco de dados e códigos auxiliares, refletindo alterações dos modelos durante o ciclo de vida do sistema.
- *Geração de esquemas de banco de dados*: Possibilidade de a ferramenta gerar esquema de banco de dados em alguma linguagem (DDL – *Data Definition Languagem*);

Tabela 1. Tabela comparativa das características de Engenharia Reversa e Geração de Código

Características	*ArgoUML*	*Poseidon*	*Rose*	*ARES*	*FuJaba*	*Together*	*JQuerena*
Engenharia Reversa de código java (.java)	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**	Não	Não
Engenharia Reversa de *byte code* java (.class)	Não	Não	Não	Não	Não	**Sim (.jar)**	Não
Engenharia Reversa a partir de pacote java	parcial	**Sim**	**Sim**	**Sim**	Não	Não	Não
Engenharia Reversa a partir de Banco de Dados	Não	Não	Não	Não	Não	Não	**Sim**
Detecção e geração de associações entre classes	Não	**Sim**	parcial	**Sim**	parcial	**Sim**	Não
Criação automática de códigos auxiliares	Fontes	Fontes	Fontes	Não	Beans	Fontes	**Sim**
Manutenção automática de códigos	Não	Não	Não	Não	Não	Não	Não
Geração de esquemas de banco de dados	Não	Não	Não	Não	Não	**Sim**	Não

A tabela 1 apresenta o comparativo entre as ferramentas de acordo com as características acima explicitadas. Quando uma característica é atendida de maneira *parcial* significa que existem alguns casos não cobertos pela ferramenta em questão. Quando existem pontos específicos de uma característica cobertos pela ferramenta, estes são citados.

Pode-se perceber que as ferramentas apresentam características basicamente relacionadas à engenharia reversa, sem se preocupar com a manutenção de código. A engenharia reversa é realizada tendo como entrada apenas códigos fontes (.java), com exceção da ferramenta Together e do JQuerena. Assim, não são considerados códigos binários ou esquemas de bancos de dados como entrada, não havendo maneira de extrair informações de sistemas que estão sendo utilizados, em que o código fonte não é acessível. Destaca-se a ferramenta JQuerena por ser específica para criação de códigos auxiliares e de modelo a partir de Bancos de Dados pré-existentes, porém, esta peca pois não permite extensões, ficando restrita a isto.

3. A Ferramenta Proposta

A ferramenta denominada GeCA é parte integrante do DiSEN (Ambiente para Desenvolvimento Distribuído de Software) [PASCUTTI, 2002], que tem como objetivo fornecer métodos e ferramentas que auxiliem o desenvolvimento colaborativo de software.

Nesta seção serão apresentados em detalhes a arquitetura, os mecanismos de importação, transformação e o mecanismo de plugins. Primeiramente será apresentada a arquitetura e em seguida os componentes ali apresentados.

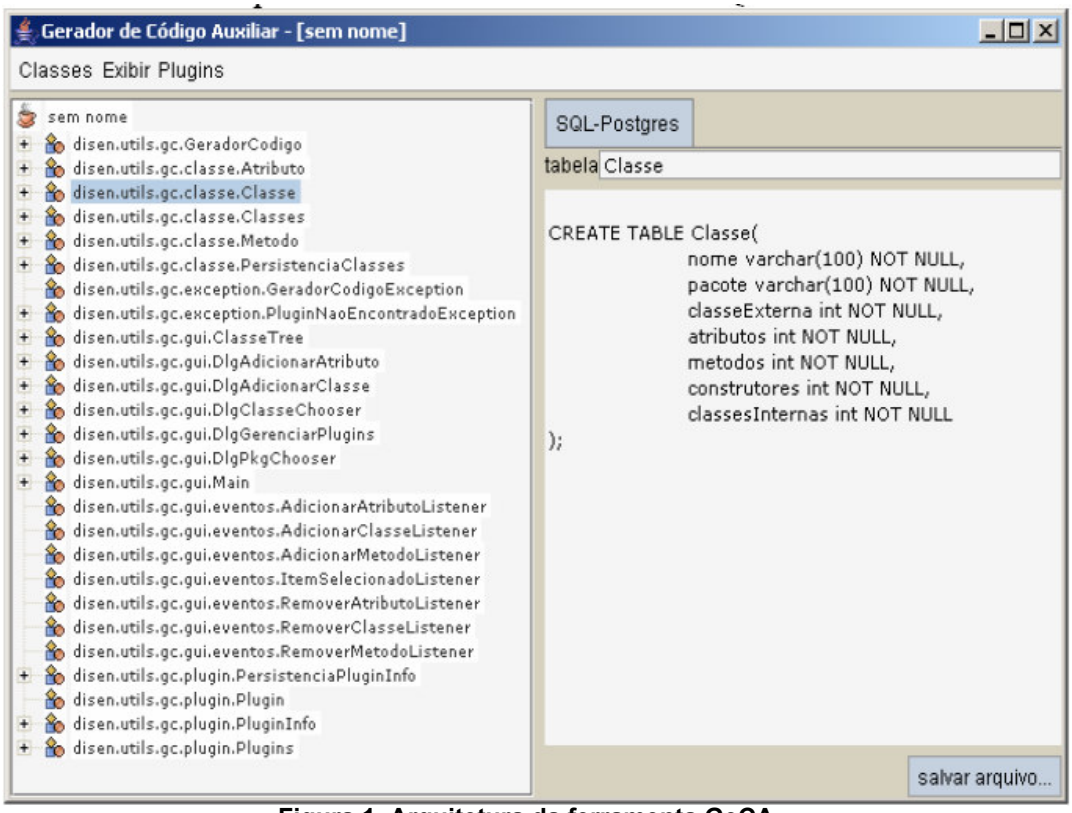


Figura 1. Arquitetura da ferramenta GeCA

3.1. Mecanismos de Obtenção de Dados

A arquitetura da ferramenta é apresentada na Figura 2 e contempla, além do componente de obtenção de dados, o componente principal da GeCA, a API para *plugins* e os *plugins* em si. Primeiramente, será detalhado o componente de obtenção dos dados de acordo com o tipo de entrada fornecida:

- **Banco de Dados**: neste caso a GeCA requisita ao SGBD os nomes de tabelas, nomes e tipos dos campos e relacionamentos entre tabelas (quando houverem) para gerar sua estrutura de classes. Tal aquisição pode ser realizada a partir de qualquer banco de dados que se tenha acesso e que tenha o *driver* correspondente no JDBC.
- **Classes ou Pacotes Java compilados**: quando se decide obter os dados por meio de classes Java compiladas (.class) o usuário pode solicitar a importação de uma classe específica ou ainda de um pacote todo. Caso seja selecionada a importação de uma classe específica, a GeCA apenas requisita o nome de classe, atributos (com seus tipos) e métodos e transforma-os em objetos. Caso seja selecionado um pacote, é feita a organização das classes dentro de pacotes, e a cada classe são atribuídos os atributos e métodos de cada uma.
- **Diagrama de Classes UML**: neste caso a ferramenta pode disponibilizar uma ferramenta própria de modelagem de classes, sendo um *plugin* que é instalado junto como padrão. Com este *plugin* o usuário pode definir classes com métodos e atributos e os relacionamentos entre elas. Esta ferramenta é também uma saída, pois, quando os dados são importados pelo ambiente estes são automaticamente representados em forma de diagrama de classes.

As duas primeiras formas de obtenção de dados apresentadas focam na Engenharia Reversa, tendo como entrada dados de sistemas já desenvolvidos. O uso destas entradas é um grande diferencial da presente ferramenta, uma vez que não se faz necessário possuir o código fonte do sistema para a realização da engenharia reversa. A importação a partir de bancos de dados faz com que o sistema possa gerar informações a partir de sistemas legados dos quais não se sabe a linguagem de programação utilizada na implementação ou cujo código fonte foi perdido.

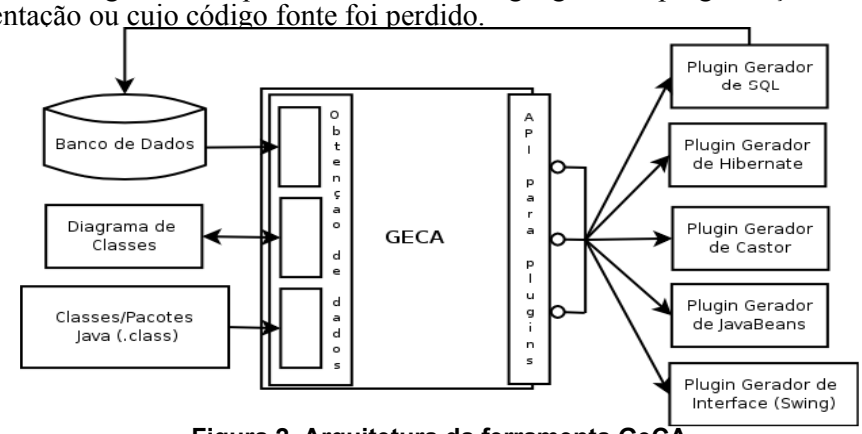


Figura 2. Arquitetura da ferramenta GeCA

A partir destes mecanismos a tarefa de extrair informações e criar documentação ou até mesmo re-desenvolver um sistema já existente se torna menos complexa e mais rápida. Quando da necessidade de adaptação de um sistema já funcional a uma nova realidade (linguagem diferente, nova plataforma), a ferramenta apresenta-se capaz de gerar diagramas de classe e entidade-relacionamento, gerar esqueleto do código fonte (no momento Java e PHP), fornecer códigos auxiliares e oferecer suporte nos casos de manutenção adaptativa, corretiva ou evolutiva.

3.2. Estrutura Interna

Como dito anteriormente, os dados obtidos são utilizados para gerar informações dentro da ferramenta. Estas informações são usadas para povoar classes representando a estrutura do sistema importado. Os metadados definidos para a GeCA são apresentados na Figura 3, e as classes relacionadas são descritas brevemente abaixo:

- A classe “Gerador Código” é a classe de controle da ferramenta. Ela é responsável por importar as classes e pacotes e transformá-los, de modo que possam ser armazenados nas demais classes ("Classe", "Atributo" e "Metodo"). Ela também é responsável pelo controle dos objetos destas classes e por informar a lista de *plugins* que a estrutura e/ou informações das classes foram alteradas e que os *plugins* devem refletir estas alterações.
- A classe "Classes" é responsável por manter a lista de classes do projeto, e as operações válidas nesta lista, isto é, inserção e remoção de classes.
- A classe “Classe” é responsável por armazenar as informações das classes, tornando possível reconstruí-la, se necessário. Estas informações se referem ao nome da classe, pacote a que pertence e métodos, construtores, atributos e classes internas que possui.
- A classe “Atributo” é uma abstração de um atributo de uma classe e possui como propriedades o nome do atributo, tipo do atributo e a classe a que este pertence.
- A classe “Método” é a abstração dos métodos de uma classe e, de maneira similar aos atributos, possui um nome, tipo de retorno, classe a que pertence e os parâmetros.
- A classe “Plugins” é responsável por manter a lista de *plugins* instalados na ferramenta e fazer a comunicação entre a ferramenta e cada um destes *plugins*.
- A classe “PluginInfo” armazena as informações sobre cada um dos *plugins* instalados. Estas informações incluem nome e versão do *plugin*, e nome da biblioteca (arquivo JAR) onde o *plugin* está localizado.
- A interface “Plugin” define os métodos que devem ser implementados por cada *plugin* que será adicionado a esta ferramenta. Esta interface será explorada na seção 3.3.

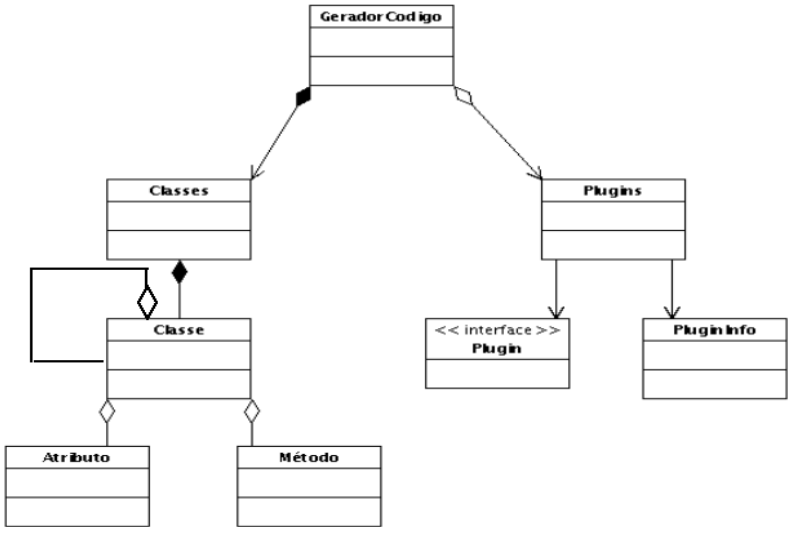


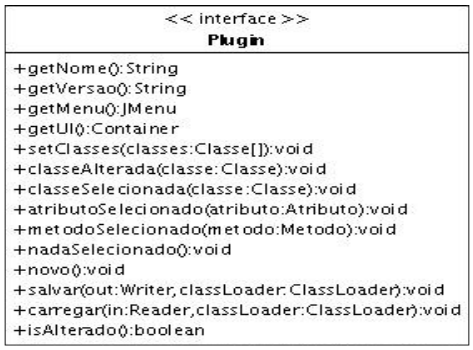
Figura 3. Diagrama de Classes para Armazenamento das Informações em Classes na GeCA

Depois de obtidas pela ferramenta, as informações são apresentadas de duas maneiras distintas para o usuário: (i) em forma de diagrama de classes UML; e (ii) em forma de árvore baseada nos pacotes e nas classes. Depois de geradas as duas formas de visualização das classes, o usuário pode interagir, alterando, inserindo ou apagando informações (classe, métodos e atributos). As alterações realizadas pelos usuários refletem em todos os módulos nativos e também nos códigos gerados pelos *plugins*. Isto quer dizer que, caso haja alguma alteração do diagrama de classes durante qualquer fase de desenvolvimento, não é necessário alterar manualmente o código do sistema e os códigos auxiliares utilizados, pois todos são atualizados pela ferramenta.

3.3. Utilização de *Plugins*

A ferraenta, como dito anteriormente, é baseada em *plugins*. Para tal fim definiu-se uma API (*Application Program Interface*) para permitir que os *plugins* utilizem as informações fornecidas pela ferramenta. Neste sentido a interface define métodos para manter um padrão de comportamento dos *plugins n*a ferramenta. A interface definida é apresentada na figura 4. Ela exige a implementação de 14 métodos para a definição de um *plugin*.

Para que um *plugin* comece a funcionar com a ferramenta é necessário que este seja instalado na mesma. Para isto, foi criado um mecanismo de importação, no qual o autor do *plugin* deve indicar um arquivo XML contendo o nome do *plugin*, o nome da classe que implementa a interface Plugin e o nome do arquivo *.jar* que contém as classes do *plugin*.




```
<plugin>
   <nome>Hibernate</nome>
   <main>disen.utils.hibernate.Hibernate</main>
   <jar>hibernate.jar</jar>
</plugin>
```

Figura 4. API para definição de *plugins* para a ferramenta e arquivo XML para instalação do *plugin*

Atualmente a GeCA possui implementados os mecanismos de obtenção de dados, a geração de estrutura interna e alguns *plugins*. Dentre estes, está o *plugin* de geração de SQL, que é nativo da ferramenta, pois é utilizado na criação automática do banco de dados no SGBD desejado. Os demais *plugins* em funcionamento são: gerador de Hibernate, gerador de *script* de mapeamento Castor e Gerador de JavaBeans. Os *plugins* de geração de Interface Gráfica, código PHP e formulários HTML a partir das informações extraídas estão em desenvolvimento.

3.4. Avaliação das propriedades da GeCA

A tabela 2 apresenta a análise das características relacionadas com engenharia reversa e geração automática de código na ferramenta GeCA.

Tabela 2. Tabela das características de Engenharia Reversa e Geração de Código para a GeCA

Características	*GeCA*
Engenharia Reversa de código java (.java)	Não
Engenharia Reversa de *byte code* java (.class)	**Sim**
Engenharia Reversa a partir de pacote java	**Sim**
Engenharia Reversa a partir de Banco de Dados	**Sim**
Detecção e geração de associações entre classes	parcial (1 para 1)
Criação automática de códigos auxiliares	**Sim**
Manutenção automática de códigos	**Sim**
Geração de esquemas de banco de dados	**Sim**

As características apresentadas na tabela foram discutidas nas subseções anteriores e são apenas destacadas aqui. O mecanismo de entrada para realização de engenharia reversa executada pela GeCA provê suporte a código binário java (arquivos *.class*), pacotes de arquivos *.class* e bancos de dados. A saída se dá na forma de diagrama de classes UML, sendo extraídas as associações 1 para 1. A ferramenta ainda gera códigos fontes e códigos auxiliares (XML, SQL, formulários) a partir da engenharia reversa. Por último, a ferramenta se baseia em *plugins*, sendo simples adequá-la às necessidades de qualquer projeto de qualquer domínio.

4. Conclusões

A ferramenta apresentada realiza engenharia reversa de códigos *byte code* java (arquivos de extensão *.class*), de classes individuais ou de pacotes, além de estruturas de bancos de dados. A ferramenta perde ao buscar informações destas fontes uma vez que se torna complexo encontrar dependências entre classes (atualmente são extraídas associações 1 para 1), bem como análise de estrutura dos métodos. Porém a ferramenta ganha ao permitir a engenharia reversa de sistemas em uso, cujo código fonte não está disponível e que necessita ser migrado para outra plataforma.

A GeCA, é capaz de realizar engenharia reversa produzindo classes UML referentes a qualquer classe java ou esquema de banco de dados. Como o processo de engenharia reversa não utiliza código fonte, e sim binários e bancos de dados, considera-se que um outro produto deste processo é a geração do código fonte e códigos de mapeamento e armazenamento.

Com relação à manutenção de sistemas, a GeCA é capaz de gerar código fonte e esquema de bancos de dados automaticamente, a partir de alterações no diagrama de classes ou na estrutura de classes. Esta automatização em parte do processo de manutenção tem diminuido o tempo necessário com alteração de código fonte, reduzindo, assim, os gastos com este processo.

No momento estão sendo desenvolvidas algumas novas funcionalidades para a ferramenta. Uma delas é a geração automática de formulários *Web* com código PHP e JSP. Estão sendo realizadas avaliações com relação ao tempo despendido com a manutenção de código comparando situações com e sem o uso da ferramenta de manutenção.

4. Referências

ARGOUML. ArgoUML, disponível em <http://argouml.tigris.com> . Acesso: Setembro, 2006.

BAXTER, I.; MEHLICH, M. Reverse Engineering is Reverse Forward Engineering. Working Conference on Reverse Engineering, 4., Amsterdam, 1997. **Proceedings…** 1997

FUJABA. FuJaba, disponível em <http://www.fujaba.de>. Acesso em: Setembro, 2006.

PASCUTTI, M.C., **Uma proposta de arquitetura de um ambiente de desenvolvimento de software distribuído baseado em agentes.** 2002, 102 p. Dissertação (Mestrado) - Universidade Federal do Rio Grande do Sul, Porto Alegre, 2002.

POSEIDON. Poseidon for UML, disponível em <http://www.gentleware.com/>. Acesso em: Setembro, 2006.

PRESSMAN, R. **Software Engineering**: A Practitioner's Approach. McGraw Hill, 2002.

QUINAIA, M.; STADZISZ, P. Identificação de Padrões Arquiteturais Usando Engenharia Reversa. Workshop de Manutenção de Software Moderna, 2004, Brasília. **Anais...** 2004.

RATIONAL ROSE. Rational Rose, disponível em <http://www-306.ibm.com/software/rational>. Acesso em: Setembro, 2006.

TOGETHER. Together, disponível em <http://www.togethersoft.com >. Acesso: Setembro, 2006.

VERONESE, G.; NETTO, F.; WERNER, C.; CORREA, L. Uma Ferramenta de Auxílio a Recuperação de Modelos UML de Projeto a Partir de Código Java. **Revista Eletrônica de Iniciação Científica**. [S.l.], v. 2, n. 4, Dez. 2002.