Uma Arquitetura Aberta para Geradores de Artefatos

Luiz Paulo Alves Franca
Programare Informática
luizp@programare.com.br

Arndt von Staa
Departamento de Informática, PUC Rio
arndt@inf.puc-rio.br

Resumo

A utilização de geradores de artefatos no processo de desenvolvimento de software proporciona ganhos de escala. Porém a adoção desta tecnologia está restrita a um pequeno número de organizações. A justificativa para esta realidade, além dos aspectos sociais e culturais da adoção de uma nova tecnologia, deve-se a complexidade na construção deste tipo de ferramenta. O presente trabalho descreve uma proposta de arquitetura aberta para geradores de artefatos resultante de pesquisas feitas nos últimos anos sobre simplificação do processo de construção de geradores. Na arquitetura proposta, cada módulo do gerador, é implementado independentemente dos demais módulos através de ferramentas de ampla utilização industrial. A arquitetura proposta foi implementada utilizando uma ferramenta CASE, um banco de dados relacional, e a linguagem de *template* ASP.

Palavras-Chaves

Geradores de Artefatos, Programação generativa, linha de produção de software, CASE, meta-geradores, desenvolvimento para WEB, XML

Abstract

Although the use of artifact generators can lead to substantial gains when developing or maintaining software, this technology has received little attention and is used only by few organizations. Besides the traditional social and cultural barriers of adopting new technologies, one of the possible explanations is the development complexity of such tools. This article describes an architecture proposal based on lessons learned during the years we researched the development process of such tools. The proposed architecture employs only widely used tools to implement the components of an artifact generator. This architecture has been implemented using commercially available tools, such as a CASE tool, a relational database management system and the ASP template language.

Keywords

Artifact generators, Generative programming, Software-product line, CASE, Web oriented development, XML

1. Introdução

Recentemente assuntos relacionados com a utilização e construção de geradores de artefatos[1] passaram a ser discutidos em diferentes contextos da área da computação. Trabalhos sobre *Generative Programming* [2] apresentam diversas abordagens para construção de geradores. Já na área de *Software Product-line* [3,4], o gerador é apresentado como uma importante ferramenta para implementação de uma linha de produtos de software. O desenvolvimento para Web, onde técnicas de geração são utilizadas para a produção de páginas dinâmicas, também contribuiu para que o tema geradores passasse a ser discutido por uma maior número de desenvolvedores. Nas propostas de desenvolvimento incremental apresentadas dentro da abordagem da *Extreme Programming* [5], o gerador de artefatos viabiliza a construção de protótipos.

Caso uma organização resolva adotar geradores em seu processo de desenvolvimento de software, os seguintes benefícios são esperados:

- **Aumento de produtividade.** Na maioria dos geradores, a maior parte de um artefato gerado corresponde a trechos fixos referentes a detalhes de implementação. A geração automática destas partes fixas proporciona uma elevação da produtividade da equipe de desenvolvimento.

- **Redução do tempo para o mercado.** O tempo de desenvolvimento de um novo artefato tende a ser reduzido ao tempo gasto no desenvolvimento e avaliação da sua especificação.

- **Prototipação.** Para criar novas versões de um artefato basta ao desenvolvedor alterar a especificação fornecida ao gerador. Este baixo custo de criação de novas versões, permite ao desenvolvedor conduzir diversos experimentos para determinar o artefato mais adequado ao usuário.

- **Qualidade do artefato gerado.** Quando da utilização de um gerador pode-se pressupor que esteja correto, bem como que o artefato gerado também esteja correto por construção. Caso o artefato gerado tenha problemas, estes estarão associados a problemas na forma de gerá-lo.

Apesar dos ganhos em ordem de grandeza proporcionados pela adoção de geradores, poucas organizações se beneficiam desta técnica. Esta baixa adoção pode ser explicada pela complexidade envolvida na construção de geradores. Vale ressaltar que, como qualquer outra nova tecnologia, após os aspectos técnicos serem dominados, os aspectos gerenciais e culturais devem ser considerados para garantir o sucesso na introdução de geradores no processo de desenvolvimento da organização.

---

[1] *"Um artefato é qualquer item criado como parte da definição, manutenção ou utilização de um processo de software. Inclui entre outros, descrições de processo, planos, procedimentos, especificações, projeto de arquitetura, projeto detalhado, código, documentação para o usuário. Artefatos podem ou não ser entregues a um cliente ou usuário final". [1]*

No sentido de suplantar a barreira técnica na adoção da tecnologia de geradores, desenvolvemos uma proposta de arquitetura aberta para geradores de artefatos, que permite a construção de geradores por um maior número de organizações.

Este artigo apresenta a arquitetura aberta para geradores de artefatos da seguinte forma: a seção 2 apresenta um arquitetura padrão para geradores; a seção 3 apresenta as características de um ambiente de construção de geradores; a seção 4 descreve a arquitetura de geradores baseados em ferramentas CASE; a seção 5 descreve a arquitetura aberta para geradores de artefatos; a seção 6 apresenta uma comparação com os trabalhos relacionados. Na seção 7, são apresentadas as conclusões e os trabalhos futuros.

2. Arquitetura Padrão de um gerador de artefato

A Figura 1 sintetiza a arquitetura padrão apresentada na literatura [6, 7, 8]. Os componentes básicos da arquitetura são: o analisador de especificação e o gerador de artefato.

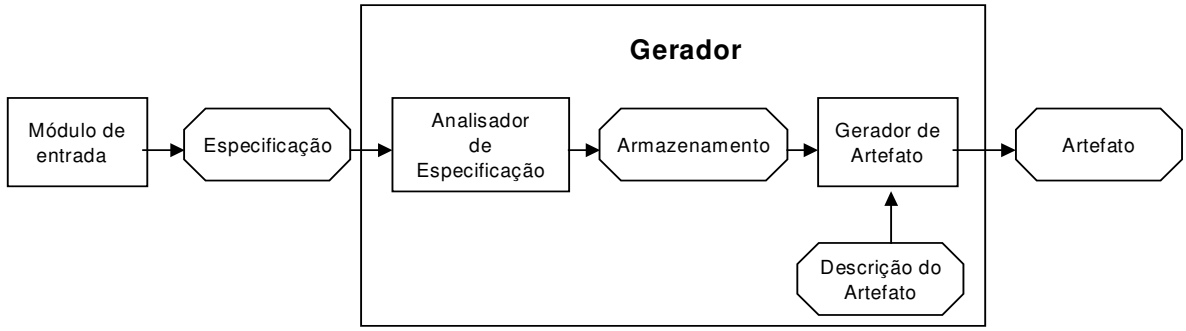

Figura 1 Arquitetura padrão de um gerador de artefatos

O analisador de especificação (ou *front-end*) é o módulo responsável por transformar uma especificação redigida numa linguagem "X" para uma representação interna mais favorável para o processo de geração. O analisador de especificação implementa as funções de um analisador léxico e de um *parser*. O formato da especificação é geralmente textual, hipertexto ou diagramático. Já a representação interna pode ser uma *abstract syntax tree (AST)*. Observa-se que a complexidade deste módulo está diretamente ligada às diferenças entre os formatos da especificação e da representação interna: quanto mais diferente, mais complexo será o módulo. Por outro lado, uma especificação que utilize diretamente a linguagem da representação interna pode apresentar problemas de legibilidade, pois esta linguagem pode ter pouca expressividade em relação ao domínio do problema.

Alguns geradores possuem um módulo anterior ao analisador de especificação, utilizado para capturar a especificação, denominado módulo de interface. Através deste módulo o usuário fornece a especificação, preenchendo formulários, respondendo a uma seqüência de diálogos (*wizard*) ou desenhando diagramas.

O gerador de artefato (*back-end*) transforma a especificação armazenada na representação interna nos componentes do artefato-alvo. Esta transformação pode ser executada em 2 etapas. A primeira etapa consiste em gerar um artefato intermediário descrito através da linguagem de representação interna. A etapa complementar realiza o mapeamento entre a linguagem

de representação interna e as linguagens do artefato final e que pode ser formados por diversos programas ou módulos codificados em diferentes linguagens de programação.

3. Ambientes de Construção de Geradores de artefato

A maioria dos geradores analisam uma especificação para fazer a geração do artefato [8, 9, 10]. Para construir um gerador que utilize esta abordagem, o desenvolvedor precisa definir uma linguagem de especificação, um analisador para esta linguagem, e desenvolver um transformador que converta a árvore de sintaxe abstrata resultante no artefato a ser gerado. Estas atividades requerem conhecimentos de linguagens formais e de construção de compiladores que não são de domínio da maioria das organizações. Por causa disto, resolvemos adotar uma outra abordagem de geração.

Para valer à pena o seu emprego, o custo de desenvolver e manter um gerador de artefatos deve ser superado pela economia obtida através da redução do custo de construção e manutenção dos artefatos gerados [11]. Esta meta pode não ser alcançada através do processo tradicional de desenvolvimento de software. Em [7], é apresentada uma lista com as características desejáveis para um ambiente de geração:

Possibilidade de definição de novas linguagens de especificação. O ambiente de geração deve poder produzir geradores que aceitem especificações tanto na forma textual como na de diagramas. O ambiente deve possuir mecanismos que permitam a alteração das linguagens de especificação disponíveis.

Estrutura interna genérica. A estrutura interna deve permitir o armazenamento de diferentes tipos de especificações.

Ampla variedade de rotinas de acesso à estrutura interna. Para facilitar a codificação de transformadores, o ambiente deve disponibilizar diferentes rotinas de acesso à sua estrutura interna.

Possibilidade de definição de novos artefatos. O ambiente de geração deve possuir mecanismos que agilizem a definição de novos artefatos.

Confrontando a lista de requisitos com as características de uma ferramenta CASE (*Computer-Aided Software Engineering*), observarmos que o papel de ambiente de geração pode ser exercido por um CASE que tenha facilidades de customização. Em [12, 13, 14, 15, 17] são descritos trabalhos nos quais ferramentas CASE são utilizadas como ambiente de geração. Em [12] é apresentada uma lista das características que uma ferramenta CASE deve ter para ser utilizada como ambiente de geração:

- Possuir um repositório contendo todos os dados dos modelos que estão sendo utilizados;
- Disponibilizar interfaces que permitam amplo acesso aos meta-dados do repositório;
- Possuir mecanismos para editar os meta-modelos disponíveis, adicionando ou modificando seus meta-dados;
- Permitir a criação e a execução de *scripts* que exploram e manipulam o conteúdo do repositório.

4. Arquitetura de Geradores de artefato baseados em ferramentas CASE

Por possuir as características desejáveis para um ambiente de geração e pela experiência acumulada na sua utilização, adotamos o CASE Talisman [16] como ambiente de construção do nosso primeiro gerador de artefatos. Ressaltamos que qualquer CASE que possua as características acima relacionadas pode ser utilizado com ambiente de geração. Em [12,13, 17] é relatada a utilização do CASE Rational Rose [18] como ambiente de geração. A figura 2 descreve a arquitetura de um gerador baseado em ferramentas CASE.

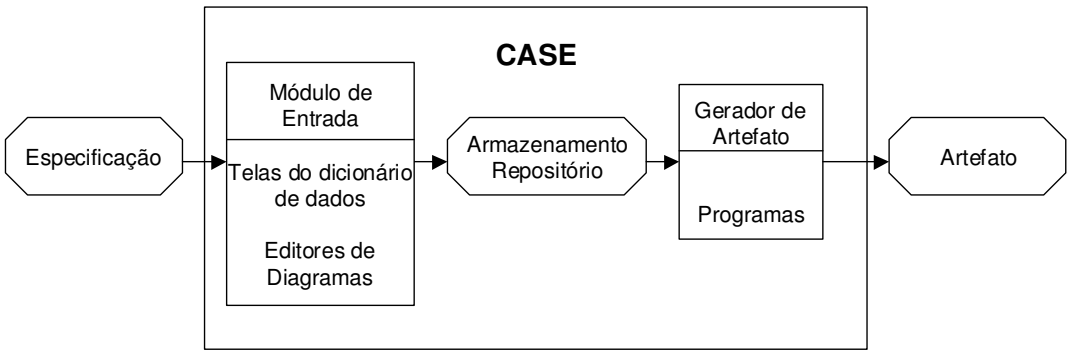

Figura 2. Arquitetura de um gerador de artefatos baseado em ferramenta CASE

Módulo de entrada é implementado através das telas do dicionários de dados e dos editores de diagramas da ferramenta CASE.

Módulo de armazenamento é implementando através do repositório do CASE.

Módulo de geração é implementado através de um conjunto de programas que extraem dados do repositório e produzem a saída no formato do artefato a ser gerado. Estes programas são codificados pelo desenvolvedor utilizando a linguagem de programação disponibilizada pelo CASE (Ex: Na ferramenta Rational Rose[18], o Visual Basic é a linguagem de programação do ambiente).

Construir geradores utilizando um CASE envolve as seguintes atividades [19, 20]:

- Definição do módulo de entrada: É preciso definir o conjunto de telas e diagramas do CASE que irão compor o gerador. Além disto, em função do tipo do artefato a ser gerado, pode ser necessário o acréscimo de novos campos nas telas do dicionário de dados, ou mesmo a utilização de novas notações diagramáticas. (Fig. 3)

- Extensão do meta-modelo: Caso o módulo de entrada precise capturar dados ainda não dicionarizados, o meta-modelo do CASE deve ser estendido para acomodar as novas informações (Fig. 3).

- Codificação do módulo de geração: O artefato gerado é produzido por programas que compõem as partes fixas do artefato com as partes variáveis extraídas da especificação armazenada no repositório da ferramenta através de funções implementadas pela linguagem de programação do CASE(Fig 4).

```
Sub Main
  Dim prop As DefaultModelProperties
  Set prop =
RoseApp.CurrentModel.DefaultProperties
  b = prop.AddDefaultProperty("Attribute",
"EXTRA",
       "default","Titulo_do_Campo",
"String", "")
  b = prop.AddDefaultProperty("Attribute",
"EXTRA",
      "default","Descreve", "String", "")
  b = prop.AddDefaultProperty("Attribute",
"EXTRA",
         "default","Obrigatório", "String", "")
  b = prop.AddDefaultProperty("Attribute", "EXTRA",
      "default","URL_Browser", "String", "")
  b = prop.AddDefaultProperty("Attribute", "EXTRA",
      "default","URL_Link", "String", "")
```

Figura .3. Exemplo de extensão do repositório e do módulo de entrada da ferramenta CASE Rational Rose

```
Print "<HTML><BODY>"

Print "<H1>" + JanelaTitulo() + "</H1>"

Print "<b>" + CampoTitulo() + "</b>"

Print "<INPUT TYPE = 'text'  " + CampoNome() + " maxlength= "+ Campo-
Tamanho() + ">"

Print "</HTML></BODY>"
```

Figura .4. Exemplo de trecho de código do módulo de geração

Para validarmos a utilização do CASE como ambientes de geração, construímos um gerador de aplicações a partir de modelo de dados [21]. A aplicação gerada é capaz de fazer as operações básicas de manutenção de tabelas, respeitando os diferentes tipos de relações definidos no modelo (1:N, N:N, auto-relacionamento, herança simples). As seguintes versões do gerador foram construídas (Figura 5):

- Gerador de aplicações em 3 camadas: Camada de interface (HTML+javascript), camada de negócio (Java Servlet), e camada de BD (JDBC).
- Gerador de aplicações cliente-servidor: Visual Basic, Delphi, e Centura.

- Gerador de documentação em HTML
- Utilização de diferentes ferramentas CASE: Talisman e Rational Rose.

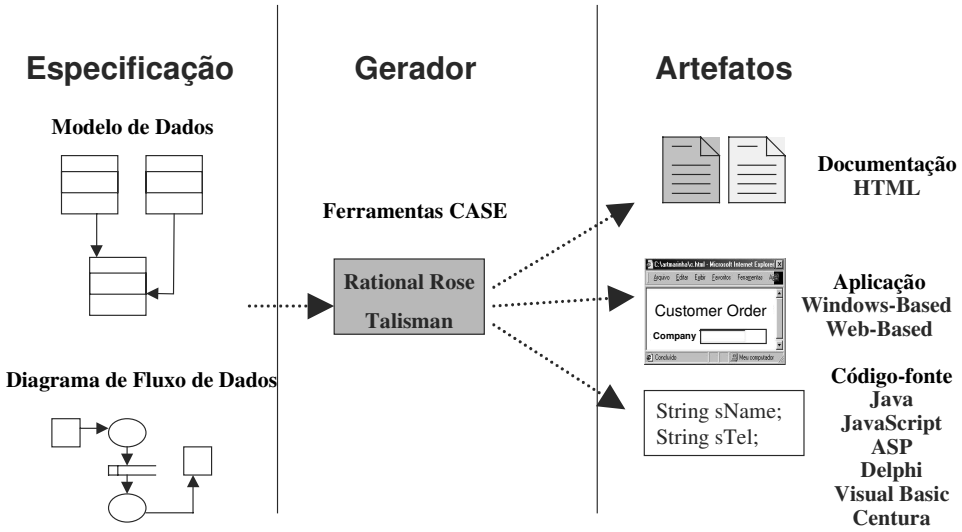

Figura .5. Experimentos de construção de geradores baseados em ferramenta CASE

Os experimentos realizados confirmaram a viabilidade em utilizar-se ferramentas CASE como ambiente de geração, porém foram identificadas as seguintes restrições:

- Limitações do ambiente de programação da ferramenta CASE: Nos experimentos realizados, em ambas ferramentas utilizadas, encontramos limitações quanto ao tamanho do programa de geração e nas facilidades de codificação.

- Baixo reúso entre as ferramentas CASE: Quando a versão do gerador utilizando o CASE Rational Rose foi criado, toda a biblioteca de acesso criada para o CASE Talisman teve que ser re-codificada.

- Dependência da ferramenta CASE: Geradores construídos podem deixar de funcionar em versões mais novas da ferramenta CASE, em função de alterações nos módulos que compõem o gerador.

Para superar as restrições encontradas, decidimos adotar uma arquitetura aberta para construção de geradores.

5. Arquitetura Aberta de Geradores de Artefato

A definição da nova proposta de arquitetura considerou as seguintes premissas:

- Atender os requisitos de um ambiente de geração (seção 3)
- Reduzir o impacto na substituição das ferramentas utilizadas na construção do gerador.
- Minimizar a dependência das ferramentas que compõem o gerador.
- A equipe de desenvolvimento da organização deve ser capaz de construir o gerador, dispensando a participação de especialistas em linguagens formais ou compiladores.

A Figura 6 sintetiza a arquitetura aberta para geradores de artefato.

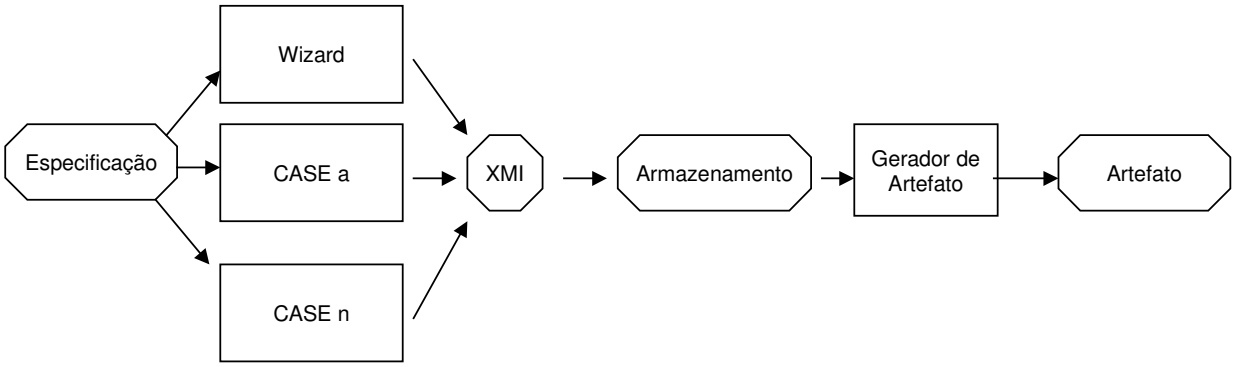

Figura .6. Arquitetura aberta para geradores de artefatos

Os módulos que compõem o gerador passam a ser independentes não fazendo parte da mesma ferramenta.

Módulo de entrada: O módulo de entrada deve ser implementado através de qualquer ferramenta disponível na organização para captura de especificações. Este módulo deve possuir mecanismo de exportação da especificação capturada. Trabalhos recentes [23] apontam a utilização do padrão XMI para o intercâmbio de informações entre ferramentas de desenvolvimento.

Módulo de armazenamento: O módulo de armazenamento deve ser implementado em qualquer ferramenta que possua as características de um repositório de dados. Este módulo deve possuir um mecanismo de importação de especificações. A adoção do padrão XMI permite que apenas uma única interface de importação seja implementada. Neighbors em [24] faz uma série de considerações sobre cuidados na adoção de padrões da indústria para construção de ambientes de geração.

Módulo de geração: O módulo de geração deve ser implementado em linguagens de programação de ampla utilização que possuam manipular o módulo de armazenamento. É interessante que a linguagem escolhida possua mecanismos para tratamento das partes fixas e partes variáveis do artefato a ser gerado.

Figura .7. Exemplo de implementação da arquitetura aberta

Baseado na arquitetura aberta proposta, foi construído uma versão industrial de um gerador de modelo de dados para ser utilizado na automação do desenvolvimento de uma pequena software-house. Esta implementação corresponde a seguinte instância da arquitetura proposta (Fig 7):

- Módulo de entrada: Utilizamos como editor de diagramas o Visual Modeler do Visual Studio da Microsoft. Foi desenvolvido um módulo de exportação da especificação pro-

duzida. Este módulo produz um arquivo XML contendo as informações necessárias para geração a partir da especificação armazenada no arquivo no formato MDL. Foi desenvolvido um módulo de entrada de dados para complementar a especificação feita pelo editor de diagramas. Em [25] é descrita uma proposta de editores de diagramas baseados em XML e independentes de ferramentas de desenvolvimento

- Módulo de armazenamento: Utilizamos o MS SqlServer 7.0 como repositório de dados. Foi desenvolvido um módulo de importação em ASP (*Active Server Page*) que lê o arquivo XML exportado pelo módulo de entrada e atualiza as tabelas da especificação.

- Módulo de geração: Utilizamos as linguagens ASP e VB Script para implementar os programas geradores. O ASP possui mecanismos de tratamento de partes fixas e partes dinâmicas que são utilizados amplamente na construção de aplicações para Web, facilitando a codificação dos programas de geração. Já o VBScript possui interfaces bem definidas para acesso ao banco de dados. A geração é disparada através de um *browser* que faz a solicitação da página ASP de geração.

6. Trabalhos Relacionados

Embora na literatura existam inúmeras referências a construção de geradores, a maioria delas refere-se a sistemas transformacionais [2, 6, 7, 8, 10], que dependem de equipe altamente especializada para sua construção. Procuramos centrar nossa pesquisa bibliográfica em trabalhos que visam simplificar a construção de geradores. Destacamos os seguintes trabalhos relacionados:

Cleaveland em [9] descreve de forma bastante objetiva o processo de construção de geradores de aplicação adotado no Bell Labs. Neste trabalho, é chamada a atenção na necessidade de utilização de ferramentas que auxiliem o processo de construção. No Bell Labs, a construção de geradores através do meta-gerador Stage trouxe um redução significativa no esforço de construção. Porém, o próprio autor assinala que a principal dificuldade encontrada nesta abordagem é o projeto e a especificação da linguagem de especificação:

> *"A difficult step in the Stage method is the design and specification of the specification language. People who build generators for the first time sometimes have little or no experience with formal languages, language design, grammars, or YACC jargon. We are developing new techniques and tools that provide dialogue-oriented rather than language-oriented generators to solve this problem." [9]*

Mais recentemente, Cleaveland publicou um livro [26], onde a principal motivação é levar a técnica de construção de geradores para um maior número de desenvolvedores. A ênfase maior do livro é no módulo de geração, onde são apresentadas diferentes alternativas de implementação utilizando Java, XML, XSLT, JSP. Comparando com a nossa proposta, a maior similaridade está no módulo de geração em que ambas abordagens utilizam linguagens orientadas a *template*. Porém enquanto o módulo de geração do Cleaveland extrai informações de uma especificação em XML, em nossa abordagem a extração é feita no repositório de dados implementado no banco de dados relacional. Já em relação aos outros módulos, Cleaveland apenas trata-os superficialmente, indicando apenas que o módulo de geração tem como input uma especificação XML proveniente dos módulos de entrada e de armazenamento.

Em um artigo publicado em 1987 [27], Bassett utilizou o termo *Frame* para descrever sua proposta de reúso adaptativo. Um meta-gerador baseado na tecnologia de *Frames* é simplesmente um pré-processador independente de linguagem que constrói módulos de software através da adaptação de componentes generalizados chamados de *Frames [28]*. Esta abordagem foi utilizada industrialmente proporcionando significativos ganhos de produtividade (70% de redução de custo de pessoal, 84% de redução do time-to-market [29]). Em relação a proposta de Basset, o Prof. Don Batory, que publicou um grande número de trabalhos sobre sistemas transformacionais, fez os seguintes comentários:

> *"Bassett's frames are a generative technique for adapting code text through pure lexical manipulation. Despite their simplicity, frames have been used with great success to create programs of significant size (e.g., million-line) in the information systems domain ". [7]*

Apesar dos experiências relatadas serem de geração de sistemas em COBOL, a tecnologia adotada para implementar o módulo de geração do gerador é bastante similar a implementação utilizando linguagens orientadas a *template*. No trabalho de Basset, a linguagem de *templates* possui comandos de *meta-template*, por enquanto nos experimentos por nós realizados, ainda não encontramos a necessidade deste tipo de comando, sendo o nível de *template* suficiente para tratar as partes fixas e variáveis do artefato a ser gerado. Já em relação ao módulo de armazenamento, enquanto em nossa proposta, o armazenamento é feito num módulo separado, na proposta de Basset, não há uma separação entre módulo de armazenamento e de geração.

Em [30] é relatada a experiência em construção de sistemas utilizando ambiente de desenvolvimento composto por módulos independentes integrados através de um repositório de dados central. Geradores de código são escritos para extrair informações do repositório. O trabalho descreve experimentos onde a utilização desta abordagem permitiu que os seguintes índices fossem alcançados: 98% de geração de código de interface e de Banco de dados, redução do tempo de especificação em 60%, e redução do tempo de implementação em 50%. Em relação a este trabalho, nossa proposta apresenta bastante similaridades nos módulos de entrada e de armazenamento do gerador, onde ambas ressaltam a importância do repositório de dados. Já em relação ao módulo de geração, o trabalho não descreve a forma que este módulo é implementado. Em relação a comunicação entre os componentes do gerador, o trabalho ressalta a importância de padronização das interfaces envolvidas. Em função da época deste trabalho, provavelmente a interface ainda não era implementada através de arquivos em XML, uma vez que ainda não existia o padrão XMI para integração de ferramentas de terceiros.

Em [17] é apresentada a experiência industrial de utilização de geradores baseados em ferramentas UML integradas com técnicas de geração de código baseadas em *templates*. O trabalho descreve a utilização da ferramenta CASE Rational Rose para implementar os módulos de especificação e de armazenamento do gerador. Já o módulo de geração é implementado através da ferramenta *Codagen Architect* que disponibiliza um ambiente de programação para linguagens de *templates*. Em relação a nossa proposta, de forma similar, este trabalho também propõe a implementação do módulo de entrada através de ferramentas CASE. Porém o módulo de armazenamento também é implementado na ferramenta CASE. Já em relação ao módulo de geração, ambas as propostas adotam linguagens de *template*, porém enquanto nos-

sa proposta procura utilizar linguagens de larga utilização, este trabalho utiliza a linguagem disponibilizada pela ferramenta Codagen Architect.

Finalmente, em [31] é descrita a ferramenta MVCASE para especificação de requisitos de sistemas com múltiplas visões e implementação automática orientada a objetos. Neste ambiente, o módulo de entrada do gerador é a ferramenta CASE, o módulo de armazenamento guarda as especificações em linguagens definidas pelo sistema transformacional Draco [10] que implementa o módulo de geração. O ambiente é integrado a ferramenta Visual Café dbDE que permite o desenvolvedor construir as interfaces gráficas do sistema. Em relação a nossa proposta, este trabalho também apresenta a preocupação em separar o gerador em componentes. As principais diferenças entre as propostas são: Neste trabalho existe uma maior dependência entre os módulos de armazenamento e de geração; e o módulo de geração é implementado numa máquina transformacional.

7. Conclusões e Trabalhos futuros

A utilização de uma arquitetura aberta para implementação de geradores, proporciona os mesmos benefícios da adoção de uma arquitetura aberta já experimentados em outros campos da computação. A preocupação do mercado, em criar o XMI como padrão de integração de ferramentas de desenvolvimento, confirma que a decisão por uma arquitetura aberta é uma tendência natural para construção de geradores.

A experiência em construir geradores, nos mostraram que os ganhos imediatos proporcionados pelas facilidades das ferramentas utilizadas, são reduzidos na medida que o tempo de vida do gerador se estende. A excessiva utilização das facilidades das ferramentas CASE para construir o gerador, torna o gerador dependente de uma tecnologia fornecida por um único fornecedor.

Comparando o esforço na construção deste gerador, com as nossas experiências anteriores de construção de geradores baseados em ferramentas CASE, observa-se um ganho de produtividade na construção do módulo de geração. Este ganho decorre da adoção de linguagem de *templates* de ampla utilização (Ex: ASP) e da analogia que a equipe de desenvolvimento fez entre construção de geradores e desenvolvimento de *web sites*, onde em ambos os casos as partes dinâmicas são extraídas de um banco de dados.

Em relação a primeira versão do gerador baseada na arquitetura aberta, as seguintes evoluções já foram planejadas:

- Colocar os arquivos do módulo de geração sob o controle de um pacote comercial de controle de versões.

- Criar tabelas e acrescentar colunas ao meta-modelo do repositório de dados para permitir o controle de versão da especificação fornecida ao gerador. Serão criados geradores que produzirão programas para automatizar a evolução do esquema de banco de dados em função das diferenças entre as versões da especificação.

- Fazer uma nova versão utilizando apenas componentes do tipo open-software. Módulo de entrada: Argo UML, Módulo de armazenamento: MySQL, e módulo de geração: JSP + java.

- Desenvolver a interface de importação e exportação de arquivos no formato XMI.

8. Referências Bibliográficas

[1] Staa, A.; *Programação Modular;* Rio de Janeiro; Campus; 2000 1996

[2] Czarnecki, K.; Eisenecker, U.; *Generative Programming: Methods, Tools, and Applications*; Addison Wesley; NJ; 2000

[3] Clements, P; Northrop, L.; *Software Product Lines: Practices and Patterns*; Addison Wesley; NJ; 2002

[4] Weiss, D.; Lai, C.; *Software Product-Line Engineering: A Family-Based Software Development Process*; Addison Wesley; NJ; 1999

[5] Wake, W.; *Extreme Prgramming Explored*; Addison Wesley; NJ; 2001

[6] Balzer, R.; "A Fifteen-Year Perspective on Automatic Programming"; In Ted J. Biggerstaff and Alan Perlis (Eds.); *Software Reusability*; Addison-Wesley/ACM Press; 1989; pp 289-311.

[7] Batory, D.; Smaragdakis, Y.; "Application Generators"; *Wiley Encyclopedia of Electrical and Electronics Engineering – Software Engineering Volume;* John Wiley and Sons; 1999

[8] Masiero, P.C.; Meira, C. A.; "Development and Instantiation of a Generic Application Generator"; *J. Systems and Software;* 23; 1993; pp 27-37.

[9] Cleaveland, C.; "Building Application Generators"; *IEEE Software* 5(4); 1988; pp 25-33

[10] Neighbors, J. M.; "Draco: A Method for Engineering Reusable Software Systems"; In Ted J. Biggerstaff and Alan Perlis (Eds.); *Software Reusability*; Addison-Wesley/ACM Press; 1989; pp 295-319

[11] Levy, L.; "A Metaprogramming Method and Its Economic Justification"; *IEEE Transactions on Software Engineering 12(2);* 1986; pp 272-277

[12] Hohenstein, U.; "An Approach for Generating Object-Oriented Interfaces for Relational Databases"; In *Proceedings of the Second International Symposium on Constructing Software Engineering Tools*; 2000; pp 101-111.

[13] Milicev, D.; "Extended Object Diagrams for Transformational Specifications in Modeling Environments"; In *Proceedings of the Second International Symposium on Constructing Software Engineering Tools*; 2000; pp. 121-131.

[14] Plantec, A.; Ribaud, V.; "Using and Re-using Application Genetarors"; *In Proceedings of the First International Symposium on Constructing Software Engineering Tools*; 1999; pp 59-65.

[15] Emde-Boas, G. v.; "Architecture Based Code Generation from UML Models"; *OOPSLA'2001 Workshop on Generative Programming*; 2001; URL http://www.generative-programming.org/oopsla01-workshop.html

[16] Staa, A.; Manual de Referência: Talisman: Ambiente de Engenharia de Software Assistido por Computador; Rio de Janeiro; 1993

[17] Bettin, J.; "Practical Use of Generative Techniques in Software Development Projects: an Approach that Survives in Harsh Environments"; *OOPSLA'2001 Workshop on Generative Programming*; 2001; URL http://www.generative-programming.org/oopsla01-workshop.html

[18] Rational Co.; Rose 98 Rose Extensibility User's Guide, 1998.

[19] Franca, L.PA.; "Um Processo de Construção de Geradores de Artefatos"; Tese de Doutorado; PUC-Rio; 2000

[20] Staa, A.; Franca, L.PA.; "Geradores de Artefatos: Implementação e Instanciação de Frameworks"; Anais do XV Simpósio Brasileiro de Engenharia de Software; SBC 2001; pp 302-315

[21] Staa, A.; Franca, L.PA.; "ProtoBD: Prototipador de Modelo de Dados"; Anais do XIV Simpósio Brasileiro de Engenharia de Software; SBC 2000; pp 374-375

[22] Staa, A.; Franca, L.PA.; "A Construction Process for Artifacts Generators using a CASE Tool"; ICSR7 Workshop on Generative Programming 2002; Austin; Texas; EUA, http://www.cwi.nl/events/2002/GP2002/GP2002 Proceedings.pdf

[23] Brodsky, S.; "XMI Opens Application Interchange"; IBM, 1999. http://www-3.ibm.com/software/ad/standards/xmiwhite0399.pdf

[24] Neighbors, J.; "Techniques for Generating Communicating Systems"; *ICSR7 WorkShop on Generative Programming 2002;* Austin, Texas, EUA, Apr. 2002; URL http://www.cwi.nl/events/2002/GP2002/GP2002-Proceedings.pdf

[25] Nentwich, C.; Emmerich, W.; Finkelstein, A.; Zisman, A.; "BOX: Browsing objects in XML"; *Software-Practice and Experience* 30; 2000; pp 1661-1676

[26] Cleaveland, C.; *Program Generators using Java and XML*; Prentice-Hall; 2000

[27] Basset, P.; ”Frame-Based Software Engineering”; *IEEE Software 4(4);* 1987; pp 9-16

[28] Basset, P.; *Framing Software Reuse;* Prentice-Hall; 1996

[29] Basset, P.; ”The Theory and Practice of Adaptive Reuse”; *Symposium on Software Reusability;* 1997; pp 2-9

[30] Oliveira, A.; Oliveira, J.; “Uma Arquitetura para Reduzir a Complexidade e Aumentar a Produtividade do Ciclo de Vida do Desenvolvimento de Sistemas”; *Anais do XIII Simpósio Brasileiro de Engenharia de Software;* SBC; 1999; pp 181-194

[31] Barrére, T.; Prado, A.; Bonafé, V.; "CASE Orientada a Objetos com Múltiplas Visões de Requisitos e Implementação Automática de Sistemas-MVCASE"; Anais do *XIII Simpósio Brasileiro de Engenharia de Software*; SBC; 1999; pp 113-128