Proposta de *Reuse Patterns* para a Reutilização Sistemática em Sistemas de Software Orientados a Objetos

Gabriela Elisa da Cunha
Orientadora Profª Drª Juliana Silva Herbert

Universidade do Vale do Rio dos Sinos – UNISINOS
Av. Unisinos, 950. São Leopoldo, RS
gabriela@atlas.unisinos.br
juliana@exatas.unisinos.br

RESUMO

A tecnologia da reutilização cresceu substancialmente nos últimos anos. Com o aumento dos níveis de reaproveitamento no processo de desenvolver sistemas, as organizações têm reduzido significativamente seus custos e têm alcançado ganhos incomparáveis em qualidade de produto, associados à redução do tempo de desenvolvimento. Como decorrência desta mudança de cultura organizacional, que incentiva uma nova forma de pensar sobre sistemas, emergem as descrições de soluções padrão. Este novo tipo de documentação, o *pattern*, descreve situações de problemas comuns no ciclo de desenvolvimento de sistemas e fornece subsídios para apoiar a solução. A partir da análise de correlações entre aspectos considerados importantes para a reutilização, são propostos neste artigo *patterns* específicos para descrever soluções nesta área, contribuindo, assim, na área de pesquisa relacionada ao aperfeiçoamento da qualidade do software Orientado a Objetos.

PALAVRAS-CHAVE: Reutilização; *Patterns;* Qualidade do software Orientado a Objetos.

ABSTRACT

The technology of the reuse grew substantially in the latest years. With the increase of levels of reuse in the process of developing systems, the organizations have reduced significantly its costs and have reached incomparable profits in the quality of the final product. Besides, all these factors are associated with the reduction of development time. As a result of changing the organizacional culture, that stimulates a new way of thinking on systems, the descriptions of standard solutions appeared. This new kind of documentation, the pattern, describes situations of common problems in the cycle of systems development and give subsidies to support the solution. From the analysis of relations between aspects considered important for the reuse, it was possible to suggest in this article specific patterns to describe solutions in this area, contributing, thus, in the research area related to quality improvement of the Object-Oriented Software.

KEYWORDS: Reuse; *Patterns*; Quality of Object-Oriented Software.

1. Introdução

O processo de desenvolvimento de sistemas sempre foi marcado pela busca de qualidade e de produtividade. Isto tem significado consideráveis investimentos em metodologia, treinamento e aferições de qualidade [BOO98]. O surgimento do paradigma de desenvolvimento de sistemas Orientado a Objetos (OO) contribuiu para o alcance destes objetivos, subsidiando a reutilização de elementos previamente construídos e testados.

Já foi constatado em vários trabalhos [GAM94] [PRE95] um aumento significativo em reutilização e o conseqüente incremento de qualidade, devido à utilização de *frameworks*. A utilização destas estruturas de aplicação desencadeou a necessidade de documentar formas e regras para a obtenção das melhores soluções. São os chamados *patterns*, mais comumente associados ao âmbito do projeto e do teste do sistema.

Este artigo foi baseado no Trabalho de Conclusão de curso realizado pela primeira autora [CUN99], submetido como requisito parcial para a obtenção do grau de Bacharel em Análise de Sistemas, em junho de 1999.

1.1 Contribuição

Ao ser identificada uma lacuna no âmbito da documentação de soluções padrão, foi possível propor uma inovação na área de descrição de soluções: os *reuse patterns*, específicos para a reutilização. Para a confecção de tais *patterns* considerou-se a modelagem de elementos OO através da *Unified Modeling Language* (UML), relacionada à aplicação de métricas de qualidade e aos decorrentes ganhos em reutilização. Os *reuse patterns* propostos neste artigo fazem parte de um conjunto de *patterns* proposto em [CUN99]. Esta nova forma de documentar soluções permite o incremento da qualidade e da produtividade no desenvolvimento OO.

Até agora, algumas categorias de descrições de solução padrão já obtiveram aceitação entre projetistas e desenvolvedores de sistemas, tais como os *design patterns* [GAM94]. Outras ainda são consideradas novidades, tais como os *test patterns* [McG99]. Ambas as categorias abordam fases específicas do ciclo de desenvolvimento. Os *reuse patterns*, cuja proposição é apresentada neste artigo, caracterizam-se por identificar aspectos importantes para a reutilização em qualquer etapa do ciclo de desenvolvimento de sistemas. Na pesquisa realizada quando da elaboração do Trabalho de Conclusão [CUN99] não foi encontrada nenhuma categoria de *patterns* similar à proposta neste artigo.

A presente seção apresentou a motivação para a realização deste trabalho, assim como destacou sua principal contribuição à área, citando, em seguida, alguns trabalhos relacionados.

A Seção 2 apresenta a importância da UML para a obtenção de ganhos em reutilização através da modelagem de elementos OO. Na Seção 3 justifica-se a adoção de métodos de reutilização em todas as fases de desenvolvimento de sistemas. A Seção 4 salienta a utilização de *patterns* para o compartilhamento das melhores soluções em software. Na Seção 5, é apresentada a proposta dos *reuse patterns* como uma nova categoria de *patterns*, baseada em relacionamentos existentes entre alguns aspectos considerados importantes para a reutilização em um ambiente de desenvolvimento de sistemas OO. A Seção 6 apresenta considerações sobre a realização deste trabalho, assim como extensões imediatas possíveis a partir destas idéias. E, finalmente, na Seção 7, são apresentadas as principais referências bibliográficas utilizadas para a realização deste trabalho.

2. *Unified Modeling Language* – UML

A UML é uma linguagem de modelagem que agrega conceitos pertencentes a um consenso que vários autores possuíam da OO [RAT97]. A UML destina-se a delimitar fronteiras de um sistema, modelar sua estrutura estática e seu comportamento, além de permitir visualizar a funcionalidade e a implementação. Como significativa contribuição para o desenvolvimento, a UML comporta conceitos considerados de alto nível, como *frameworks* e padrões de projeto (*design patterns*) [PRE95] [GAM94].

A UML foi considerada a linguagem de modelagem de sistemas OO mais adequada para a utilização nos *reuse patterns*, uma vez que pode ser considerada simples, poderosa e independente do domínio no qual é aplicada, além de estar notoriamente se transformando em um padrão para modelagem.

3. Reutilização

A principal motivação para a reutilização está relacionada ao aumento dos níveis de qualidade e produtividade no desenvolvimento de software [BOO98]. O aumento de qualidade é uma conseqüência da reutilização de componentes que foram previamente documentados, testados e aprovados. O aumento da produtividade é resultado de uma redução no tempo de desenvolvimento, evitando a reconstrução de partes do sistema que já existem.

A reutilização sugere níveis mais altos de abstração, dos quais resultam as estruturas de aplicação (*frameworks*). Considera-se como *framework* uma arquitetura semi-pronta de aplicação, que consiste em um conjunto de classes que são especificamente desenhadas para serem refinadas e usadas como um grupo. Este conjunto de classes incorpora um desenho abstrato do domínio do problema e encoraja a reutilização. Os *frameworks* representam o nível mais alto de reutilização. Sua aplicabilidade sugere a necessidade da adoção de modelos de solução padrão, que são denominados *patterns*. Além dos *patterns* existem os *metapatterns* (Subseção 4.2), que são tipos de *design patterns* (Subseção 4.1), propostos por Pree [PRE95], que suportam o desenvolvimento e a reutilização de *frameworks* na fase de projeto OO.

A reutilização pode ser considerada a principal motivação para a concentração de esforços no desenvolvimento de *patterns*. Para a obtenção destas soluções padrão, é necessário identificar os elementos estáveis do sistema, que poderão fazer parte de várias soluções para problemas similares. O entendimento dos aspectos fundamentais que contribuem para a reutilização é de fundamental importância para que seja possível propor *patterns* específicos de reutilização.

4. *Patterns*

Apesar de todas as melhorias já propostas no processo de desenvolvimento de sistemas, as organizações ainda precisam estruturar o seu conhecimento para minimizar problemas na comunicação das soluções e práticas adotadas [SAL97]. Neste sentido, a utilização de *patterns* tem-se fixado como uma das abordagens mais promissores voltadas à melhoria da qualidade de desenvolvimento. Os *patterns* têm como característica fundamental a identificação e o registro apropriado da solução, para que seja possível disseminá-la posteriormente.

O *pattern* será sempre oriundo de alguma experiência prática, embora não possa ser validado através de testes. A identificação do *pattern* é muito importante, devendo, portanto, direcionar o pensamento do usuário para o problema em questão. Um *pattern* é necessário para documentar, de forma genérica, a solução de um problema importante e de ocorrência repetida.

A Subseção 4.1 apresenta os *design patterns* e a sua importância. A utilização de *metapatterns* é abordada na Subseção 4.2. Por fim, a Sub-seção 4.3 compreende as linguagens específicas para a descrição de *patterns*, as *pattern languages*.

4.1 *Design Patterns*

De maneira geral, os *design patterns* podem ser entendidos como uma descrição da comunicação entre classes e objetos destinada a resolver um problema genérico de projeto em um contexto particular [GAM94]. Estes *patterns* são destinados a descrever soluções simples e elegantes para problemas específicos do projeto OO, sendo elaborados a partir da captura de soluções genéricas que foram desenvolvidas e evoluíram ao longo do tempo, resultando em uma forma sucinta e fácil de ser aplicada.

O catálogo de *design patterns*, proposto por Gamma [GAM94], já obteve aceitação no âmbito da Engenharia de Software por representar o processo de captura, abstração e classificação dos aspectos importantes para a criação de um projeto OO reutilizável.

4.2 *Metapatterns*

Entende-se por *metapatterns* o conjunto de *design patterns* que descreve como construir *frameworks*, independente de um domínio específico [PRE95]. Os *metapatterns* constituem uma abordagem poderosa que pode ser aplicada para categorizar e descrever qualquer *design pattern* que exemplifique um *framework*. Desta forma, os *metapatterns* não substituem a especificação dos *design patterns*, mas servem como complemento, auxiliando na sua documentação.

4.3 *Pattern Language*

Pattern language, ou linguagem de *patterns*, é a coleção de *patterns*, obtidos em todos os níveis de abstração, que oferece um trabalho conjunto na resolução de um problema complexo, cuja solução sugere uma linha de conduta [SAL97]. Desta forma, cada *pattern*, então, possui uma instrução explícita de como se realiza algo.

A linguagem de *patterns* proposta por Coplien [COP95] tem como objetivo apoiar as técnicas emergentes na comunidade de projetos de software, sendo voltada, principalmente, para a evolução organizacional. Coplien sugere a estrutura dos *pattern* em seis seções:

- problema: descrição do problema a ser resolvido;
- contexto: descrição do ambiente que abrange a solução do problema e das circunstâncias envolvidas;
- forças: relação das características relevantes que têm influência no problema;
- contexto resultante: descrição do ambiente resultante após a aplicação da solução;
- racionalidade: descrição das razões e justificativas para a adoção da solução proposta.

A estrutura desta linguagem foi utilizada para descrever os *reuse patterns*. Desta forma, a comunicação das idéias de reutilização, propostas neste artigo, utiliza uma linguagem de *patterns* já conhecida e divulgada na literatura relacionada.

5. *Reuse Patterns*

O conhecimento a respeito dos domínios deve ser reutilizado com o objetivo de gerar soluções padrão. Para tanto, é necessário identificar os elementos estáveis que estão presentes em várias soluções de problemas similares. Os resultados em reutilização dependerão do entendimento destes aspectos de estabilidade ou variabilidade do domínio do problema.

Este trabalho propõe uma nova categoria de descrições de solução padrão: *patterns* específicos para a reutilização, denominados *reuse patterns*. Estes *patterns* não foram obtidos a partir de resultados práticos porque, até o presente momento, não foram adequadamente aplicados (esta é uma das extensões futuras mais imediatas no seguimento deste trabalho). Eles foram deduzidos a partir de estudos das características de OO mais relevantes para o estabelecimento de um bom nível de reutilização. Para a descrição dos *reuse patterns*, utilizou-se a linguagem de *patterns* proposta por Coplien [COP95], por ser considerada a mais adequada para a documentação das soluções em reaproveitamento.

Acredita-se que os *reuse patterns* possam servir de apoio às ferramentas CASE (*Computer Aided Software Engineering*) OO, na avaliação da modelagem em relação aos indicativos fundamentais de reutilização. A ferramenta de modelagem da empresa *Rational*, *Rational Rose* [RAT97], é atualmente a principal ferramenta de modelagem de sistemas OO, através da UML. Repositórios de elementos reutilizáveis constituem outra maneira de aplicar os *reuse patterns*, sob forma de prover uma melhor documentação e a conseqüente organização dos seus elementos.

As Tabelas 1 e 2 apresentam alguns *patterns* do catálogo composto por sete *reuse patterns*, apresentado de forma completa em [CUN99].

O *reuse pattern* "Quantidade de vezes que a Classe é Referenciada", explicado através da Tabela 1, foi identificado com base na análise de estruturas de aplicação *frameworks*, e indica a confecção de classes abstratas como sendo o tipo de classe de maior índice de reutilização. Estas classes, porém, devem ser planejadas no projeto do sistema, pois necessitam de maior atenção na fase de análise e geram maior custo. As estruturas de aplicação semi-prontas são constituídas, em grande parte, de classes abstratas. Decorre daí a justificativa para a análise dos pontos de flexibilidade do sistema que poderão ser reaproveitados no domínio da aplicação.

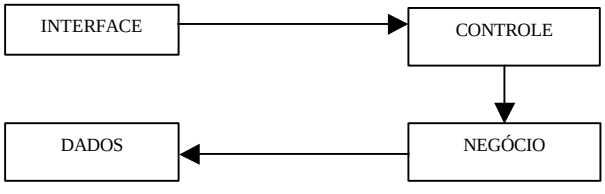
O *reuse pattern* "Implementação da Arquitetura em Camadas", pertinente à Tabela 2, sugere a organização do sistema sob a forma de camadas de abstração. A utilização deste tipo de estrutura implica em uma análise profunda e detalhada dos requisitos do sistema, bem como a identificação dos objetos que serão necessários. Cada objeto, por sua vez, atua em uma das quatro camadas estruturais. A comunicação entre objetos de camadas distintas deve respeitar o fluxo "Interface-Controle-Dados-Negócio", para que sejam asseguradas ao sistema as características de flexibilidade e manutenibilidade e atomicidade, visto que os objetos contemplam apenas a sua própria responsabilidade.

Tabela 1 – *Pattern 1* – Quantidade de vezes que a Classe é Referenciada.

Problema	Como identificar se uma classe abstrata deve ser criada?
Contexto	A classe foi identificada e deve-se verificar se ela oferece condições de ser uma classe abstrata.
Forças	As classes abstratas são constituídas através da união de conceitos genéricos, que serão adequados ao domínio da aplicação através apenas de suas subclasses. As classes abstratas requerem um esforço maior na fase de análise. Classes abstratas são as candidatas em potencial para a reutilização, já que são modeladas para este fim. Elas representam a melhor solução para reutilizar comportamentos não associados diretamente ao domínio da aplicação. Uma classe com muita funcionalidade torna-se mais difícil de ser reutilizada, uma vez que tende a ser mais complexa.
Solução	Identificar, no domínio da aplicação, as classes cuja responsabilidade poderia ser generalizada para facilitar a reutilização.

	Como as classes abstratas são mais complexas, elas devem justificar a sua existência. Devem ser definidas para servirem ao domínio público da organização, deixando flexível a implementação específica. A definição do comportamento das classes abstratas deve ser sob a forma de métodos virtuais. Testar muito bem a classe, identificá-la de forma clara e associar a ela uma documentação. Construir classes pequenas, de funcionalidade específica.
Contexto Resultante	Excelente nível de reutilização. O modelo do sistema torna-se mais robusto e flexível a mudanças. Permite manutenções mais rápidas. Força uma utilização correta dos conceitos de herança. Biblioteca com classes reutilizáveis bem documentadas e de fácil entendimento.
Racionalidade	A experiência comprova que em um modelo de sistemas, de 10 a 15% das classes devem ser abstratas. A intenção em reutilizar a classe pode acarretar em manutenções na classe, sob forma de seu aprimoramento.

Tabela 2 – *Pattern* 2 – Implementação da Arquitetura em Camadas.

Problema	Como estabelecer uma arquitetura em camadas?
Contexto	O projeto de sistemas OO deve possuir uma arquitetura adequada para a implementação de especificações do modelo de classes.
Forças	Consideram-se camadas da arquitetura os vários níveis de abstração entre os diversos serviços que os sistema deve possuir. O conhecimento dos objetos do sistema e de seus relacionamentos é imprescindível para o estabelecimento deste tipo de arquitetura. Devem ser conhecidas as dependências entre os objetos para poder estabelecer a camada na qual estes devem estar situados. É importante, ainda na fase de modelagem, identificar os objetos que Terão função de controle. Estes objetos serão geralmente responsáveis pela coordenação das regras do negócio.
Solução	Utilizar o conceito de pacotes, propostos pela UML [RAT97]. Isto significa que o sistema deve possuir seus objetos dispostos sob a forma de pacotes, cuja organização é apresentada a seguir: • Pacote Interface: objetos de comunicação com o usuário final. • Pacote Controle: objetos de lógica transacional. Funcionam como gerentes que coordenam o trabalho de uma equipe de objetos. Podem atuar, inclusive, como seqüencializadores das atividades entre objetos dependentes. • Pacote do Negócio: objetos que contêm o conhecimento específico do negócio. • Pacote de Dados: objetos que compõem o modelo estável do sistema. Os pacotes devem comunicar-se apenas na seqüência através da qual foram apresentados, uma vez que há dependência entre eles. Por exemplo, um objeto de negócio não pode solicitar nada para um objeto de interface, sob pena de violar a dependência entre as camadas.
Contexto Resultante	A reutilização é a grande conseqüência. A partir da arquitetura em camadas, os elementos reutilizáveis, bem como sua atuação, são de fácil localização.
Racionalidade	A disposição dos sistemas que a organização pretende desenvolver deveria basear-se em: 

	Observa-se a importância da existência dos objetos de controle, mesmo em casos onde a classe é considerada bastante simples. Quando do acréscimo de funcionalidade, os objetos de controle são necessários para coordenar as atividades.

6. Considerações Finais

Em cada um dos *patterns* apresentados, algumas características deste trabalho foram consideradas de forma a auxiliar na resolução de um problema de ocorrência repetida. A identificação e a aplicação de *patterns* no ciclo de desenvolvimento de sistemas OO indicou a conquista de um elevado nível de maturidade do processo, através do acúmulo de conhecimento e de experiência.

Os *reuse patterns*, específicos para apoiar a reutilização, agregam conhecimento a respeito de problemas similares. Dessa forma, entre as várias soluções consideradas para estes problemas, foram identificados os aspectos estáveis que, por sua vez, viriam a se tornar a base da descrição da solução genérica. Os *reuse patterns* foram baseados em observações de modelos OO e conhecimento teórico relacionado. Na pesquisa realizada não foi encontrada nenhuma categoria de *patterns* similar à proposta neste trabalho. Como extensão futura, os *reuse patterns* poderiam ser reavaliados, após serem utilizados de forma prática, no âmbito de desenvolvimento de sistemas OO.

Toda a arquitetura do projeto do sistema deve voltar-se para a reutilização. Uma arquitetura é tão melhor quanto menos transformadora ela for. Isto é, quanto mais próxima ela for da realidade do negócio e dos princípios OO utilizados, menos alterações nas soluções de implementação serão necessárias. Dessa forma, o código resultante será um produto fiel do modelo projetado, refletindo características essenciais na qualidade do produto e do processo de desenvolvimento de software.

7. Referências Bibliográficas

[BOO98] BOOCH, G. "**Quality Software and the Unified Modeling Language"**, abril 1998. *http://www.rational.com/support/techpapers/ softuml.html* .

[COP95] COPLIEN, J. **"Pattern Languages of Program Design"**. 1a. ed. United States of America: Addison-Wesley, 1995.

[CUN99] CUNHA, G. E. **Proposta de *Reuse Patterns* para a Reutilização Sistemática em Sistemas de Software Orientado a Objetos**. Trabalho de Conclusão de Curso. Centro de Ciências Exatas e Tecnológicas/Universidade do Vale do Rio dos Sinos-UNISINOS. 1999.

[GAM94] GAMMA, E. **"Design Patterns – Elements of Reusable Object-Oriented Software."** Massachusetts: Addison-Wesley Publishing Company, 1994.

[LOR94] LORENZ, Mark; KIDD, Jeff. **"Object-Oriented Software Metrics"**. 1a. ed. New Jersey : Prentice Hall, 1994.

[McG99] McGREGOR, John. **"A Component Testing Process",** maio 1999. *http://cyclone.cs.clemson.edu/~johnmc/new/pact/node8.html*

[PRE95] PREE, Wolfgang. "**Design Patterns for Object-Oriented Software Development"**. England: Addison-Wesley Publishing Company. 1995.

[RAT97] RATIONAL Software Corporation. "**UML Semantics – version 1.1"**. 1997. Disponível em *http://www.rational.com/uml.*

[SAL97] SALVIANO, C. F. **Introdução a Software Patterns.** Fundação Centro Tecnológico para Informática - CTI. Campinas. São Paulo, 1997.