Uma Abordagem Orientada a Objetos para Reutilização baseada em Linguagens de Domínio

Flavio Araujo de Mattos
Cláudia Maria Lima Werner

COPPE - Programa de Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro
Caixa Postal 68511
CEP. 21945-970 - Rio de Janeiro, RJ - Brasil

Resumo

Este artigo apresenta uma abordagem de integração das tecnologias compositiva e gerativa de reutilização, mais especificamente as tecnologias de orientação a objetos e a de sistemas baseados em transformação, obtendo flexibilidade e manutenibilidade das tecnologias compositivas, juntamente com um alto grau de abstração e facilidade de uso das tecnologias gerativas. A abordagem dá ênfase ao papel das linguagens de alto nível orientadas a domínios de aplicação.

Abstract

This paper presents an approach for the integration of compositive and generative reuse technologies, more specificaly the object oriented technology and transformation based systems, joining flexibility and maintainability from compositive technologies, and high level abstraction and ease of use from generative technologies. The approach emphasizes the role of application-domain oriented high-level languages in software reuse.

1. Introdução

A reutilização de software, mais que uma nova linha de pesquisa dentro da engenharia de software, é uma necessidade principal dada a crescente demanda por sistemas maiores e mais complexos. Traduzida em termos de limitações de custo, tempo, qualificação e disponibilidade de recursos humanos, esta necessidade transcende as fronteiras da reutilização de código objeto e atinge o processo de desenvolvimento em sua totalidade.

O grande desafio das propostas de amplo emprego da reutilização é abranger a enorme diversidade de formas e tecnologias através das quais o software se manifesta. Uma solução de reutilização a médio e longo prazo deve, ao mesmo tempo, dar apoio às principais linhas tecnológicas hoje adotadas e ser extensível para capturar as novas que virão.

A orientação a objetos, além de oferecer mecanismos específicos úteis ao reutilização (como a herança e o polimorfismo) [Booc94], traz a seu favor o poder de conciliar tecnologias de diversas áreas a que o paradigma já está integrado e frutificando, tais como linguagens de programação, bancos de dados orientados a objetos, métodos de análise e projeto, entre outras. Este caráter interdisciplinar é fundamental para a pesquisa em uma área de tão amplo espectro como a reutilização.

Em [Wern92], propôs-se como continuidade de pesquisa em reutilização a elaboração de uma abordagem híbrida, que combinasse fortemente aspectos gerativos e compositivos. Tal abordagem híbrida permitiria unir o alto nível de abstração e facilidade de uso das tecnologias de geração com a flexibilidade e manutenibilidade das tecnologias de composição. As tecnologias de composição cumpririam, nesta abordagem, o papel de intermediar o refinamento da especificação de alto nível até a construção de um sistema, especificando partes reutilizadas no processo de geração.

Este artigo apresenta uma abordagem de integração das tecnologias de orientação a objetos e a de sistemas baseados em transformação [Matt95]. Esta abordagem dá ênfase ao papel das linguagens de alto nível orientadas a domínios de aplicação, provendo uma nova leitura do processo de desenvolvimento de software como sendo uma evolução de conceitos abstratos dirigida por linguagem, em analogia aos processos intelectuais em que a linguagem exerce o papel de ordenador do pensamento.

O artigo está dividido em cinco seções. Na seção 2, introduzimos a questão da reutilização de software, apresentando as tecnologias de reutilização e a importância da linguagem na reutilização. Na seção 3, apresentamos uma abordagem para reutilização baseada em linguagens voltadas para domínios de aplicação. Na seção 4, é descrita a base conceitual sobre a qual esta abordagem foi elaborada. Finalmente, concluímos o artigo indicando trabalhos em andamento.

2. Reutilização

O desenvolvimento de software é muito mais que a elaboração de programas. Antes e depois desta etapa há um grande esforço intelectual despendido em atividades organizadoras distribuídas ao longo do ciclo de vida de um software. Durante estas atividades, o engenheiro de software constrói novas abstrações e trabalha com as de que já dispõe por experiência ou educação. A ausência de reutilização é percebida quando é observado que, enquanto a experiência prévia de um indivíduo é integralmente reaplicada em problemas similares, os frutos desta experiência não o são.

Diante da crise de software, a perspectiva de aumento de produtividade pela capacidade de aproveitar produtos de esforços anteriores delineia um horizonte onde a produção de software assuma feições de produção industrial, com benefícios de qualidade e produtividade. Enquanto o desenvolvimento se dá em pequena escala, o âmbito do problema é meramente tecnológico. Quando a experiência a

reutilizar transcende a de uma pessoa ou equipe fixa e pretendemos exercer a reutilização de modo intensivo e por um período arbitrariamente grande de tempo, temos caracterizado a reutilização em larga escala. Neste caso, somente a aplicação de técnicas e disciplinas específicas permite avaliar com segurança se algum trabalho pode ser reaproveitado com esforço menor ao do desenvolvimento a partir do zero. A viabilidade da solução de reutilização em larga escala envolve mudanças no modo atualmente empregado para a construção de software. A busca das soluções que viabilizem a reutilização é o alvo da pesquisa denominada Engenharia de Software Reutilizável (ESR) [Free87].

Neste trabalho adotamos uma visão ampla do que é software, incluindo em um sistema o conhecimento usado para desenvolvê-lo. Nesta visão, não há desenvolvimento sem reutilização, mas há a distinção de reutilização informal, que é totalmente subjetivo em qualidade e produtividade, e da reutilização formal, que repousa sobre instrumentos de apoio e cuja aplicação é desejável em larga escala.

A reutilização formal vem sendo amplamente descrita como uma área onde o sucesso é a exceção e não a regra [Prie93]. A percepção de que a reutilização formal não é largamente empregada serve de estímulo a novas experiências e à produção de tecnologia para transformar a reutilização informal em reutilização formal.

A reutilização formal caracteriza-se, portanto, pela intenção de reutilizar sistematicamente. Esta intenção pode ser traduzida no cuidado a cada passo da evolução de um sistema de software, aproveitando ao máximo as experiências prévias similares. Para que o aproveitamento seja máximo, a experiência prévia deve estar de algum modo codificada em diversos níveis de abstração para reutilização futura.

2.1. Tecnologias de Reutilização

Tradicionalmente, as tecnologias para reutilização são divididas segundo dois princípios fundamentais de reutilização: composição e geração [Bigg87]. O princípio da composição é caracterizado pela criação de software através da reutilização de componentes atômicos e idealmente não modificáveis no curso de sua reutilização. O princípio da geração baseia-se na reutilização de seqüências de instruções e seqüências de transformações embutidas em um gerador, sendo possível que o produto gerado seja bem diferente das seqüências que lhe deram origem.

São exemplos de reutilização compositiva as tecnologias baseadas em bibliotecas de sub-rotinas e orientação a objetos. Sob a reutilização gerativa estão as linguagens de altíssimo nível, os geradores de aplicação e os sistemas baseados em transformação.

Estas duas categorias, entretanto, não são estanques, sendo a fronteira entre o compositivo e o gerativo cada vez mais tênue e difusa, na medida em que são elaboradas soluções que mesclam características de ambas as linhas.

2.2. A importância da linguagem na reutilização

Os produtos materiais do desenvolvimento de software (como programas, documentos e diagramas) são reflexos da atividade intelectual que os produziu. Pensar que a reutilização pode ater-se a este material produzido é incorrer no erro de perder sua essência intelectual. É dissociar forma e conteúdo. A dinâmica da reutilização em qualquer de suas formas sempre recai em, a partir da representação, apreender os conceitos abstratos que ela representa e deles derivar uma nova forma que nos interessa. Cabe à tecnologia facilitar esta transformação, de modo que o que foi matéria prima para a mente, também, o seja para a representação.

Uma das mais importantes linhas de pesquisa para a reutilização é o desenvolvimento de linguagens de muito alto nível que propiciem uma forma de expressão intelectualmente natural dentro de um domínio específico do conhecimento. Esta abordagem possui forte respaldo na ciência da cognição, onde pensadores como Vigotsky apresentam o pensamento como uma processo fortemente guiado pela linguagem. "A formação de conceitos (...) é dirigida pelo uso de palavras como meio de centrar ativamente a atenção, abstrair determinados traços, sintetizá-los e simbolizá-los por meio de um signo" [Vigo87]. Estas linguagens de altíssimo nível expressam o conhecimento como um padrão de atribuição de significados a suas sentenças.

Dentro desta linha uma das mais importantes propostas foi a estratégia DRACO [Neig89], que lançou as bases para a análise de domínio[1] e provê um suporte genérico à construção de linguagens para domínios de aplicação. A abordagem DRACO prevê que cada domínio da aplicação é representado por uma linguagem específica, que serve como organizador de objetos e operações modelados a partir do domínio. A linguagem de domínio passa a ser exercida no desenvolvimento de sistemas para aquela área, oferecendo o que Neighbors caracteriza como *reutilização de análise*. Trabalhos de aperfeiçoamento desta abordagem vem sendo realizados trazendo novas perspectivas [Leit92][Leit94].

3. A Abordagem Gremlin

Nesta seção, apresentamos a abordagem Gremlin, cujo nome representa nosso objetivo principal que é o de gerar reutilização por mecanismos baseados em linguagens.

Gremlin é uma abordagem que se alinha com a estratégia DRACO, no sentido de enfatizar o papel da linguagem de alto nível orientada ao domínio de aplicação na reutilização de software. Gremlin, à semelhança do DRACO, oferece níveis de estratificação de conceitos que mapeiam domínios de aplicação, para formar espaços de trabalho dentro dos quais a solução abstrata para um problema é especificada. Tanto a construção destes espaços de trabalho, como os mecanismos de tradução das especificações de alto nível em especificações progressivamente mais próximas da máquina foram desenvolvidos para explorar as vantagens do paradigma da orientação a objetos (O.O.)

Para isto, foi desenvolvido um modelo de objetos particular, que acrescenta às características tradicionais de O.O. um suporte mais rico às transformações. Este modelo estendido foi integrado à abordagem de reutilização orientada a domínios de aplicação, servindo como ferramental básico da caracterização destes domínios.

A reutilização na abordagem Gremlin se dá em dois níveis. No primeiro nível, componentes de software orientados a objetos são reutilizados na representação de novos domínios de aplicação e na evolução deste domínio no tempo. A orientação a objetos é empregada para representar conceitos de análise e alternativas de projeto ("design") disponíveis para a solução de problemas neste domínio de aplicação. O modelo estendido é empregado, ainda, na caracterização das linguagens de alto nível especializadas no domínio. O segundo nível de reutilização é obtido pelo uso das linguagens de domínio para a construção de soluções de problemas.

---

[1] A análise de domínio é a atividade que organiza o conhecimento sobre a construção de software em uma determinada área [Aran91].

3.1. O Modelo Orientado a Objetos Estendido

Um modelo de objetos é a sistematização dos princípios da orientação a objetos para fins específicos, geralmente associado a uma linguagem de programação, banco de dados ou ferramenta de software que dá suporte operacional a ele. Cada modelo de objetos possui sua definição particular para os conceitos da orientação a objetos. Estas divergências tornam as linguagens orientadas a objetos diferenciadas, de um modo similar ao que acontece com as linguagens de programação convencionais. As diferenças entre estes modelos privilegiam a construção de representações de uma determinada classe de problemas em detrimento a outras. O que varia é a naturalidade com que as soluções de uma área são construídas em cada classe de problemas.

Os modelos orientados a objetos convencionais não dão suporte específico à transformações de programas ou de especificações. Um ambiente orientado a objetos que promove a reutilização baseada em geração convive, por isto, com dois universos conceitualmente distintos. Em um deles está a orientação a objetos, cujas facilidades gerais são usadas para a implementação da vuncionalidade do ambiente e da representação interna das especificações. Em outro está o modelo de transformações, construído como uma camada conceitual grandemente independente da orientação a objetos subjacente, em que a reutilização é exercido.

Um modelo de objetos estendido para dar suporte específico a transformações promove a integração destes dois universos, permitindo a elaboração de ambientes de reutilização que explorem o melhor de cada abordagem. Os requisitos identificados para tal modelo são:

1. Mecanismo geral de transformação - O modelo estendido deve ser dotado de um mecanismo genérico de transformação, bem como de instrumentos para especializar estes mecanismos. Deste modo, mecanismos específicos de reutilização gerativa podem ser criados.
2. Homogeneidade conceitual - As extensões do modelo devem estar homogeneamente integradas às demais características da orientação a objetos, evitando a distinção conceitual do que seja ou não gerativo.
3. Fidelidade ao paradigma da orientação a objetos - As características de herança, encapsulação, cooperação através de mensagens e polimorfismo devem ser preservadas. Isto garante que não sejam perdidos o poder de expressão e a generalidade da orientação a objetos.

O modelo de objetos estendido foi criado a partir de uma revisão do mecanismo de troca de mensagens entre objetos. De um modo geral, os modelos de objetos são muito similares no modo de implementação da troca de mensagens, segundo o padrão definido na figura 1. Inicialmente, o emissor codifica a mensagem como um seletor, usualmente um nome ou um número que a identifica. Opcionalmente, objetos complementares podem ser agregados a ela na forma de argumentos. Quando o receptor recebe a mensagem, ele usa uma função de decodificação para determinar o código que será ativado. Ao fim da execução, um valor é retornado ao emissor como resposta.

Buscando uma analogia com os mecanismos encontrados na comunicação humana, observamos que a estrutura das mensagens são de organização extremamente complexa. Dentro desta complexidade, são estabelecidos os relacionamentos entre os significantes, de modo que o significado final da comunicação possa ser construído pelo receptor da mensagem. Apenas em poucos casos, a interpretação de uma mensagem é imediata como na orientação a objetos. Na maioria das vezes, a compreensão do significado de uma mensagem é obtida por processos intelectuais superiores, intimamente ligados ao processo de pensamento e à linguagem humana [Vigo87].

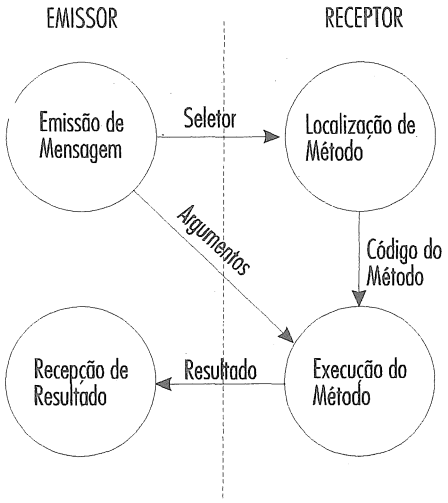

Figura 1 - Mecanismo tradicional de troca de mensagem entre objetos

Portanto, uma das formas para se obter os requisitos anteriormente identificados para um modelo de objetos estendido é aproximarmos o mecanismo de troca de mensagens à comunicação humana. Tal aproximação envolve: a adoção de mensagens de estrutura complexa, um mecanismo mais poderoso de interpretação das mensagens e a encapsulação deste mecanismo em uma linguagem que define a organização das mensagens que um objeto pode receber.

A figura 2 mostra o modelo de troca de mensagens estendido. Neste modelo, as mensagens de um objeto são descritas por uma linguagem livre de contexto, cujo mecanismo de transformação é regido por procedimentos associados à aplicação de produções da linguagem. O paradigma de envio de mensagens permanece muito próximo da orientação a objetos tradicional. A interpretação da mensagem é um método de compilação tradicional, que passa por sucessivas transformações da mensagem original até que um resultado seja completamente definido.

3.2. Domínio de Aplicação no enfoque Gremlin

Para desenvolver software de qualquer tipo usamos, mesmo que informalmente, a especificação de requisitos do que ele fará. A necessária interpretação desta especificação requer sempre informações sobre o contexto em que o software está inserido. Hoje, reconhecemos que este conhecimento sobre o contexto pode ser organizado segundo a área do conhecimento ou da tecnologia a que cada software se presta. Este conhecimento, organizado e sistematizado, dá origem a um grande potencial de reutilização que pode ser exercido de diversos modos. A própria organização do conhecimento consiste em elemento passível de reutilização. O maior ganho, porém, advém da construção de software reutilizável que dê suporte operacional a esta área.

Denominamos *domínio de aplicação* a área do conhecimento que se serve ou se servirá de software, e que pretendemos organizar para a reutilização. Nesta definição, caracterizamos um ponto de vista em que os domínios de aplicação não pré-existem à reutilização. O domínio de aplicação apenas nasce quando há a percepção do seu potencial de reutilização. As técnicas de organização do conhecimento em domínios de aplicação são alvo da pesquisa em *análise de domínios*. Hoje, existem diversas propostas de métodos para análise de domínios [Aran94].

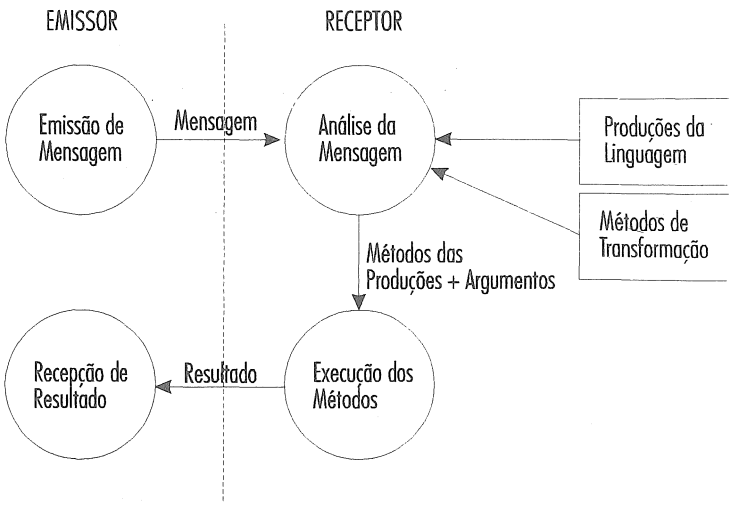

Figura 2 - Extensão do mecanismo de troca de mensagens

Um ambiente para a reutilização em larga escala deve prover uma base conceitual em que os diferentes métodos de análise de domínio possam se encaixar. Na próxima seção, apresentamos a base conceitual utilizada pela abordagem Gremlin para a reutilização em domínios de aplicação específicos.

4. Patamar de Reutilização, o representante Gremlin

A abordagem Gremlin agrupa as informações reutilizáveis em ambientes de trabalho, denominados *patamares de reutilização*. Para cada domínio de aplicação identificado, há um único patamar de reutilização. Um patamar de reutilização concentra todas as informações consideradas relevantes em um domínio de aplicação, incluindo documentação, uma base de conceitos do paradigma e componentes de software que especificam, operacionalmente, a visão atual que se tem deste domínio. O patamar de reutilização cumpre a função de organizar esta diversidade de informação em um ambiente de desenvolvimento coerente.

Um patamar de reutilização pretende ser um ambiente suficiente para o desenvolvimento de soluções de software em um domínio de aplicação específico. Para conquistar este objetivo, é necessário que dê suporte básico a várias atividades dentro do ciclo de vida do desenvolvimento. A fim de atender a diversos ciclos de vida, estabelecemos como requisitos fundamentais de um patamar de reutilização:

- Apoiar o processo de compreensão de conceitos do domínio, bem como dos instrumentos através dos quais são reutilizados;
- Representar modelos de conceitos do domínio de aplicação, incluindo modelos operacionais, que possam ser empregados na construção de software; e
- Representar linguagens de desenvolvimento orientadas ao domínio de aplicação.

As tomadas de decisão envolvidas em um processo de reutilização exigem que o engenheiro de software conheça os conceitos do domínio de aplicação envolvidos no processo. Por isto, o suporte à compreensão é fundamental em um ambiente de reutilização. A abordagem Gremlin apóia o estudo dos conceitos de um domínio de aplicação através da capacidade de registrar documentos em

formato livre em uma rede de hipertexto. Navegando por esta rede, o usuário adquire o conhecimento sobre o domínio de aplicação, capacitando-se para o processo de reutilização.

Dentre as possíveis representações de um domínio de aplicação, a construção de modelos operacionais é aquela que oferece resultados à reutilização em prazos mais curtos. Um modelo operacional registra aspectos estruturais e comportamentais do domínio de aplicação, de modo que pode ser empregado em protótipos e em avaliações de qualidade. A abordagem Gremlin usa a orientação a objetos para representar modelos operacionais. Deste modo, um modelo operacional representa arquiteturas do domínio de aplicação e padrões de interação entre objetos do domínio, similarmente às abordagens que empregam a tecnologia de *frameworks*[2] [Cima94]. A maturidade de um modelo operacional orientado a objetos é obtida quando este é formado por componentes reutilizáveis, que podem ser empregados na construção de soluções de software dentro do domínio de aplicação.

O papel das linguagens de alto nível é o de oferecer um mecanismo para descrever soluções de software, abstraindo-se dos detalhes irrelevantes à concepção da solução. Estes detalhes são supridos durante o processo de tradução da linguagem, seguindo padrões de complexidade variável. Retirando estes detalhes da atenção do engenheiro de software, a linguagem de alto nível dirige sua atenção para as características relevantes do domínio de aplicação. Este processo de tradução poupa trabalho ao engenheiro de software, aumentando sua produtividade e reduzindo as áreas onde pode haver a incidência de erro humano. Ao direcionar a atenção, também, influi positivamente a capacidade de criar soluções [Vigo87].

Para atender aos requisitos enumerados, o patamar de reutilização é composto por quatro subsistemas (figura 3):

- **Subsistema de documentação** - responsável pela armazenagem, organização, classificação e acesso a documentos em formato livre;
- **Subsistema de modelagem** - responsável pela construção e implementação de modelos de software, destinados a representar conceitos de domínios de aplicação como arquiteturas de software reutilizável;
- **Subsistema lingüístico** - responsável por construir linguagens de alto nível orientadas a domínios, para a especificação de soluções de software, e por traduzir estas especificações em termos de reutilização de arquiteturas já definidas; e
- **Subsistema Compositivo** - responsável por classificar e organizar os diversos tipos de informação (documentos, modelos e linguagens) de modo útil à reutilização.

4.1. Subsistema de Documentação

O subsistema de documentação provê mecanismos para registrar e tornar disponíveis informações pertinentes ao exercício da reutilização dentro de um patamar de reutilização. A unidade de armazenamento de informação é chamada *documento*. Dentro de um patamar de reutilização, um documento é qualquer objeto informativo, que pode ser examinado ou alterado através de uma interface adequada.

---

[2] Frameworks consistem de classes que foram especialmente projetas para serem refinadas e usadas em grupo [Wirf90].

Os documentos integram-se aos demais objetos dentro do patamar de reutilização em uma rede. O engenheiro de software interessado em reutilização navega por ela como em um hipertexto, de modo a atingir os documentos ou outros objetos que deseje.

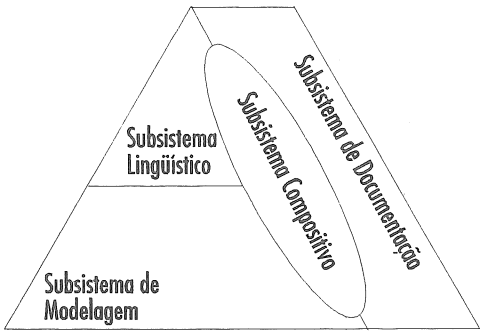

Figura 3 - Relacionamento entre os subsistemas de um patamar de reutilização

4.2. Subsistema de Modelagem

Uma vez que os modelos de objetos são a forma de representação mais importante para a reutilização dentro de um patamar de reutilização, sua construção deve avançar em conjunto com o processo de análise de domínio. Por isto, o subsistema de modelagem precisa dar suporte a ciclos iterativos de desenvolvimento de modelos. Nestes ciclos, a construção de um modelo surge pelo refinamento sucessivo a partir de um modelo abstrato e incompleto. A cada refinamento, o modelo de objetos incorpora mais informações colhidas da análise do domínio de aplicação. É inevitável que, neste processo de refinamento, algumas decisões de representação sejam tomadas. Para reduzir o impacto destas decisões na generalidade da representação do domínio em consideração, é importante avaliar as alternativas disponíveis.

O subsistema de modelagem representa cada modelo de um patamar de reutilização como uma entidade distinta, composta de classes e objetos inter-relacionados. O processo de refinamento é exercido sobre estas entidades e pode envolver a construção de versões alternativas para a avaliação das decisões de representação consideradas.

Um modelo é dito *prototipável* quando a definição das classes é suficientemente rica para a construção de protótipos de soluções. O papel dos protótipos é levantar aspectos específicos do desenvolvimento de aplicações no domínio, baseado nos conceitos representados pelo modelo do domínio de aplicação.

Um modelo é dito *maduro* quando suas classes constituem componentes de software cuja qualidade seja julgada adequada para a reutilização dentro de um domínio de aplicação. Cada patamar de reutilização disponível para o desenvolvimento de software está associado a um único modelo de objetos, necessariamente maduro. Este modelo representa todos os aspectos do domínio de aplicação disponíveis para reutilização.

Para cada linguagem de um patamar de reutilização, há pelo menos uma gramática que reconheça sua sintaxe. Quando duas ou mais gramáticas reconhecem a mesma linguagem, elas são ditas *gramáticas polimórficas* e representam alternativas de tradução para uma mesma linguagem. O nome "gramática polimórfica" vem do modelo de objetos subjacente, que trata a tradução de sentenças como envio de mensagens. No sentido rigorosamente O.O., dois objetos que atendem de modo diferente a mesma mensagem são ditos polimórficos. Uma exemplo trivial de gramáticas polimórficas são gramáticas que traduzem uma especificação de alto nível em diferentes linguagens de baixo nível.

Componentes gramaticais ou gramáticas abstratas são fragmentos de linguagens que são usados para construir novas gramáticas, ou para criar novas versões de gramáticas já existentes, de modo a alterar sua sintaxe ou sua semântica de tradução. As gramáticas, abstratas ou não, são objetos da mesma natureza, com a diferença que a definição das gramáticas abstratas é incompleta, impedindo que ajam como elementos transformadores de sentenças.

4.4. Subsistema Compositivo

O subsistema compositivo tem por finalidade prover o instrumental básico para a obtenção de reutilização compositiva na construção das estruturas complexas que constituem um patamar de reutilização, proporcionando a reutilização da experiência prévia na estruturação de patamares de reutilização. Este subsistema é responsável por organizar grandes conjuntos de componentes de software, sejam eles documentos, classes ou gramáticas. Além disto, oferece mecanismos de busca e seleção de componentes, de modo que estes são tornados disponíveis ao usuário. O exercício específico da integração destes componentes em uma organização é realizado dentro de um dos demais subsistemas.

O processo de seleção de componentes é baseado em um esquema de classificação por facetas [Prie91]. Esta classificação auxilia a exploração seletiva de conjuntos de componentes, de modo a identificar aqueles que contribuem ao processo de reutilização.

5. Conclusões

Neste artigo apresentamos uma abordagem para a reutilização de software orientada a domínios de aplicação. A abordagem baseia-se no emprego de técnicas compositivas, oriundas da orientação a objetos, na construção de modelos genéricos sobre domínios de aplicação, cuja manipulação é regida por linguagens especializadas construídas para estes domínios.

Esta abordagem compreende a especificação de um modelo de objetos que dá suporte nativo a mecanismos transformacionais e caracteriza um modelo de extensão do modelo baseado no enriquecimento do mecanismo de troca de mensagens. Sobre tal modelo elaborou-se uma proposta de organização de domínios e de linguagens especializadas que aproveita a reutilização compositiva em sua construção e a reutilização gerativa na criação de sistemas aplicativos.

Para dar suporte operacional à abordagem descrita neste artigo, foi desenvolvido um protótipo de ferramenta, escrito na linguagem C++, em microcomputadores 486 [Matt95]. Neste momento, estão sendo realizados experimentos que possibilitarão a avaliação empírica da abordagem apresentada, dentro de domínios de aplicações específicos. Esta avaliação contempla o esforço de aprendizado das técnicas apresentadas, os benefícios e os problemas eventualmente encontrados.

Os componentes de software reusável encontrados em um modelo maduro de um domínio de aplicação são chamados *componentes semânticos*. Estes componentes dividem-se em duas grandes categorias. A mais importante engloba os *componentes de domínio*, que descrevem aspectos relevantes do domínio. A segunda categoria de componentes semânticos refere-se aos *componentes auxiliares*, que se prestam a tarefas secundárias, mais relevantes às implementações de um modelo, à verificações de consistência, etc.

Cada patamar de reutilização possui um conjunto de componentes peculiar, chamados de *componentes fundamentais*. O que caracteriza um componente fundamental não é o seu caráter semântico ou auxiliar, e sim o fato de não possuir definição dentro do patamar de reutilização em que está inserido. A definição de um componente fundamental está em outro patamar de reutilização (e, portanto, seu significado reside em um domínio de aplicação diverso). Estes componentes servem de ponte entre este domínio e domínios de nível mais baixo, estabelecendo uma ordem parcial entre os domínios semelhante à criada na abordagem DRACO[3].

4.3. Subsistema Lingüístico

O subsistema lingüístico é responsável por oferecer o potencial de reutilização do modelo de objetos do domínio de aplicação através de linguagens de desenvolvimento especializadas. O analista de sistemas comum vê um patamar de reutilização como um ambiente de desenvolvimento, dotado de ferramentas de apoio à construção de sistemas. Através deste ambiente, ele tem acesso a uma ou mais linguagens de desenvolvimento voltadas àquele domínio de aplicação. Estas linguagens e ferramentas, todavia, compartilham a mesma base conceitual que representa o domínio de aplicação, com a maior fidelidade possível.

A proposta da multiplicidade de linguagens de desenvolvimento em um domínio vem de diversos fatores. Em primeiro lugar, podemos constatar empiricamente de que os domínios de aplicação já bem caracterizados podem possuir mais de uma linguagem de desenvolvimento. Em segundo lugar, sabemos que uma linguagem de desenvolvimento é um produto tecnológico e, como tal, sujeito a mudanças e evolução. A base conceitual de um domínio de aplicação, ao contrário, tende a assumir um caráter mais estável, de modo que é necessário estabelecer um certo grau de liberdade entre domínio e linguagem de desenvolvimento. Gremlin oferece esta independência relativa, abrindo a possibilidade de que coexistam mais de uma linguagem de desenvolvimento dentro de um mesmo domínio.

A unidade de organização do subsistema lingüístico é a *gramática*. Conceitualmente, uma gramática é uma descrição da sintaxe de uma linguagem do patamar de reutilização, bem como dos mecanismos de transformação de sentenças desta linguagem em objetos do modelo de objetos do domínio.

Operacionalmente, gramáticas são objetos do modelo O.O. estendido, cuja interface reconhece sentenças desta linguagem. Cada gramática tem a capacidade de sintetizar objetos que representam a sentença da gramática recebida, através de transformações encapsuladas em sua implementação.

---

[3] A hierarquia de domínios do DRACO caracteriza um grafo orientado acíclico, onde o refinamento é orientado de domínios de níveis mais alto de abstração (i.e. *domínios de aplicação e domínios de modelagem)* para os de nível inferior (i.e. *domínios executáveis)* [Leit94].

Referências Bibliográficas

[Aran91] Guillermo Arango; Rubén Prieto-Díaz - *"Domain Analysis Concepts and Research Directions"* em Domain Analysis and Software Systems Modeling, (ed.) Rubén Prieto-Díaz & Guillermo Arango - IEEE Computer Society Press, 1991.

[Aran94] Guillermo Arango - *"Domain analysis methods"* em Software Reusability, (ed.) Wilhelm Schäfer, Rubén Prieto-Díaz, Masao Matssumoto - Ellis Horwood, 1994.

[Bigg87] Ted J. Biggerstaff e Charles Richter - *"Reusability Framework, Assessment, and Directions"*, IEEE Software, Vol 4. #2, março de 1987.

[Booc94] Grady Booch - *"Object Oriented Analysis and Design with Applications"*, The Benjamin/Cummings Publishing Company, Inc.,1994.

[Cima94] Alberto M. De Cima, Cláudia M. L. Werner, Alessandro A. C. Cerqueira - *"The Design of Object-oriented Software with Domain Architeture Reuse"*, Third International Conference on Software Reuse, Rio de Janeiro, 1994.

[Free87] Peter Freeman - *"A Perspective on Reusability"* em Tutorial: Software Reusability, (ed.) Peter Freeman, Computer society Press of the IEEE, 1987.

[Leit92] Júlio C. P. Leite, Antonio Prado, Marcelo Sant'Anna - *DRACO-PUC, experiências e resultados de re-engenharia de software*, VI Simpósico Brasileiro de Engenharia de Software, novembro 1992.

[Leit94] Júlio C.P.Leite, Marcelo Sant'Anna, F.G. de Freitas - *"Draco-PUC: A Techonology Assembly for Domain Oriented Software development"*, Third International Conference on Software Reuse, Rio de Janeiro, 1994.

[Matt95] Flavio Araujo de Mattos - *Uma Ferramenta de Reuso por Interconexão de Linguagens,* COPPE/UFRJ, abril de 1995.

[Neig89] James M. Neighbors, *"Draco - A Method For Engineering Reusable Software systems"*, Software Reusability, Volume I, Concepts and Models, (ed.) Ted J. Biggerstaff e Alan J.Perlis, ACM Press, 1989.

[Prie91] Rubén Prieto-Diaz - *"Implementing Faceted Classification for Software Reuse"*, Communications of the ACM, maio 1991.

[Prie93] Rubén Prieto-Diaz - *"Status Report: Software Reusability"*, IEEE Software, maio 1993.

[Vigo87] L. S. Vigotsky - *Pensamento e Linguagem* - Livraria Martins Fontes Editora Ltda, 1987.

[Wern92] Cláudia M. L. Werner - *Reutilização de Software no Desenvolvimento de Software Científico* - Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, março 1992.

[Wirf90] R.J. Wirfs-Brock, R.E. Johnson - "*Surveying Current Research in Object-Oriented Design*", Communications of the ACM, setembro 1990.