ORÁCULO: Um Sistema de Críticas para UML

Alexandre Ribeiro Dantas, Alexandre Luis Correa e Cláudia Maria Lima Werner
{alexrd, alexcorr, werner}@cos.ufrj.br
COPPE/UFRJ – Programa de Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro
Caixa Postal 68511 – CEP. 21945-970
Rio de Janeiro – Brasil

Resumo

Este artigo apresenta um mecanismo para verificação de críticas sobre modelos construídos durante o desenvolvimento de software, denominado Oráculo. As críticas existentes são baseadas nas regras de boa-formação da linguagem UML, e novos tipos de críticas podem ser facilmente adicionadas. São também analisadas a importância e a implementação de sistemas de críticas no contexto de reutilização de software.

Palavras-chave: Sistema de Críticas, Modelagem, UML, Reutilização.

1 – Introdução

A qualidade dos artefatos produzidos durante as fases iniciais de um processo de desenvolvimento de software é fundamental para o sucesso do decorrer do projeto e para a manutenção de sua viabilidade econômica. A maior parte dos erros de um sistema está associada às fases de análise de requisitos e projeto, e só são descobertos em etapas mais avançadas como codificação e testes (Kotonya e Sommerville, 1996).

Os principais artefatos das fases de análise e projeto são modelos que representam o problema sendo resolvido em um alto nível de abstração, permitindo um melhor entendimento das várias visões do projeto e servindo como base para as etapas seguintes do processo, como codificação e validação. Quando não há verificação da consistência e da correção dos modelos sendo construídos, em virtude de notações imprecisas ou suscetíveis a interpretações incorretas aliadas ao baixo conhecimento do domínio do problema, elevam-se o grau de incerteza sobre as especificações e chances de ambigüidades e erros serem introduzidos despercebidamente. Neste sentido, o objetivo deste trabalho é apresentar a utilização de modelos consistentes, segundo a *Unified Modeling Language* (OMG, 1999), como um fator relevante para o sucesso do desenvolvimento de software. Para uma efetiva modelagem consistente é apresentado um mecanismo de críticas que atua durante a construção dos modelos, verificando algumas da regras de boa-formação especificadas pela UML.

Este artigo está dividido em quatro seções. A primeira ilustra as motivações para o trabalho. A segunda apresenta os principais conceitos relacionados aos sistemas de críticas e algumas abordagens e ferramentas existentes. Na terceira seção é descrita uma proposta para a verificação da consistência dos modelos sendo construídos, segundo as regras estabelecidas pela especificação da linguagem utilizada. São analisadas também as principais características desejadas para um mecanismo de críticas. Em especial, é apresentada a infra-estrutura de desenvolvimento de software baseado em modelos de domínio - Odyssey - onde a proposta

foi implementada. Finalmente, são apresentadas conclusões, contribuições e perspectivas futuras.

2 – Sistemas de Críticas e suas Abordagens

Um sistema de críticas pode ser entendido como um mecanismo que atua sobre ferramentas de modelagem oferecendo correções e sugestões sobre os modelos sendo projetados. Duas abordagens são possíveis: a autoritária, que analisa o modelo segundo a presença ou ausência de determinada propriedade, e a informativa, que visa detectar potenciais problemas e auxilia o projetista a tomar as melhores decisões e a evoluir seu projeto (Robbins *et al*, 1998). Neste sentido, algumas abordagens de utilização desses sistemas são válidas para verificação de:

- construções sintáticas de uma linguagem de modelagem;
- heurísticas de bons projetos e suporte a tomada de decisões;
- consistência entre múltiplas visões de modelos.

A presença de um sistema de críticas sobre uma ferramenta de modelagem, é motivada por vários aspectos (Robbins *et al*, 1998), como:

- conhecimento limitado sobre o domínio de aplicação levando a erros;
- baixos custos de revisão imediata e altos custos de retrabalho nos erros;
- possibilidade de aprendizado contínuo;
- redução de tempo de desenvolvimento;
- melhor gerenciamento de riscos.

O uso de representações formais ou semiformais, pode ser visto como uma abordagem para obter e verificar a consistência de modelos. Entretanto, especificações formais são complexas em função do rigor matemático das linguagens, além de dificuldade de integração com ferramentas de desenvolvimento com suporte gráfico para a criação de modelos orientados a objetos. O uso de formalismos é motivado pela diminuição de ambigüidades, ganhos na consistência dos diagramas e especificações, correção através de provas formais, e refinamento de um modelo mais abstrato para um modelo implementacional correto (Evans *et al*, 1999). A OCL (*Object Constraint Language*), por exemplo, pode ser usada de forma a eliminar ambigüidades, adicionar semânticas de restrições, pré e pós-condições. A descrição da semântica estática da UML, através das regras de boa-formação, são representadas em OCL. A semântica dinâmica, entretanto, ainda permanece descrita através do uso de linguagem natural e ambígua. Além disto, através da OCL não há mecanismos de provas formais e validações rigorosas sobre os modelos.

Além das abordagens formais, podem ser utilizados sistemas de críticas. Os principais requisitos identificados para um mecanismo de críticas envolvem:

- estrutura hierárquica e flexível entre críticas e regras;
- possibilidade de ativação e desativação completa do mecanismo de verificação;
- configuração de críticas e regras que podem estar ativas ou inativas;
- resposta preemptiva às ações do usuário durante a modelagem;
- interação com usuário clara e explicativa para as violações verificadas;
- criação de relatórios assíncronos e completos de verificação dos modelos; e
- fontes independentes de ajuda ao usuário sobre a UML e suas regras.

Algumas ferramentas existentes atualmente no mercado procuram disponibilizar mecanismos de verificação de consistência de modelos. A ferramenta ArgoUML (Robbins *et al*., 1998) possui agentes que analisam críticas de projeto e possíveis melhorias, trabalhando transparentemente durante a edição. Estas críticas englobam erros sintáticos, pontos incompletos, guias de estilos e recomendações de projetistas experientes. Outras ferramentas como Rational Rose (Rational, 2001), Objecteering (Softeam, 2001) e SoftModeler (Softera, 2001), MagicDraw (NoMagic, 2001) também oferecem algum tipo de suporte. A tabela 1 exibe uma comparação destas ferramentas, com base nos fatores identificados nos requisitos.

Fator de Análise / Ferramenta		Argo	Magic	Object'ing	Rose	S'Modeler
Subconjunto da UML	Diagramas	☺	☺	☺	☺	☹
	Elementos	☺	☺	☺	☺	😐
	Propriedades	😐	☺	☺	☺	😐
Categorias das Críticas	Classes e Core	😐	☹	😐	☹	☹
	Casos de Uso	☺	☺	☺	☺	☺
	Estados	😐	☺	😐	😐	😐
Configuração	Desativação	☺	☹	☺	☹	☹
	Por Crítica	☺	☹	☹	☹	☹
	Expansibilidade	☹	☹	☹	☹	☹
Interação com Usuário	Relatório	😐	☹	☹	😐	☹
	Ajuda UML	😐	☺	☹	☺	☺
	Preemptiva	☹	☺	☺	☺	☺
	Explicativa	☺	☹	☺	☺	☺

Tabela 1 - Comparação entre as ferramenta

3 – Um Sistema de Críticas no contexto de Reutilização

A reutilização de software é uma aposta simples e poderosa para se atingir melhor produtividade no desenvolvimento através da construção de novos sistemas a partir do uso de qualquer artefato já produzido e utilizado em soluções de problemas similares. A criação de componentes é uma etapa também conhecida como desenvolvimento para reutilização, onde é realizada a engenharia de domínio. A utilização de componentes previamente criados, avaliados e armazenados é a etapa conhecida como desenvolvimento com reutilização (Werner, 1999). A análise de domínio é uma atividade envolvendo um conjunto de possíveis aplicações que pertencem a uma família ou domínio comum, compartilhando similaridades e diferenças essenciais, resultando em modelos de domínio. Um modelo para um sistema individual pode ser criado a partir do refinamento do modelo do seu domínio com os requisitos específicos para o sistema em questão. O uso de modelos de domínio e suas visões contendo inconsistências, ambigüidades e erros, multiplica os efeitos desastrosos sobre a produtividade e qualidade, pois afeta toda uma família de sistemas sendo desenvolvidos a partir de componentes comuns.

O Odyssey é uma infra-estrutura de reutilização que oferece ferramentas para apoio automatizado tanto para o desenvolvimento para reutilização (Odyssey-DE) quanto para o desenvolvimento com reutilização (Odyssey-AE). A infra-estrutura é composta por ferramentas que procuram automatizar as diversas etapas definidas pelo Odyssey-DE e Odyssey-AE (Werner *et al*, 2000). Podemos citar ferramentas como a para captura de

conhecimento de domínios, documentação de componentes, especificação e instanciação de arquiteturas específicas de domínios, planejamento e análise de risco, camada de mediação e navegador inteligente, gerador de código executável, acompanhamento de processos, engenharia reversa e, finalmente, a ferramenta de diagramação UML, objeto de atenção deste trabalho. Toda esta infra-estrutura é implementada utilizando-se a linguagem JAVA.

Um mecanismo de críticas foi implementado como uma ferramenta auxiliar à ferramenta de diagramação do Odyssey. Sua execução está relacionada a três ocasiões específicas de ação: inclusão, remoção e edição das propriedades de qualquer elemento de modelagem. O mecanismo se baseia na verificação de regras, que são agrupadas em determinados conjuntos de acordo com suas características. Estes conjuntos de regras são chamados de críticas. Foram implementadas algumas regras de boa-formação a partir da especificação da linguagem UML. As regras foram agrupadas em cinco categorias de críticas. A primeira envolve considerações sobre a validade dos nomes dos elementos. Há categorias próprias para regras do modelo de casos de uso, máquinas de estado e modelo de classes. Finalmente, uma categoria está relacionada às questões de visibilidade entre elementos. A tabela 2 exibe alguns exemplos desta categorização.

Crítica	***Exemplo de Regras***
Considerações de Nome	Nomes começam com letras, "$" e "_". Contém apenas dígitos, letras, "$" e "_", sem espaços brancos
Modelo de Casos de Uso	Casos de Uso não podem ter associações com Casos de Uso especificando a mesma entidade
Máquina de Estados	Um estado inicial não pode ter transições de chegada, e tem no máximo uma transição de saída
Modelo de Classes	Uma instância não pode pertencer por composição a mais de uma instância de composição
Questões de Visibilidade	Elementos importados e pertencentes a um *namespace* não podem ter o mesmo nome. No caso das associações, não pode haver a mesma combinação de nome conectando dois classificadores

Tabela 2 – Exemplos de Críticas e Regras

A figura 1 exibe o modelo conceitual do mecanismo de críticas. Todas as informações sobre as críticas e regras são armazenadas de forma flexível e independente do código fonte do ambiente. Para isto, é utilizado um arquivo descritor em XML, com *tags* que definem semanticamente a estrutura representada no modelo conceitual. Qualquer alteração ou expansão sobre o conjunto de verificações requer apenas alterações no arquivo descritor. Um gerente de críticas é a entidade responsável pelo controle de todo o mecanismo de verificação. Este gerente é formado por um conjunto de objetos de críticas, que são instanciados a partir de um objeto especial (*loader*), responsável por extrair as informações a partir do arquivo descritor. A execução do mecanismo é realizada pelo gerente de críticas, que delega a verificação para cada crítica e suas regras. A propriedade *agente* de um objeto crítica representa o objeto que contém todos os métodos de implementação algorítmica das regras. É utilizado o mecanismo de reflexão da linguagem JAVA para instanciar os agentes e invocar os métodos de cada regra.

Se uma regra for violada, o usuário é alertado e o conselho associado a regra é exibido, impedindo que a ação que causou a violação prossiga. A figura 2 mostra o mecanismo de verificação em ação para o caso de herança a partir de uma classe folha (ou final), uma das regras definidas pela especificação da UML.

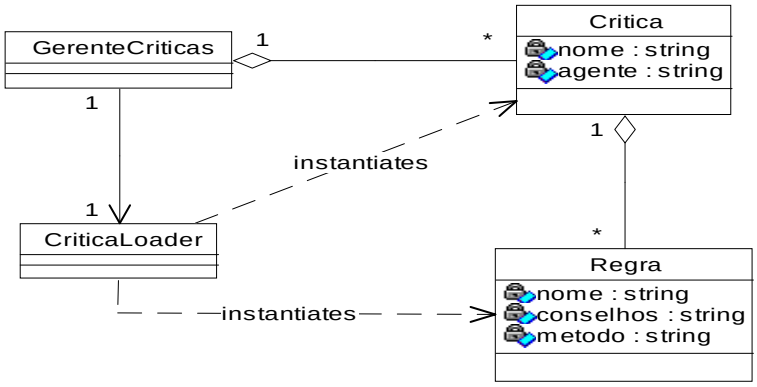

Figura 1 - Modelo Conceitual

O usuário pode configurar a granularidade da verificação a ser realizada. É permitida a completa desativação do mecanismo ou desativação de apenas determinadas regras ou conjunto de regras (críticas). Nestes casos, entretanto, inconsistências podem estar sendo criadas em diversos pontos dos modelos sem qualquer tipo de notificação. Dois componentes importantes para este caso são as fontes de ajuda e os relatórios. As fontes de ajuda permitem que o desenvolvedor adquira conhecimento sobre a linguagem e suas regras, diminuindo, desta forma, possíveis construções errôneas e ambigüidades. Na infra-estrutura Odyssey, esta ajuda é implementada através de documentos em hipertexto. A criação de relatórios é uma atividade assíncrona e não preemptiva, permitindo ao usuário uma visão completa dos estados de todos os modelos criados, detectando possíveis erros adicionados previamente.

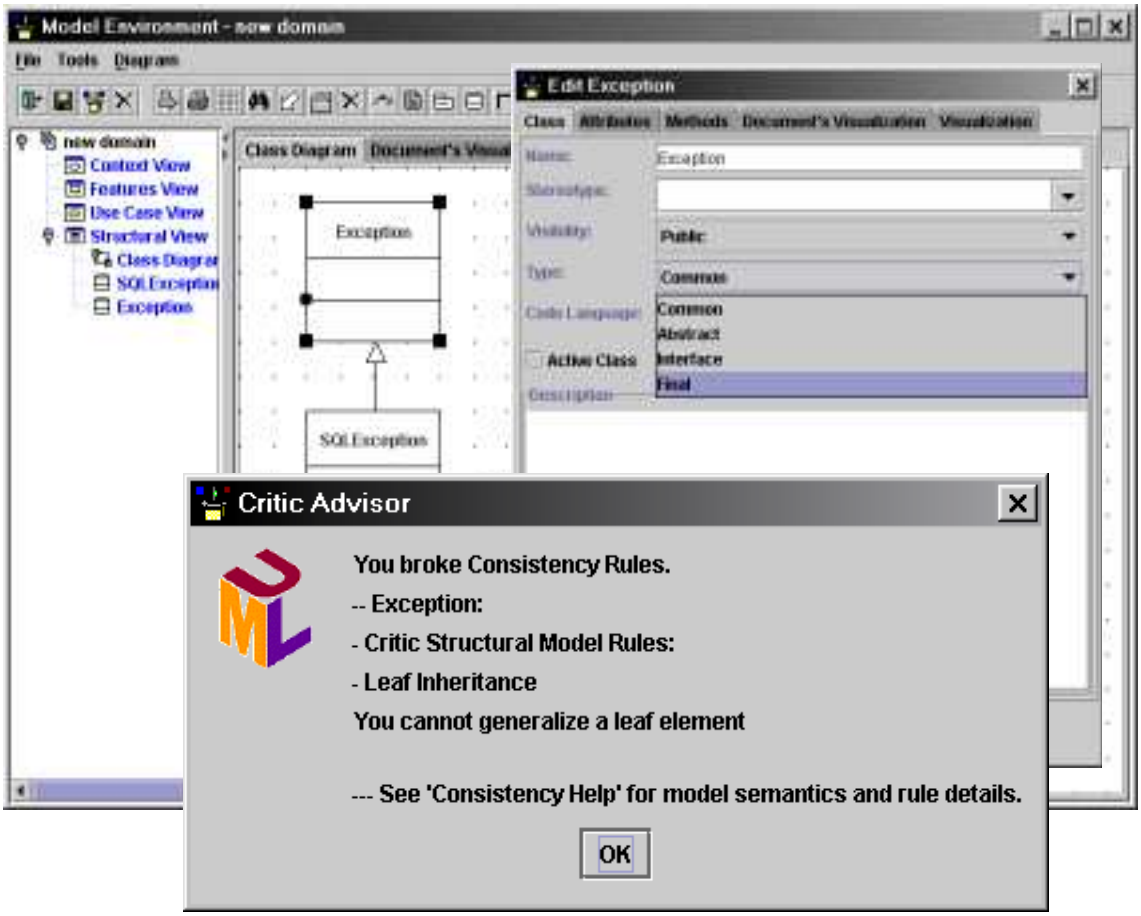

Figura 2 - Mecanismo de Críticas

4 – Conclusão

As principais contribuições deste trabalho envolvem a melhoria de qualidade de especificações e modelos de projetos de software, principalmente quando há criação de modelos como artefatos de reutilização em diversas aplicações desenvolvidas. A detecção antecipada de inconsistências, erros e ambigüidades diminui o esforço de retrabalho e os custos do desenvolvimento, garantindo a viabilidade do projeto. Em adição, os projetistas envolvidos também apresentam relativo ganho de aprendizado sobre a linguagem e suas regras para a criação de bons modelos e projetos.

Este trabalho focaliza apenas questões de consistência a partir de regras da linguagem UML, porém estas regras estão limitadas apenas a verificações notacionais, de sintaxe e semântica estática. Conhecimentos mais completos sobre a semântica dinâmica são apenas descritos de forma textual. O mecanismo de críticas proposto e implementado na infra-estrutura Odyssey, porém, é flexível para futuras expansões de verificações de modelos, além do escopo da linguagem de modelagem. Futuros trabalhos estão relacionados à criação de críticas baseadas em boas heurísticas no contexto de suporte a tomada de decisões de projeto, e a verificação de consistência entre as diferentes visões de modelos de domínio para apoiar o processo de reutilização.

Referências Bibliográficas

EVANS A., LANO K., FRANCE R., RUMPE B. *Meta-Modeling Semantics of UML*. IN: Behavioural Specifications for Businesses and Systems, Kluwer Academic Publishers, Editor: Haim Kilov, Chapter 4, 1999 .

KOTONYA G. SOMMERVILE I., *Requirements Engineering with Viewpoints*. Software Engineering Journal. Janeiro 1996.

MAGIC DRAW 4.0, *NoMagic* - http://www.nomagic.com/, Abril, 2001.

ODYSSEY, *COPPE/UFRJ* - http://www.cos.ufrj.br/~odyssey, Abril, 2001.

OMG. *OMG Unified Modeling Language Specification.* Versao 1.3. 1999. 808 p.

RATIONAL ROSE, *Rational* - http://www.rational.com/, Abril, 2001.

ROBBINS J.E., HILBERT D.M., REDMILES D.F. *Software Architecture Critics in Argo*. Formal Demonstration at the 1998 Conference on Intelligent User Interfaces (IUI'98).

SOFTMODELER Enterprise Edition 2.5, *Softera* - http://www.softera.com/, Abril, 2001.

WERNER, C., BRAGA, R., MATTOSO, M., *et alli. Odyssey: Estágio Atual.* Caderno de Ferramentas do XV Simpósio Brasileiro de Engenharia de Software (XIV SBES). João Pessoa, Brasil. Outubro, 2000.