Editor Cooperativo para Diagramação de Software OO

Márcio de Sousa Dias [1]
mdias@cos.ufrj.br

Geraldo Xexéo [1,2]
xexeo@cos.ufrj.br

Abstract

Complex software development requires an effective interaction among several specialists. This interaction can only be effective with the support of collaborative tools. This article presents a collaborative editor to support the diagramming activities of object oriented software modeling, describing the first version, where we exploited some synchronous aspects of the interaction.

Keywords: *Collaborative Software Design, CASE Tools, Groupware.*

1. Introdução

A fim de suportar efetivamente o trabalho em equipe durante o desenvolvimento de software, algumas das mais recentes ferramentas e ambientes de engenharia de software incorporam características de suporte à cooperação e coordenação. Apesar de uma das principais atividades no desenvolvimento de software orientado a objetos ser a criação e a manutenção de modelos OO, geralmente na forma de diagramas, poucas ferramentas oferecem suporte de cooperação diretamente relacionado às atividades de diagramação. Enquanto isso, na área de Trabalho Cooperativo Suportado por Computador (CSCW), não é difícil de se encontrar ferramentas para edição síncrona de desenhos geométricos, mas estes não oferecem suporte a qualquer técnica específica de diagramação de software OO.

2. O Editor Cooperativo

O principal objetivo deste editor é suportar aspectos de cooperação na modelagem de software OO, focando-se nas atividades de diagramação. Esse suporte envolve alguns aspectos, discutidas a seguir, tais como a notação de modelagem adotada, controle de concorrência, comunicação e percepção (*awareness*) entre membros da equipe.

Cada especialista envolvido no desenvolvimento de um modelo de software possui diferentes graus de conhecimento e experiência em relação às técnicas de modelagem OO. Uma característica que pode facilitar a cooperação e a participação desses especialistas é permitir que um mesmo modelo possa ser visualizado através de notações de diferentes técnicas. Apesar das diferenças entre os métodos OO, a semântica envolvida é similar. Essa similaridade se deve à convergência dos métodos OO elaboracionais [Martin96], como pode ser percebido pela atual tendência de unificação de métodos, com propostas tais como Rational™ UML [Rational96] e OPEN Methodology [OPEN96].

Assim, procuramos separar a semântica do modelo da notação, utilizando um conjunto extensível de elementos semânticos, derivados do método UML. De forma semelhante ao padrão de projeto Model-View-Controller [Gamma94], a notação é responsável por mapear os elementos semânticos do modelo em sua representação visual.

---

[1] Programa de Engenharia de Sistemas e Computação - COPPE / UFRJ

[2] Departamento de Ciência da Computação - IM / UFRJ

Durante a edição cooperativa, é necessário que mecanismos de coordenação possam impedir que vários usuários editem sincronamente uma mesma característica. Apesar de existirem características que necessitem de algum mecanismo de controle de concorrência, através de, por exemplo, comandos *lock/unlock*, esse controle pode ser prejudicial para uma edição cooperativa de outras características.

Identificamos duas camadas distintas no documento para o controle de concorrência: a informação conceitual e a representação visual. Quando o usuário edita um elemento, o comando de *lock* atua apenas sobre a informação conceitual, permitindo que um usuario, por exemplo, modifique a posição e o tamanho de elementos bloqueados por outros usuários. Além disso, os elementos podem possuir diferentes graus de prioridade de *lock*. Por exemplo, quando um relacionamento estiver bloqueado, as classes envolvidas não podem ser removidas.

Além da comunicação indireta proporcionada pelo compartilhamento do documento em edição, é importante haver suporte à comunicação direta, desde simples mecanismos de troca de mensagens textuais até sistemas de conferência. Neste editor, mecanismos simples de troca de mensagem textual são oferecidos para permitir uma melhor cooperação entre os usuários.

Assim como a comunicação, a percepção é muito importante para melhorar a eficiência e compreensão do processo de cooperação [Borges97], evitando uma edição "competitiva". Desta forma, todos os usuários devem perceber imediatamente qualquer alteração na sessão (por exemplo, entrada e saída de outros usuários na sessão) e no documento (conteúdo e estado dos elementos).

3. O Protótipo

A fim de permitir que a ferramenta seja utilizada pelos usuários em seu próprio ambiente de trabalho, é importante uma implementação em uma linguagem independente de plataforma. Apesar de haver várias linguagens deste tipo no meio acadêmico, somente Java parece ter obtido ampla aceitação comercial. O atual protótipo, desenvolvido com JDK 1.0.2 [JavaSoft96], suporta trabalho síncrono em equipe.

Possuindo um arquitetura cliente-servidor (Figura 1), o editor tem seu protocolo de comunicação baseado em mensagens textuais ASCII através de *streams* TCP-IP, a fim de facilitar extensões, a evolução do protocolo e a integração com outras ferramentas [Comer93].

O servidor mantém uma base centralizada com uma cópia de cada documento compartilhado e, através do gerente de cooperação, controla a edição concorrente dos documentos. Cada cliente mantém uma cópia consistente do documento. Enquanto o servidor gerencia a parte semântica do modelo, o cliente é que se responsabiliza pela representação visual do modelo.

O servidor inicia carregando o modelo, disparando o gerente de cooperação e aceitando conexões por *socket* dos clientes. Quando uma aplicação cliente se conecta ao (ou desconecta do) servidor, este avisa os outros clientes sobre a presença desse usuário. O servidor recebe dos clientes comandos para criar, modificar, remover, bloquear/desbloquear (*lock/unlock*) e mover os elementos. Ao realizar alguma tarefa, o servidor avisa os clientes sobre as alterações ocorridas, a fim de manter os documentos consistentes e os membros da equipe cientes da operação. Há ainda outros comandos que permitem a comunicação entre os usuários.

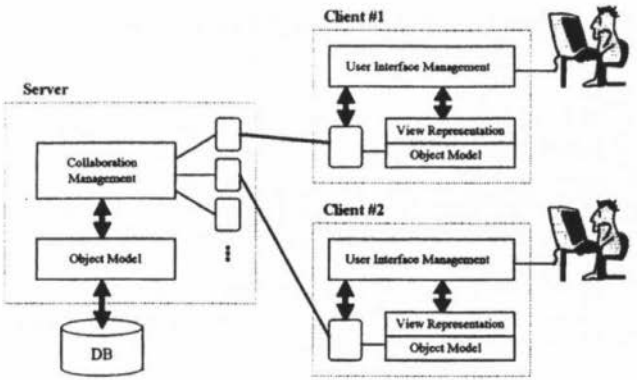

Figura 1 - Arquitetura Cliente-Servidor do Editor

Durante a conexão do cliente com o servidor, as informações sobre o contexto de trabalho atual são carregados pelo cliente. Com a cópia do modelo de classes, o cliente cria a representação visual de acordo com a notação do método. Ele utiliza mecanismos simples de percepção, permitindo visualizar os participantes da sessão e o estado dos elementos bloqueados através de cores (Figura 2). Permite ainda duas formas de comunicação entre os membros: sistema de mensagem e de conversa (*chat*). A mensagem é útil para comunicações unidirecionais ou não-interativas, pois se baseia em uma comunicação sem contexto histórico, apenas abrindo uma caixa de diálogo com o comunicado. Já o sistema de conversa mantém o histórico da conversação, permitindo uma melhor troca de informações (Figura 3).

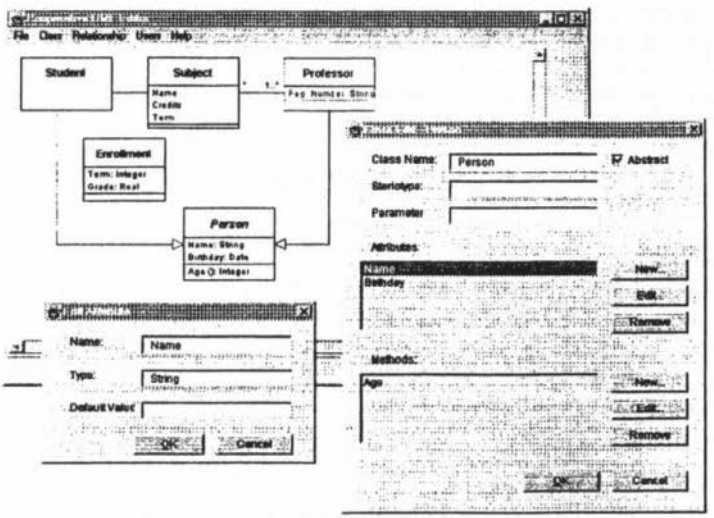

Figura 2 - Classe Person está sendo editada, enquanto a classe Professor está bloqueada. Diferentes modos de visualização podem ser percebidos nas classes Student (mínimo), Subject (parciál) e Professor (total).

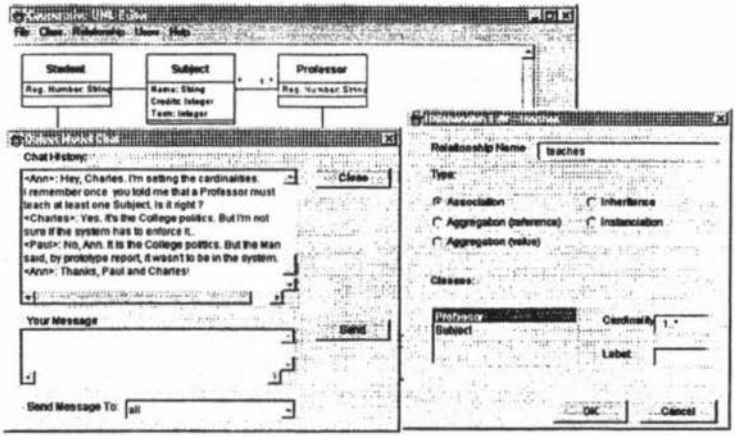

Figura 3 - Comunicação através do Sistema de Conversa

4. Conclusões

Este editor aparece como um experimento em direção a ferramentas de suporte ao desenvolvimento cooperativo de software. Apesar do editor se deter em aspectos síncronos da atividade de diagramação, considerou, ainda que de forma simples, questões importantes para a atividade cooperativa, como comunicação, coordenação e percepção. Aspectos assíncronos e outras questões também devem ser explorados em um ambiente que se proponha ao desenvolvimento cooperativo de software, como, por exemplo, a presença de uma memória de grupo que disponibilize a realização do processo de cooperação (por exemplo, registro das tomadas de decisão) junto aos documentos de artefato de software, permitindo também a reutilização de decisões de projeto (*design rationale*).

5. Agradecimentos

Agradecemos à CAPES e ao CNPq pelo suporte financeiro, e ao Prof. Marcos Borges por discussões valiosas sobre o assunto.

6. Referências

[Borges97] Borges, M.; "Awareness Mechanisms for SISCO", http://www.nce.ufrj.br/ ~mborges/sisco/

[Campione96] Campione, M.; Walrath, K.; "The Java Tutorial: Object-Oriented Programming for the Internet", Sun Microsystems, http://www.javasoft.com

[Comer93] Comer, D.E.; Stevens, D.L.; "Internetworking with TCP/IP Volume III: client-server programming and applications -BSD Socket Version", Prentice-Hall, 1993.

[Dias97] Dias, M; Xexéo, G.; "Supporting Software Deging with a Collaborative Editor", Proceddings of SCI'97, Caracas/Venezuela, July, 1997.

[Ellis91] Ellis, C.A.; Gibbs, S.J.; Rein, G.L.; "Groupware: Some issues and experiences", Communications of the ACM 34(1), 1991, pp. 39-58.

[Gamma94] Gamma, E. et al.; "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Massachusetts, 1994.

[JavaSoft96] JavaSoft Home Page, Sun Microsystems Inc., http://www.javasoft.com/

[Martin96] Martin, R.; "A Translational Vs. Elaborational View of Methodology", Object Currents, SIGS Publications, February 1996.

[OPEN96] OPEN Consortium, "Proposing an open standard", Object Expert, 2(1), Nov/Dec 1996.

[Rational96] Rational Soft. Corp., "Unified Modeling Language", http://www.rational.com/