Estudos

MAC5701 - Tópicos em Ciência da Computação

Modelo de Persistência baseado em Padrões Semânticos para Objetos Comportamentalmente Completos.

Marcos Eduardo Bolelli Broinizi
broinizi@ime.usp.br

Orientador:

João Eduardo Ferreira

Dept. de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo
jef@ime.usp.br

19 de junho de 2004

Resumo

O problema de persistência de objetos comportamentalmente completos [19] será abordado baseando-se em padrões semânticos, como a abordagem canônica para abstração de generalização [5]. Cada objeto comportamentalmente completo não apenas conhece os atributos da entidade do domínio que este representa, mas também sabem como modelar o comportamento dessa entidade. Dessa forma todos os comportamentos associados a um objeto, que são necessários à aplicação em desenvolvimento, devem pertencer a esse objeto e não implementados em um outro lugar dentro do sistema.

Nosso objetivo nesse projeto será formalizar um modelo de persistência baseado em padrões semânticos para objetos comportamentalmente completos, estudar sua implementação sobre o modelo relacional, e estabelecer um protótipo de implementação, seguindo o modelo formalizado, interagindo com o *framework* chamado *Naked Objects* [17], já existente, para utilização de objetos com tais características.

1 Motivação

Os objetos pelados são objetos comportamentalmente completos denominados assim por Pawson e Matthews. Em [19] os autores defendem que a abordagem tradicional para desenvolvimento de sistemas de negócio falha em integrar os dados e processos dos objetos, enquanto os objetos pelados são capazes de unir tais aspectos e encapsular o verdadeiro objeto de negócio que eles representam.

Consideramos a capacidade de prototipação de sistemas de negócios baseada em objetos comportamentalmente completos um método eficiente de identificar as entidades de negócio do sistema. Visamos então prover um método eficiente de permitir a criação de um protótipo do modelo conceitual e seu mapeamento para o modelo relacional.

2 Problema

Propomos rever o problema de persistência de objetos baseando-se na perspectiva de padrões semânticos. Muitas são as soluções implementadas atualmente para prover persistência para objetos [4, 12, 8], porém a maioria dessas soluções concentram seus esforços em persistir os objetos mais genéricos possíveis criados em uma determinada linguagem. Nossa abordagem visa sobretudo uma modelagem conceitual adequada. Para tal estenderemos a idéia de representação de objetos comportamentalmente completos aumentando o valor semântico dos dados. A aplicação dos padrões semânticos forneceria um protótipo adequado do modelo conceitual da base de dados específica, facilitando o projeto do sistema, sobretudo da base de dados.

3 Objetivos

Nosso objetivo principal nesse projeto será formalizar um modelo de persistência baseado em padrões semânticos para objetos comportamentalmente completos. Para isso estudaremos formalizações de objetos complexos e seus relacionamentos [16, 1, 14], padrões semânticos e seu mapeamento para o modelo de dados relacional e o conceito de objetos comportamentalmente completos.

Como objetivos secundários estudaremos a aplicação desse modelo de persistência sobre o modelo relacional e estabeleceremos um protótipo de implementação capaz de interagir com o *framework Naked Objects*. Para isso estudaremos este *framework* específico assim como outras soluções implementadas para persistência de objetos.

As contribuições esperadas para este trabalho são:

- Estudo detalhado de padrões semânticos e seu mapeamento para o modelo de dados relacional;
- estudo detalhado do conceito de objetos comportamentalmente completos e sua relevância;
- estudo detalhado da implementação do *framework Naked Objects*;
- formalização do modelo de persitência baseado em padrões semânticos para objetos comportamentalmente completos;
- investigação da possibilidade de aplicação do modelo formalizado sobre o modelo de dados relacional;
- estabelecimento de um protótipo de implementação do modelo formalizado interagindo com o *framework Naked Objects*.

4 Plano de estudos

Para a disciplina MAC5701 esperávamos estudar a bibliografia citada, além de identificar outras bibliografias relevantes visando avançar sobre os seguintes objetivos, além de bibliografias sobre formalização de objetos complexos:

- Estudo detalhado de padrões semânticos e seu mapeamento para o modelo de dados relacional;
- estudo detalhado do conceito de objetos comportamentalmente completos e sua relevância;
- estudo detalhado da implementação do *framework Naked Objects*.

Esse relatório cobrirá nosso avanço nos seguintes aspectos:

- Estudo do conceito de objetos comportamentalmente completos e sua relevância;
- estudo da implementação do *framework Naked Objects*;
- estudo inicial sobre objetos complexos.

O estudo de padrões semânticos e seu mapeamento para o modelo de dados relacional não será abordado com detalhe neste trabalho como o previsto inicialmente uma vez que as poucas referências estudadas, com permissão dos autores, ainda não foram oficialmente publicadas. Além disso muito ainda deve ser pesquisado nesse aspecto e voltaremos nosso esforço para realizar isso no próximo semestre.

5 Objetos comportamentalmente completos e o *Framework* Naked Objects

Um objeto comportamentalmente completo não apenas conhece os atributos da entidade do domínio que este representa, mas também sabe como modelar o comportamento dessa entidade. Dessa forma todos os comportamentos associados a um objeto, que são necessários à aplicação em desenvolvimento, devem pertencer a esse objeto e não implementados em algum outro lugar do sistema.

Essa noção retoma as origens de orientação a objetos com Simula [3], cujos inventores trouxeram a idéia de construir sistemas a partir de objetos, cada um representando algum elemento do domínio a ser simulado. Uma simulação envolvia diversas classes de objetos, que representavam *templates* a partir dos quais eram criadas instâncias individuais de acordo com a necessidade da simulação.

A palavra *encapsulamento* é normalmente usada neste contexto. A palavra tem dois significados em Inglês. O primeiro tem a ver com existência fechada, como uma cápsula medicinal. Este é o significado que muitas pessoas usam no contexto de orientação a objetos: um objeto é fechado por uma interface de mensagem, com uma implementação interna escondida. Este é uma propriedade importante de objetos, mas não é a única. As idéias de operação caixa-preta e *"ocultamento de informações"* são comuns em muitas formas de desenvolvimento de sistemas baseados em componentes. O segundo significado de encapsulamento é que alguma coisa exemplifica as características essenciais de outra coisa, como em *"este documento encapsula nossa estratégia de marketing"*. Este segundo significado, que corresponde ao princípio de completeza comportamental, é muito mais importante no contexto de modelagem orientada a objetos.

Segundo Pawson e Mathews esse princípio é essencial porque é a chave para obter um dos principais benefícios da orientação a objetos: a habilidade de lidar com as mudanças inesperadas nos requisitos.

Em [19] os autores afirmam que as pessoas acham que estão usando a orientação a objetos para projetar e desenvolver sistemas; mas não estão. As pessoas projetam sistemas que separam o processo de seus dados, embora revestidos com linguagens e tecnologias orientadas a objetos. A separação dos processos de seus dados pode estar relacionada principalmente a inércia, ou seja, as pessoas aprenderam a projetar sistemas dessa forma e têm dificuldades de pensar de outra maneira.

A inércia individual é reforçada por práticas organizacionais consagradas que forçam a separação dos processos de seus dados, mesmo que o projetista de software queira adotar uma abordagem mais pura de orientação a objetos:

- Orientação a processos de negócio.
- Interfaces de usuário otimizadas a tarefas.
- Métodos orientados a use-cases.
- O padrão Model-View-Controller.
- Desenvolvimento de software baseado em componentes.

Essas cinco forças são consideradas práticas consagradas de orientação a objetos então não se pode simplesmente descartá-las. São práticas conscientes que claramente geram benefícios e foram projetadas para reduzir riscos conhecidos no processo de desenvolvimento. Porém, essas práticas provocam o efeito colateral de desencorajar o projeto de objetos comportamentalmente completos.

5.1 Orientação a processos de negócio

A idéia de modelar as atividades de negócio em termos de processos tornou-se popular no início de 1990 com a idéia da reengenharia de processos de negócio [7]. Depois de sair da moda no final de 1990, o pensamento de processos de negócio experimentou sua segunda onda, abastecido em parte pela emergência de uma nova geração de modelagens e ferramentas de gerenciamento de processos de negócio.

A orientação a processos envolve duas idéias: a primeira é a de focar, organizar e alcançar os resultados definidos externamente(tais como preencher um pedido) ao invés de atividades puramente definidas internamente. Sendo esta uma idéia útil para sistemas orientados a objetos; a segunda idéia é a de que processos podem e devem ser reduzidos a um processo determinista que transforma entradas em saídas.

O problema dessa abordagem é que muitas tarefas de negócio que devem ser modeladas, simplesmente não se encaixam no modelo de processos [2]. Existem várias atividades onde não se pode identificar entradas e saídas discretas, muito menos os passos seqüenciais que realizam tal trasnformação.

Historicamente muitos profissionais de tecnologia têm definido um sistema de informação como um mecanismo de transformar informações de entrada em informações de saída através da aplicação sucessiva de pequenas transformações. Isso já pode ter sido uma descrição precisa, mas é uma maneira muito pobre de descrever as capacidades modernas dessa área.

A metáfora que compara o papel dos sistemas de informação com uma linha de produção somente adiciona problemas. Sendo necessário buscar novas metáforas para se descrever um sistema de informação.

O modelo da linha de produção está vinculado a idéia de cadeia de valor identificado por Michael

Porter(1985) [20]. Na cadeia de valor, o produto é o meio de transferência de valor entre a empresa e seus clientes. O grande problema é que a cadeia de valor é visto por muitos como um modelo universal de negócio, um modelo que reforça um estilo de pensamento orientado a processos. Por sorte, a sua validade universal tem sido fortemente desafiada.

Charles Stabell e Oystein Fjeldstadt [21] dizem que a cadeia de valor é atualmente um modelo de negócio muito pobre, e seu uso pode conduzir a falhas perigosas no nível estratégico. Eles também argumentam que a proporção do negócio compatível com o modelo de cadeia está decaindo rapidamente assim como afirmam que o conceito de cadeia de valor é melhor aplicado para a análise de setores industriais que produzem bens manufaturados e homogêneos. No entanto, para determinados setores da economia, especialmente o setor de serviços, a lógica de criação de valor é diferente dos setores industriais de produção de bens manufaturados. Para a análise da indústria de serviços, o conceito de cadeia de valor não se configuraria como instrumento adequado para análise das fontes de vantagem competitiva. Eles sugerem que existem dois mecanismos diferentes, além da cadeia de valor, através dos quais negócios criam valor:

- loja de valor e
- rede de valor.

A loja de valor (tal como consultorias, construtores e principalmente organizações de saúde) cria valor aplicando recursos para resolver problemas individuais. Suas atividades raramente seguem uma seqüência linear e são freqüentemente iterativos por natureza. Os fornecedores de serviços utilizam as chamadas tecnologias intensivas para personalizar seus serviços às necessidades dos clientes. A distinção principal em relação ao conceito de cadeia de valor é que as empresas que prestam serviços tendem a trabalhar em tempo real para entregar aos seus clientes novas soluções ao invés de definirem uma solução e a reproduzem continuamente. Essas empresas selecionam, combinam e ordenam a aplicação dos seus recursos e de suas atividades de acordo com os requisitos do problema.

A rede de valor (que inclui muitas companhias de telecomunicações, bancos e seguradoras) cria valor pela venda de clientes a outros clientes. Sendo a configuração de valor baseada na reunião de diversos compradores e vendedores independentes, por uma empresa que atua como intermediária, através das tecnologias de mediação. Tecnologias que facilitam os relacionamentos entre os consumidores distribuídos no tempo e no espaço.

Ao invés de apenas imaginar o papel dos sistemas de negócio como um meio de executar um processo determinístico, que transforma informações de entrada para informações de saída através de uma seqüência de passos que adicionam valor, precisamos encontrar metáforas alternativas. Como a da loja de valores, onde o usuário constrói uma solução para um problema específico. (sendo muito fácil adicionar roteiros/ scripts otimizados para um modelo de cadeia de valores).

5.2 Interfaces de usuário otimizadas a tarefas

Muitas interfaces de usuário são projetadas para implementar um conjunto finito de tarefas programadas. Isso está implícito no método de projeto de interfaces, como também fica explícito no projeto resultante: comumente um menu de tarefas, que permite ao usuário escolher a tarefa desejada selecionando as sub-opções. Um estilo de interface de usuário conveniente para desenvolvedores de sistemas, pois é fácil de realizar o mapeamento para um conjunto de transações definidas, as quais uma após a outra manipulam suas estruturas de dados.

Uma idéia por trás desta abordagem é a de que um conjunto de ações programadas é a chave para a otimização. Essa idéia pode ser rastreada diretamente recordando Frederick Winslow Taylor [22] e seus princípios de gerenciamento científico.

Sua busca por *"uma melhor maneira"* de trabalho começou com a própria tecnologia. Na casa de máquinas onde ele trabalhou, conduziu milhares de experimentos para encontrar a velocidade ótima de corte para a operação de vários tipos de máquinas. Seus resultados mostraram significativos melhoramentos na média de produtividade comparado aos métodos gerais usados pelos trabalhadores.

Seu próximo alvo foi a organização do trabalho. Novamente, usando milhares de experimentos, ele tentou encontrar a resposta ótima para todos os aspectos de organização do trabalho manual, desde o tamanho ótimo de uma pá à freqüência ótima de interrupção de trabalho. Para encorajar os trabalhadores a adotarem suas melhores práticas, ele defendeu a introdução da diferença percentual de peças, onde o percentual adicional de pagamento elevava-se com o seu desempenho.

Taylor ainda não ficou satisfeito. Os trabalhadores usavam suas técnicas, mas não consistentemente. A solução foi remover todos os direitos de decisões dos trabalhadores criando roteiros de todas as suas ações. Para que eles obedecessem exatamente seus comandos, executando a seqüência de tarefas como ele havia definido, organizou os locais de trabalho e disse: *"Trabalhem quando eu ordenar, como ordenar e descansem quando eu ordenar. Isso irá aumentar o percentual geral de pagamento."* A quantidade de aumento desse pagamento foi apenas um outro valor a ser determinado cientificamente e girou em torno de trinta e cinco por cento.

O biógrafo Robert Kanigel chamou isso de *"ato Faustiano"*, aquele que vende a sua alma ao diabo [13]. Segundo ele Taylor acreditava que estava motivando os trabalhadores, no sentido específico de motivá-los a obter uma vida melhor. Mas no sentido moderno da palavra, ele foi evidentemente um desmotivador. As instruções programadas eram fornecidas em cartões indexados. Especulando como ele iria reagir com as modernas tecnologias de informação: suspeita-se que iria se deleitar pelas capacidade de estender seus métodos tanto em largura quanto em profundidade. Muitos sistemas de negócio atuais tratam o usuário como um mero seguidor de roteiros. O sistema controla todo o processo, subcontratando o usuário somente para aquelas subtarefas que ele não está capacitado a realizar autonomamente.

A abordagem alternativa é projetar sistemas que tratem os usuários como solucionadores de problemas. Muitos negócios já possuem alguns sistemas que são, por natureza, problemas a serem resolvidos como programas de desenho, do PowerPoint aos sistemas CAD/CAE, assim como as planilhas eletrônicas. No entanto, na maioria dos negócios, esses sistemas não são considerados *tradicionais*. Os sistemas tradicionais estão normalmente preocupados com o processamento padronizado das transações de negócio, e são otimizados para um conjunto de tarefas, os quais são quase sempre implementados como processos seqüenciais.

Muitas pessoas acham que sistemas de problemas a serem resolvidos e sistemas transacionais refletem duas necessidades muito diferentes dentro do negócio, que não existe necessidade de uní-los, e que fazer tal coisa só iria tirar a otimização do processamento das tarefas padrão que representam a maior parte das atividades de negócio. Os autores sugerem que existe uma necessidade muito real de trazer as duas idéias próximas uma da outra, ou seja, tornar os sistemas transacionais tradicionais tão *expressivos* quanto um programa de desenho [18].

Mas segundo os autores colocar estas duas idéias juntas não significa apenas colocar interfaces de usuários gráficas em sistemas transacionais tradicionais, como muitas empresas aéreas descobriram a muito custo, a primeira geração dos sistemas de reservas baseados em GUI foram menos expressivos do que as velhas interfaces de linha de comando, que apesar de poderem ser difíceis de aprender, permitiam ao usuário um alto grau de controle.

Experiências mal sucedidas com interfaces sugerem conclusões erradas de que *"usuários realmente não gostam de interfaces gráficas"*. Os autores afirmam que os usuários realmente não gostam é de interfaces gráficas ruins. Um outro preconceito é que introduzir mais expressividade num sistema transacional tradicional

irá reduzir sua eficiência. Isso pode ser verdade num contexto reduzido de tarefas padrões específicas, mas no sentido amplo a eficiência pode, na verdade, aumentar.

Ao invés de perseguir a eficiência ótima na execução de cada um dos conjuntos finitos de tarefas roteirizadas, deve-se projetar uma forma de interação com o usuário que maximize a efetividade geral do usuário em satisfazer sua gama de responsabilidades.

5.3 Métodos orientados a Use-Cases

O conceito de use-case foi definido por Ivar Jacobson [11] como *"uma seqüência de transações num sistema cuja tarefa produz um valor mensurável para um particular ator do sistema"*. Na abordagem de desenvolvimento de sistemas orientado a use-case, os use-cases que capturam os requisitos de um sistema são descritos e, então, procura-se identificar objetos comuns entre eles. Os métodos orientados a objetos mais populares são orientados a use-cases.

Existem diversos problemas quando se utiliza essa abordagem como resumiu Don Firesmith [6]: "use-cases não são orientados a objetos; cada use case captura uma grande abstração funcional que pode causar inúmeros problemas associados à decomposição funcional que a tecnologia de objetos supostamente deveria evitar; e uma vez que os use-cases são criados antes que os objetos e classes tenham sido identificados, use-cases ignoram o encapsulamento de atributos e operações em objetos. Afirmando ainda que uma abordagem orientada a use-case resulta em um protótipo arquitetural de subsistemas, com um objeto de controle funcional isolado representando a lógica de um use-case individual,e vários objetos de entidade com pouca funcionalidade controlados pelos objetos controladores. Tais arquiteturas exibem, normalmente, encapsulamento pobre, excessivo acoplamento, e uma distribuição inadequada da inteligência da aplicação entre as classes."

Jacobson observou, no entanto, que use-cases servem ao propósito de testar o sistema resultante, constituindo uma excelente ferramenta para testes de integração desde que explicitamente interconectam várias classes e blocos. "Quando todos os use-cases tiverem sido testados (em vários níveis) o sistema estará testado em sua totalidade" [10].

Os autores sugerem que use-cases são muito poderosos quando descritos em termos de operações sobre objetos que já tenham sido identificados e especificados, e usados para testar esse modelo de objetos. Por outro lado, use-cases são muito perigosos quando descritos antes do modelo de objetos e usados para identificar objetos e suas responsabilidades compartilhadas, como defende a abordagem orientada a use-cases.

Os autores sugerem identificar os objetos de negócio através de conversas diretas e não estruturadas entre usuários e desenvolvedores. Sendo o necessário para isso um meio correto de capturar um modelo de objetos emergente na forma de um protótipo de trabalho, onde tanto usuários quanto desenvolvedores possam entender e contribuir. Não significando o desenvolvimento de um protótipo convencional que capture os requisitos das tarefas do usuário em termos de formulários e menus mas sim um protótipo com uma interface de usuário orientada a objetos, onde o usuário possa ver na tela, a representação direta dos relacionamentos existentes entre os objetos de negócio fundamentais, não apenas com relação à atributos e associações, mas também comportamento.

5.4 O padrão Model-View-Controller

Um padrão arquitetural comum que explicitamente separa três papéis: os objetos de negócio fundamentais que correspondem às entidades de negócio são o Modelo(*Model*), os objetos que fornecem a visão do modelo ao usuário formam a Visão(*View*) e os objetos que controlam a interação entre o usuário e o modelo, Controlador (*Controller*).

Uma versão deste padrão é o *Model-View-Controller*, ou MVC [15]. Um outro é *Entity-Boundary-Controller* [9].Existem distinções sutis entre esses dois padrões, mas eles são, de modo geral, similares. Os autores utilizam o MVC como referência.

A motivação para usar o MVC é a separação de conceitos. O argumento do MVC é que os objetos de negócio fundamentais de uma classe de objetos de negócio fundamental, podem ser visualizados de várias maneiras diferentes: sobre diferentes plataformas; em diferentes contextos; usando diferentes representações visuais. O conhecimento combinado de todas essas visões diferentes, bem como o conhecimento de como afetá-las dentro de objetos de negócio faz com que os objetos fiquem inchados e pesados devido à duplicação de funcionalidades nesses objetos.

Usando MVC, os objetos do Modelo não têm conhecimento dessas diferentes visões. Objetos de Visão dedicados especificam o que aparece em cada visão, e de que forma, e são capazes de criar a apresentação visual. Objetos Controladores realizam a intermediação, povoando as visões com atributos a partir dos objetos entidade do negócio, e chamando métodos desses objetos em resposta a eventos do usuário.

Embora não tenha sido a intenção original da abordagem MVC, os objetos Controladores tendem a ser uma representação explícita das tarefas de negócio, especialmente se a abordagem de projeto for orientada a use-case, mas isso ocorre também em outros casos. Esses objetos deixam de exercer o papel limitado de ser apenas intermediadores entre interfaces de usuários e objetos de negócio, e passam a assumir o papel de roteiro de tarefas, incorporando não apenas a seqüência de atividades otimizadas, como também regras de negócio, com isso usurpando as responsabilidades que deveriam ser dos objetos de negócio fundamentais. O resultado final é que o conhecimento específico de negócio é espalhado através dos domínios do Modelo, Visão e Controlador.

Qualquer mudança no Modelo de objetos fundamentais irá potencialmente exigir mudanças num grande conjunto de objetos Visão e Controlador. No entanto, não existe nada no MVC que force esta tendência, mas a prática sugere que devemos contestá-lo veementemente quando procuramos por objetos comportamentalmente completos.

Uma abordagem alternativa deve evitar cair no problema para o qual o MVC foi projetado para evitar: devendo facilitar a portabilidade de uma aplicação entre múltiplas plataformas técnicas, até entre múltiplos estilos de interação, sem necessitar que o modelo de negócio seja editado; ao mesmo tempo, ela deve acomodar a necessidade de representar múltiplas visões do modelo sob a mesma plataforma, onde genuinamente exista tal necessidade.

Os autores sugerem fornecer um mecanismo genérico de visualização, incorporando os papéis dos objetos de Visão e Controlador. Isso significa escrever um mecanismo de visualização para cada plataforma de cliente solicitada (por exemplo: Windows, Linux, browser web ou Palm Pilot). Uma vez que um mecanismo de visualização genérica exista para a plataforma alvo, tudo que o desenvolvedor precisa é escrever os objetos do Modelo de Negócio. Essa abordagem não viola a essência do MVC, mas é uma re-interpretação radical de como aplicá-la. Uma maneira de considerar seria que ela gera os objetos de Visão e Controlador considerando o Modelo.

5.5 Desenvolvimento de sistemas baseados em componentes

Os autores não são contrários à idéia de desenvolvimento de sistemas baseados em componentes, mas estão preocupados com a maneira em que esta idéia consegue confundir o projeto orientado a objetos. Certamente eles possuem elementos comuns, como quaisquer duas idéias possuem elementos comuns no desenvolvimento de software. Mas o projeto orientado a objetos e desenvolvimento de sistemas baseado em componentes são conceitos um tanto diferente.

O desenvolvimento baseado em componentes está principalmente interessado em fornecer uma abordagem *plug-and-play* para o desenvolvimento de sistemas. O *plug-and-play* possibilita, em teoria, maior flexibilidade no fornecimento de sistemas, permitindo a compra de componentes no mercado, a copia de uma biblioteca pública de componentes, ou reutilização de componentes escritos internamente tendo em mente várias aplicações. Essa flexibilidade pode potencialmente economizar gastos diretos, reduzir o esforço de desenvolvimento, elevar a qualidade e promover a padronização.

A modelagem orientada a objetos não deveria estar preocupada com o *plug-and-play*. Deveria sim estar preocupada em adequar a estrutura do software à estrutura do domínio de negócio do mundo real, seja ocasionalmente em resposta às mudanças nos requisitos de negócio, ou dinamicamente em resposta a um problema específico.

O modelo de componentes tem tido muito sucesso com relação aos serviços técnicos, no sentido de que é possível mudar o banco de dados de uma aplicação sem, necessariamente, ter que mudar outras camadas da arquitetura, por exemplo.

Mas a redução da granularidade dos componentes de negócio não tem no final aparência de objetos, ao menos no sentido de entidades instanciáveis comportamentalmente completas, mas na verdade a aparência de sub-rotinas, apenas blocos de código que podem transformar uma entrada numa saída.

Os autores consideram melhor deixar o conceito de orientação a objetos e componente separados. Aplicando o conceito de montagem de componentes na sua infra-estrutura técnica e continuando com a modelagem pura de objetos na camada de negócio.

5.6 O *framework* Naked Objects

O *framework* Naked Objects é uma pacote de software *open-source* escrito em Java que facilita a construção completa de aplicações de negócio a partir de naked objects (objetos comportamentalmente completos).

Integra um ambiente de execução, que inclui um mecanismo de visualização que cria, em tempo real, uma representação manipulável pelo usuário de qualquer naked object que o usuário precisar acessar e um mecanismo de persistência, que torna o naked objects persistente via uma classe de persistência específica. O *framework* já vem com uma classe de persistência que armazena cada naked object como um arquivo XML. Isso agiliza o processo de prototipação, mas não é escalável. Classes de persistência mais sofisticadas devem ser escritas em sistemas reais.

Ambos os mecanismos funcionam de forma autônoma utilizando um mecanismo comum de reflexão para inspecionar suas definições de objetos de negócio quando necessários. O programador não precisa se preocupar diretamente com os mecanismos de visualização ou de persistência. O ambiente de execução também fornece interface entre os naked objects e outros serviços de infra-estrutura tais como para segurança e autorização.

Além disso inclui também um conjunto de *frameworks* de teste. O *framework* de testes unitários é uma simples extensão do popular *framework* JUnit, tratando as capacidades específicas dos naked objects. Existe um *framework* separado para testes de aceitação, o qual facilita a confecção dos testes de aceitação antes de escrever as funcionalidades (como defende a *Extreme Programming*). Como a maioria desses cenários de teste representam tarefas comuns executadas pelos usuários, este *framework* tem a vantagem adicional de poder gerar a documentação do usuário dessas tarefas diretamente a partir da definição dos testes.

Para construir sistema com base no Naked Objects, o importante é se concentrar nos objetos de negócio e deixar a interface e persistência para o *framework*. Esses objetos são definidos como classes Java seguindo um pequeno número de convenções de codificação. A implementação da interface é fornecida pelo Naked Objects.

A abordagem através de objetos comportamentalmente completos associada ao *framework* Naked Objects permite construir sistemas de negócio apenas definindo os objetos do domínio específico. No sistema todas as ações de usuários consistem em criar ou recuperar objetos, especificando seus atributos, estabelecendo associações entre eles, ou invocando métodos em um objeto(ou coleção), ou seja, através de uma interface com o usuário realmente orientada a objetos. O encorajamento ao desenvolvimento de sistemas de negócio realmente orientados a objetos acontece de uma maneira negativa, tornando muito difícil para o desenvolvedor separar os dados dos procedimentos; e de uma maneira positiva, tornando a idéia de objetos comportamentalmente completos concreta e rápida

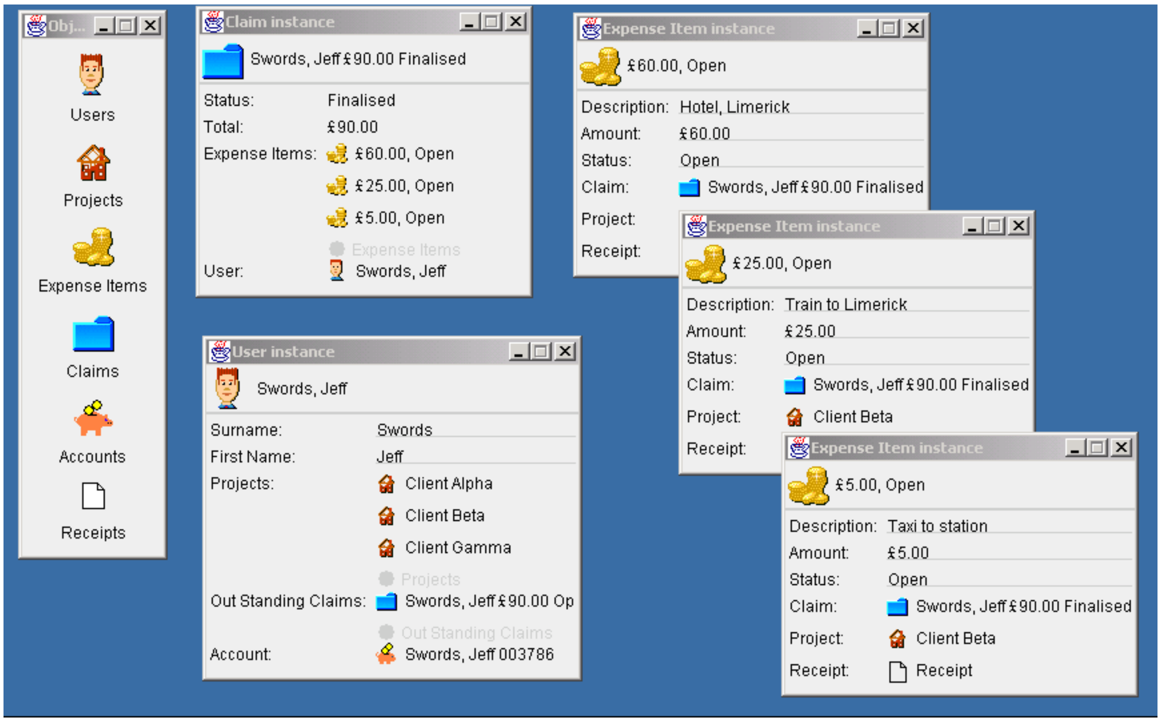

Figura 1: Ambiente *Naked Objects*

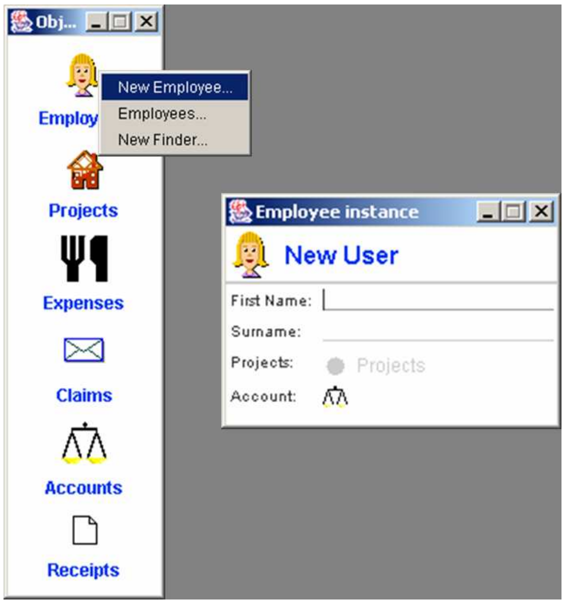

Figura 2: As classes de objetos com as quais o usuário pode iniciar o trabalho (criar novas instâncias, encontrar instâncias existentes, etc.) é exibido na janela de classes. Cada ícone representa uma classe(esquerda). Novas instâncias de um objeto podem ser criadas arrastando-se a classe para a área de trabalho. Ou através do menu pop-up acessado com o botão direito do mouse.

Figura 3: Cada objeto criado está disponível ao usuário e quando exibidos aparecem na área restante da tela.

para desenvolver sistemas.

Os autores consideram como principais benefícios disso:

- Aumento na produtividade de desenvolvimento, uma vez que não é necessário escrever a interface com o usuário.
- Aumento na produtividade para manter e evoluir o sistema. considerando que os sistemas se tornam mais ágeis sendo capazes de melhor acomodar futuras alterações nos requisitos de negócio.
- Facilita a identificação dos requisitos de negócio uma vez que os naked objects podem constituir uma linguagem comum entre desenvolvedores e usuários.

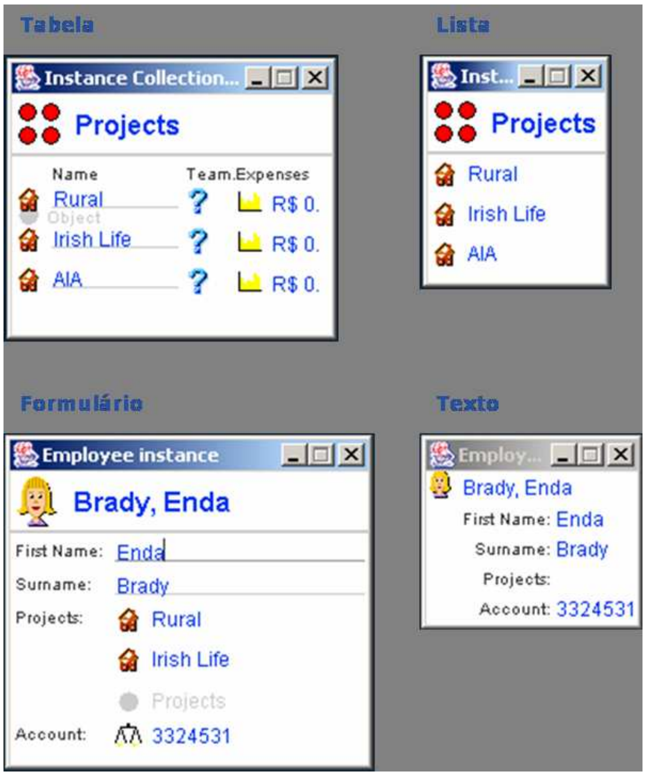

Figura 4: Cada objeto pode ser visualizado de formas diferentes, coleções podem ser visualizadas como tabelas e listas, Empregados como formulários, etc.

Figura 5: Cada objeto de negócio também tem um menu de contexto que é usado para acessar as capacidades desse objeto.

Figura 6: Os objetos de valor(texto, número, datas) que um objeto de negócio possui, podem ser editados.

Figura 7: Os objetos de negócio podem também ser arrastados e soltos sobre outros objetos de negócio ou dentro de campos que permitem esse tipo de objeto

6 Objetos complexos

Nos anos oitenta os sistemas de banco de dados relacionais dominaram o mercado de aplicações de negócio que lidavam com processamento de volumes consideráveis de dados. A simplicidade do modelo relacional de dados e o gerenciamento dinâmico de banco de dados trouxeram um aumento significante de produtividade no desenvolvimento de aplicações.

Com o surgimento de novas classes de domínios de aplicação, incluindo sistemas multimídias, novos modelos científicos e sistemas de análise de informações, as limitações dos sistemas de banco de dados relacionais foram expostas. Com a abordagem ralacional, o espaço de objetos do mundo real deveria ser mapeado para uma coleção de relações *"planas"*(tabelas), acarretando que muito da semântica herdada da composição dos objetos complexos fosse perdida. A necessida de novas formas de modelar e manipular objetos complexos tornou-se um desafio para os pesquisadores de banco de dados e para a indústria.

Objetos Complexos e Orientação a Objetos são duas das principais direções de pesquisa para melhorar a tecnologia de banco de dados. Objetos complexos evoluiram do ambiente relacional. Apesar do fato de possuir atributos no sentido relacional, objetos complexos podem ser construídos de componentes que são objetos por si próprios e podem ser propiamente complexos. Na modelagem de objetos, objetos complexos podem ser construídos através da repetida aplicação de agregação e construtores de agrupamento de objetos, podendo então cobrir a maioria dos aspectos estruturais de novas aplicações. Existem duas razões importantes para se considerar o paradigma orientado a objetos como a base para uma nova geração de tecnologia em banco de dados.

Primeiramente a abordagem orientada a objetos permite uma base uniforme para o tratamento de relações *"planas"* e objetos complexos arbitrário tanto no aspecto estrutural quanto no comportamental. Os conceitos mais importantes de orientação a objetos nesse ponto de vista incluem a abstração de dados e seu encapsulamento, identidade própria de um objeto e sua classificação, herânça, comunicação e computação através de troca de mensagens.

A segunda razão é que o paradigma orientado a objetos provê um paradigma fundamental baseado na nossa maneira de pensar e modelar aplicações. Sua ênfase em modularidade, extensibilidade e reutilização se encontra em cada estágio da modelagem e do desenvolvimento de uma aplicação. Então a abordagem de modelagem orientada a objetos se adequa ao objetivo de reduzir as dificuldades de desenvolvimento de sistemas. Este é precisamente o objetivo que dirigiu a tecnologia de gerenciamento de dados do sistema de arquivos para o banco de dados relacional algum tempo atrás.

O primeiro passo para aplicar a abordagem orientada a objetos para o desenvolvimento de sistemas de informação concentra-se na descrição abstrata de aspectos estáticos e dinâmicos dos objetos do mundo real de interesse. Freqüentemente, tais objetos possuem outros objetos como partes (objetos complexos). Esses relacionamentos são chamados de *"é-parte-de"*. Alguns objetos do mundo real podem se conectar com outros sem carregar a semântica de *"é-parte-de"*. Este tipo de relacionamento é chamado de associação. Em [16] é definido um modelo extensível de especificação orientado a objeto. Objetos complexos e composição de obejtos são descritos por meio de abstrações de dados de alto nível, como classificação, generalização, agregação e associação. Além disso é apresentada uma formalização dos construtores para a modelagem de objetos complexos que será estudada em mais detalhes futuramente.

Referências

[1] E. Bertino and M. Lorenzo. *Object oriented database systems*. International Computer Science Series. Addison-Wesley, 1993.

[2] J. S. Brown and P. Duguid. *The Social Life of Information*. Harvard Business School Press, 2000.

[3] O. J. Dahl and K. Nygaard. Simula, an algol-based simulation language. *Comunications of ACM(9):671-678*, 1996.

[4] Homepage do enterprise java beans (ejb). http://java.sun.com/products/ejb/index.jsp.

[5] J. E. Ferreira, S. B. Navathe, L. P. S. Gomes, and C. Traina Junior. A canonical approach to relational database design for managing the knowledge of generalization. *Processo de avaliação do Journal of Information System*, 2003.

[6] D. Firesmith. *Use Cases: The Pros and Cons in Wisdowm of the Gurus*. SIGS books, 1996.

[7] M. Hammer and J. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. HarperCollins, 1993.

[8] Homepage do hibernate. http://www.hibernate.org/.

[9] I. Jacobson and J. Rumbaugh et al. *The Unified Software Development Process*. Addison-Wesley, 1999.

[10] I. Jacobson and M. Christersson et al. *Object-oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, 1992.

[11] I. Jacobson and M. Ericsson et al. *The Object Advantage - Business Process Reengineering qith Object Technology*. Addison-Wesley, 1995.

[12] Homepage do java data objects (jdo). http://java.sun.com/products/jdo/index.jsp.

[13] R. Kanigel. *The One Best Way - Frederick Winslow Taylor and the Enigma of Efficiency*. Little, Brown and company, 1997.

[14] W. Kim. *The Object Model, Interoperability and Beyond*. Addison-Wesley, 1995.

[15] G. Krasner and S. Pope. A cookbook for using model-view-controller user interface paradigm in smaltalk-80. *Journal of Object Oriented Programming 1(3):26-49*, 1988.

[16] L. Liu. *A formal approach to structure, algebra and communication of complex objects*. Wibro dissertatiedrukkerij, Helmond, 1992.

[17] Homepage do framework naked objects. http://www.nakedobjects.org/.

[18] R. Pawson and J. L. Bravard et al. The case for expressive systems. *Sloan Management Review Winter 1995:41-48*, 1995.

[19] R. Pawson and R. Matthews. Naked objects. http://www.nakedobjects.org/content.html.

[20] M. Porter. *Competitive Advantage: Creating an Sustaining Superior Performance.* Free Press, 1985.

[21] C. Stabell and O. Fjeldstad. Configuring value for competitive avantage: On chains, shops and networks. *Strategic Management Journal 19:413-437*, 1998.

[22] F. Taylor. *The Principles of Scientific Management.* W.W. Norton and Co., 1911.