Uma Visão Realística da Reutilização em Orientação a Objetos

Janeiro, 1998

Para alcançar os reais benefícios da reutilização em orientação a objetos, você deve entender as diferentes espécies de reutilização e onde e como aplicá-las.

Por: *Scott Ambler*

.

Traduzido por: Ivan M. Silva (Novembro, 2001)

Complemento: *Recursos Reutilizáveis*
Complemento: *E Quanto a Objetos de Negócio?*

Reusabilidade é uma das grandes promessas da tecnologia orientada a objetos. Infelizmente, é uma promessa que freqüentemente não é realizada. O problema é que reutilização não vem de graça; não é uma coisa que você simplesmente obtém porque está usando ferramentas de desenvolvimento orientado a objetos. Ao contrário, é algo pelo qual você precisa trabalhar arduamente se quiser obter sucesso. A primeira lição é que há mais a reutilizar do que código. Reutilização de código é a forma menos produtiva de reutilização disponível. Não me entenda errado, reutilização de código é ainda um boa coisa; mas é que você pode dar um maior alcance aos seus saltos em reutilização em outra parte. Há muito mais em uma aplicação do que código-fonte; portanto, você deve ser capaz de reutilizar muito mais do que código. Vamos explorar os tipos de reutilização em detalhes, e ver onde você pode aplicá-los na construção de aplicações.

Reutilização de Código

Reutilização de código, o tipo mais comum de reutilização, refere-se à reutilização de código-fonte dentro de seções de uma aplicação e potencialmente através de múltiplas aplicações. No melhor caso, reutilização de código é alcançada compartilhando-se classes e coleções de funções e rotinas comuns (isso é possível em C++, mas não em Smalltalk ou Java). No pior caso, reutilização de código se faz copiando e depois modificando código existente. Uma lamentável realidade de nossa indústria é que copiar é geralmente a única forma de reutilização praticada pelos desenvolvedores.

Um aspecto chave da reutilização de código é que você precisa ter acesso ao código-fonte. Quando necessário, você mesmo o modifica ou tem alguém para modificá-lo para você. Isso é ao mesmo tempo bom e ruim. Ao olhar o código, você pode determinar, embora em geral demoradamente, se você quer ou não reutilizá-lo. Ao mesmo tempo, liberando o código totalmente para você, o desenvolvedor original pode ficar menos motivado a documentá-lo adequadamente, aumentando o tempo que você leva para entendê-lo e consequentemente diminuindo o benefício para você.

A principal vantagem de reutilização de código é que ela reduz a quantidade real de código que você precisa escrever, diminuindo potencialmente os custos, tanto do desenvolvimento quanto da manutenção. As desvantagens são que o escopo do efeito é limitado à programação e geralmente aumenta o acoplamento dentro da aplicação.

Reutilização de Herança

Reutilização de herança refere-se ao uso de herança em sua aplicação para tirar vantagem de comportamento implementado em classes existentes. Herança é um dos conceitos fundamentais de orientação a objetos, permitindo que você modele relacionamentos “é um”, “é como” e “é do tipo”. Por exemplo, para desenvolver uma classe VerificarConta, você começa por obter herança de ContaResultado, diretamente reutilizando todo o comportamento implementado nessa classe.

A vantagem da reutilização de herança é que você tira proveito de comportamento previamente desenvolvido, o que diminui tanto o tempo de desenvolvimento como o custo da aplicação. Infelizmente, há diversas desvantagens na reutilização de herança. Primeiro, a má utilização de herança resulta freqüentemente nos desenvolvedores perdendo a oportunidade de reutilizar componentes, que oferecem um nível mais alto de reutilização. Segundo, desenvolvedores novatos freqüentemente se restringem em testar a regressão da herança (a execução de casos de teste da superclasse numa subclasse), resultando numa hierarquia de classes frágil, que é difícil de manter e incrementar. Como você pode ver, isso é reutilização, mas a um custo proibitivo.

Reutilização de Modelos* (*Templates*)

Reutilização de modelos é tipicamente um forma de reutilização de documentação. Refere-se à prática de usar um conjunto comum de layouts para artefatos chaves de desenvolvimento - documentos, modelos (*models**) e código-fonte - na sua organização. Por exemplo, é muito comum as organizações adotarem modelos (*templates*) comuns de documentação para casos de uso, relatórios de status, cronogramas de desenvolvimento, solicitações de mudança, requisitos de usuário, arquivos de classes e cabeçalhos de documentação de métodos. A principal vantagem de modelos de documentação é que eles aumentam a consistência e a qualidade de seus artefatos de desenvolvimento. A principal desvantagem é que os desenvolvedores têm uma tendência a modificar modelos para seu próprio uso e não compartilham suas alterações com os colegas. Para os melhores resultados com modelos, você precisa tornar fácil para os desenvolvedores trabalharem com eles. Tenho visto implementações de modelos tão simples como modelos de documentos do *Microsoft Word*, tanto como complexos bancos de dados do Lotus Notes compartilhados por todos os desenvolvedores. Sua organização também precisa providenciar treinamento sobre como e quando usar os modelos, de forma que qualquer um possa usá-los consistente e corretamente.

* NT.: Alguns termos técnicos que se distinguem perfeitamente em inglês acabam por se confundir na tradução para o português, exigindo que o leitor atente para o contexto em que está empregado, "template" é modelo na acepção concreta, no sentido de molde, uma base concreta de onde partir, tal como um arquivo ".dot" do Word, ou um modelo de formulário. "Model", é também modelo, mas numa acepção mais abstrata, como usado em "modelo de dados". Framework, a ser visto adiante, também traz um sentido de molde, mas mais como estrutura fundamental, ou alicerce, e muitas vezes pode ser entendido como "ambiente" ou "contexto", mas nesse documento optamos por não traduzi-lo dada sua utilização corrente na forma inglesa..

Reutilização de Componente

Reutilização de componentes refere-se ao uso de componentes pré-construídos e totalmente encapsulados no desenvolvimento de suas aplicações. Componentes são tipicamente auto-suficientes e encapsulam um único conceito. A reutilização de componentes difere da reutilização de código pelo fato de que você não tem acesso ao código fonte. Difere-se da reutilização de herança porque não faz criação de subclasses. Exemplos comuns de componentes são os *Java Beans* e componentes *ActiveX*.
Há muitas vantagens na reutilização de componentes. Primeiro, ela oferece um escopo maior de reusabilidade do que o da reutilização de código ou de herança, porque os componentes são auto-suficientes - você literalmente os "pluga" e eles fazem o trabalho. Segundo, a utilização ampla de plataformas comuns, como o sistema operacional Win32 e o Java Virtual Machine, provêem um mercado amplo o bastante para que terceiros criem e vendam componentes para você a baixo custo. A principal desvantagem da reutilização de componente é que os componentes são pequenos e encapsulam um único conceito, e você acaba precisando de uma grande biblioteca deles.
O caminho mais fácil para o trabalho com componentes é começar com objetos (widgets) da interface de usuário – barras de deslocamento, componentes gráficos e botões gráficos (para nomear alguns). Mas não se esqueça que há muito mais em uma aplicação do que a interface de usuário. Você pode ter componentes que encapsulem recursos do sistema operacional, tal como acesso à rede, ou que encapsulem recursos de persistência, tais como componentes que acessam bancos de dados relacionais. Se você está construindo seus próprios componentes, tenha o cuidado de que eles façam somente uma coisa. Por exemplo, um componente de interface de usuário para edição de endereços de superfície é bastante reutilizável por que você pode usá-lo em várias telas de edição. Um componente que edita endereços de superfície, endereços de e-mail e um número de telefone já não é tão reutilizável – não há muitas situações onde você queira todos esses três recursos simultaneamente. Em vez disso, é melhor construir três componentes reutilizáveis e usar cada um quando necessário. Quando um componente encapsula apenas um conceito, é um componente coeso.

Reutilização de *Framework*

Reutilização de *framework* refere-se ao uso de coleções de classes que implementam em conjunto a funcionalidade básica de um domínio técnico ou de negócios comum. Os desenvolvedores usam *frameworks* como o alicerce a partir do qual eles constróem uma aplicação; os 80% comuns já estão no lugar, eles apenas necessitam acrescentar os restantes 20% específicos da aplicação. *Frameworks* que implementam os componentes básicos de uma GUI são muito comuns. Há *frameworks* para seguros, recursos humanos, manufatura, bancos e comércio eletrônico. Reutilização de *framework* representa um alto nível de reutilização no nível do domínio. Os *frameworks* fornecem uma solução inicial para um domínio de problema e freqüentemente encapsulam uma lógica complexa que levaria anos para ser desenvolvida desde o rascunho. Infelizmente, a reutilização de *framework* sofre de diversas desvantagens. A complexidade dos *frameworks* torna-os difíceis de serem dominados, exigindo um longo processo de aprendizagem por parte dos desenvolvedores. *Frameworks* geralmente são de plataformas específicas, amarrando você a um único fornecedor, aumentando o risco de sua aplicação. Embora os *frameworks* implementem 80% da lógica necessária, são geralmente os 80% mais fáceis; a parte difícil, a lógica de negócio e os processos que são únicos de sua organização, ainda é deixada para você fazer. *Frameworks* raramente trabalham juntos a menos que eles venham de um mesmo fornecedor ou consórcio de fornecedores. Eles sempre exigem que você modifique seu negócio para adequar-se ao *framework* em vez de outro caminho.

Reutilização de Artefatos

Reutilização de artefatos refere-se ao uso de artefatos de desenvolvimentos previamente criados – casos de uso, documentos-padrão, modelos específicos de domínio, procedimentos e diretrizes, e outras aplicações – para dar o chute inicial em um novo projeto. Há diversos níveis de reutilização de artefatos, indo dos 100% de reutilização pura onde você toma um artefato tal qual ele é e o utiliza em um novo projeto, para exemplo de reutilização quando você observa o artefato para ter uma idéia sobre como proceder. Por exemplo, documentos-padrão tais como padronização de código e de interface são artefatos valiosos para reutilização entre projetos, tanto como documentos de notação de modelagem e documentos de resumo de metodologia. Tenho sido também capaz de reutilizar aplicações existentes, seja via uma interface de dados comum, seja por colocar um invólucro (*wrapper*) de orientação a objeto em torno delas para fazê-las parecer como classes normais.
A reutilização de artefatos promove consistência entre projetos e reduz a carga de gerenciamento para cada novo projeto.
Uma outra vantagem é que você pode freqüentemente adquirir vários artefatos ou encontrá-los online: padrões de interface de usuário são comuns para muitas plataformas, padrões de codificação para as principais linguagens estão geralmente disponíveis e metodologias padronizadas de orientação a objetos e notações de modelagem têm estado disponíveis há anos. A principal desvantagem da reutilização de artefato é que ela é freqüentemente percebida como reutilização superficial por programadores "*hard-core*" – você simplesmente passa os padrões e conectores de procedimentos de uma mesa para outra. O ponto (*baseline*) é que a reutilização de artefatos é uma técnica importante e viável que não deveria ser ignorada.

Reutilização de Padrão (*Pattern*)

Reutilização de padrão refere-se ao uso de abordagens publicamente documentadas para solução de problemas comuns. Padrões são freqüentemente documentados como um simples diagrama de classes e tipicamente compreendem de uma a cinco classes. Na reutilização de padrão, você não está reutilizando código; ao contrário, você está reutilizando a inteligência que está por trás do código. Padrões são uma forma de reutilização muito alta, que experimentará uma longa expansão de vida – pelo menos entre as linguagens de computação que você está atualmente utilizando e potencialmente pelo próprio paradigma de orientação a objeto.
O padrão de "Ponto de Contato", tirado de meu livro *Building Object Applications That Work* (SIGS Books, 1997), foi modelado com o uso de um diagrama de classe da *Unified Modeling Language* (UML) 1.1, mostra um padrão comum para rastrear os pontos de contato entre sua empresa e outras entidades de negócio. Esse padrão mostra que você pode tratar endereços de e-mail, endereços de superfície e números de telefone como o mesmo tipo de objetos – pontos de contatos através dos quais sua organização interage com outras entidades de negócio (clientes, empregados, fornecedores e assim por diante). Esse padrão aumenta a flexibilidade de suas aplicações. Ele mostra que você pode não somente postar uma carta para um cliente mas também enviar-lhe um e-mail ou um fax. Em vez de enviar um CD-ROM ou uma fita de vídeo para um endereço, você pode transmitir o produto eletronicamente. O padrão "Ponto de Contato" é uma chave de habilitação para fazer essas coisas. Eu tenho implementado com sucesso esse padrão de análise em várias aplicações, reutilizando a parte mais difícil de um modelo – o pensamento que ficou por trás dele. A reutilização de padrões provê uma reutilização de alto nível que você pode implementar em muitas linguagens e plataformas. Padrões encapsulam o aspecto mais importante do desenvolvimento – a inteligência que está embutida na solução. Padrões aumentam a capacidade de manutenção e de incrementar sua aplicação usando abordagens comuns para problemas que são reconhecidos por qualquer desenvolvedor experiente em orientação a objetos. A desvantagem da reutilização de padrão é que os padrões não fornecem uma solução imediata – você ainda tem que escrever o código que implementa o padrão.

Reutilização Componente de Domínio

A reutilização de componente de domínio refere-se à identificação e ao desenvolvimento de componentes de negócios reutilizáveis de larga escala. Um componente de domínio é uma coleção relacionada de classes de domínio e de negócios que trabalham juntas para suportar um conjunto coeso de responsabilidades. Um diagrama de componente para uma firma de telecomunicações tem diversos componentes de domínio, cada qual encapsulando muitas classes. O componente "OfertaDeServiços" encapsula mais de 100 classes indo desde uma coleção de classe de modelagem de planos de chamadas interurbanas até de pacotes de televisão a cabo ou de pacotes de serviços de Internet. Componentes de domínio são inicialmente identificados dando-se uma abordagem dirigida a arquitetura para a modelagem, são modificados e incrementados durante o desenvolvimento de novas aplicações para a organização.
Componentes de domínio fornecem o maior potencial de reutilização, porque eles representam pacotes coesos de larga escala de comportamentos de negócio que são comuns a muitas aplicações. Tudo o que você produz durante o desenvolvimento de domínio deveria poder ser reutilizado. Componentes de domínio são efetivamente escaninhos arquiteturais nos quais os comportamentos de negócio são organizados e depois reutilizados.

As boas notícias sobre essas abordagens de reutilização é que você pode usá-las simultaneamente. Sim, a reutilização de *framework* freqüentemente prende você à arquitetura daquele *framework* e aos padrões e diretrizes que ele usa, mas você ainda pode tirar vantagem das outras abordagens de reutilização com *frameworks*. As reutilizações de artefato e de componente são o ponto mais fácil para começar; com um pouco de pesquisa, você pode rapidamente encontrar itens reutilizáveis.
Você pode comprar modelos de documentação online, mas se a sua organização não tiver um processo de desenvolvimento bem definido, pode ser que você obtenha pouco benefício de modelos. A reutilização de herança e de padrão é de domínio dos projetistas e eles começarão a aplicar essas formas de reutilização como matéria do dia.
Realisticamente, eu começaria com a reutilização de artefato e de componente para treinar os projetistas no uso apropriado de herança e de padrões, implementaria uma abordagem de desenvolvimento dirigida a arquitetura para identificar e desenvolver componentes de domínio e confiaria aos meus codificadores a reutilização de todo o código que pudessem.

Um Ponto de Vista Tipo-Classe (*Class-Type*)

Agora que temos diversas técnicas em nossa caixa de ferramentas de reutilização, vamos ver onde podemos aplicá-las. A arquitetura de tipo-classe em quatro camadas indica as abordagens de reutilização para cada camada. Como cada camada tem suas propriedades únicas e demandas de desenvolvimento próprias, faz sentido que a abordagem para reutilização deva ser única para cada camada. Vamos olhar cada camada em detalhe.

A **camada de interface de usuário** encapsula telas e relatórios, pontos nos quais os usuários interagem com sua aplicação. O tipo mais comum de reutilização para a camada de interface de usuário é, obviamente, a de componente em objetos de interface – é muito comum aos desenvolvedores adquirirem bibliotecas de componentes de barra de rolagem, gráficos, listas e outros elementos de interface. Também importante para essa camada é a reutilização de artefato e, mais especificamente, de padrões (*standards*) de interface de usuários e de layout de relatórios. Uma das coisas mais importantes no projeto de interface é a consistência e seguir um conjunto comum de padrões e diretrizes é a maneira mais fácil de promovê-la dentro da aplicação.

A **camada de negócios** encapsula a lógica de domínio de problema de sua aplicação. A reutilização de padrão (*pattern*), mais especificamente de padrões de análise, é importante para a camada de negócios, tanto quanto a reutilização de *framework* para o domínio de negócios relevante e para bibliotecas comuns de classes de negócios. Embora todas as três abordagens de reutilização sejam ainda bastante imaturas, elas estão se desenvolvendo rapidamente para muitos domínios de negócios. Uma vasta coleção de padrões está sendo desenvolvida tanto nas academias quanto no mundo de negócios e *frameworks* comuns estão sendo cultivados em grandes companhias de consultoria e tecnologia que têm experiência em um ou mais mercados verticais.
O mais importante para a reutilização dentro da camada de negócios, contudo, é a introdução de componentes de domínios na sua empresa. Componentes de domínio – produto de um abordagem de desenvolvimento orientada a arquitetura - são importantes para uma abordagem que é fundamentada sobre reutilização.

A **camada de persistência** encapsula e generaliza o acesso a vários mecanismos de persistência que você usa para armazenar objetos permanentemente, tais como banco de dados relacionais, banco de dados orientados a objetos, e arquivos simples de disco (*flat files*). As mais importantes formas de reutilização para esta camada são a reutilização de *framework* e a reutilização de componente. Isso porque você pode comprar pacotes que implementam total ou parcialmente essa camada de persistência. Reutilização de padrão de projeto (*design*) também tem forte probabilidade, visto que projetos de camada de persistência têm estado presentes em muitas aplicações. O uso de dicionários de dados, como reutilização de artefato, dentro da organização também deveria ser possível.

A **camada de sistema** encapsula e generaliza o acesso ao sistema operacional, à rede, ao hardware e a outras aplicações. A reutilização de padrão de projeto (*design pattern*), especialmente para envelopamento (*wrapping*) de hardware e aplicações, é muito comum para a camada de sistema, tanto quanto a reutilização de artefato como classes de empacotamento (*wrapper*).

Dentro das quatro camadas, a reutilização de código e de herança é obviamente aplicável. Você poderia também argumentar que a reutilização de padrão é alcançável na camada de interface de usuário. Durante todo o projeto, a reutilização de modelos para layouts de documentos-padrão e para padronização (*standards*) de desenvolvimento e para procedimentos é também significativa.

Os Segredos para o Sucesso na Reutilização

Então como você realmente atinge a reutilização em orientação a objetos? Eu desejaria poder dizer que tudo o que você precisa fazer é correr e comprar algumas ferramentas e um repositório para armazenar seu trabalho reutilizável para ter um bom começo. Infelizmente, reutilização é algo mais do que ferramentas. De fato, muitas organizações tem falhado miseravelmente por causa de seu foco em ferramentas e não no processo. Aqui vão algumas dicas de reutilização:

> ***Você não pode chamar algo de reutilizável a menos que tenha sido reutilizado pelo menos três vezes em três diferentes projetos por três equipes distintas.*** Você pode tentar projetar para reutilização, mas não pode reivindicar sucesso até que algum elemento da aplicação tenha sido reutilizado pelo menos umas três vezes. A reusabilidade é do ponto de vista de quem a emprega não de quem a cria.
>
> ***Itens reutilizáveis precisam ser bem documentados e ter um ou mais exemplos reais de como usá-los.*** E mais, a documentação deve indicar quando **não** reutilizar o item, de forma que os desenvolvedores possam entender o contexto no qual ele deve ser usado.
>
> ***A única maneira pela qual você pode tornar a reutilização uma realidade é se você planeja para isso.*** Você precisa alocar tempo e recursos necessários para tornar seu trabalho reutilizável. Se você não der um passo atrás e gastar tempo para generalizar seu trabalho, isso nunca acontecerá – pressões de projeto nunca o motivarão a por o trabalho de lado até que você tenha tempo para fazê-lo. O Ponto é que, se o gerenciamento de reutilização – o ato de especificamente reutilizar itens existentes e constantemente adicioná-los ao estoque de itens reutilizáveis – não for parte de seu processo de desenvolvimento, então a reutilização não acontecerá.
>
> ***Reutilização é uma atitude.*** Quando você começa um projeto, a primeira coisa que deveria fazer é determinar que porções de sua aplicação pode ser reutilizadas por outra pessoa. Talvez algum outro tenha construído o que você precisa, ou talvez você possa comprar componentes ou outros itens reutilizáveis. O outro lado da moeda é que você precisa estar disposto a compartilhar o seu trabalho com outras pessoas de modo que elas possam reutilizá-lo. Um bom líder de equipe estará constantemente à procura de oportunidades de reutilização e irá promover e recompensar a reutilização em seu time. Uma excelente abordagem é procurar por reutilização durante inspeções: durante a revisão de modelos, observar procurar por oportunidades para reutilização de herança e de padrão (*pattern*); durante a revisão de código, procurar por reutilização de componente e de código.

Freqüentemente você terá que combater a síndrome do "Não Inventado Aqui" (NIA), um problema que poderia impedi-lo de tentar espalhar a atitude de reutilização em sua equipe. De acordo com a síndrome do NIA, os desenvolvedores não reutilizarão o trabalho de outros porque não foram eles mesmos que criaram. Pura tolice. Desenvolvedores profissionais constantemente procuram reutilizar o trabalho de outros, porque isso os libera para trabalhar em porções específicas de domínio de suas próprias aplicações. Minha experiência é que desenvolvedores profissionais irão prontamente reutilizar o trabalho de outros tão logo ele atenda às suas necessidades, seja de boa qualidade e seja bem documentado e fácil de entender. A síndrome do NIA é um mito. Se fosse verdade, então os ambientes de desenvolvimento orientados a objetos não viriam com bibliotecas de classes, nem centenas de companhias estariam vendendo componentes reutilizáveis e não existiriam *frameworks*.

Propaganda boca-a-boca é freqüentemente a maneira pela qual as pessoas encontram coisas a reutilizar. Claro, repositórios de reutilização são boas coisas para ter, mas eles são similares em conceito à cadeia de comando de sua organização. Assim como algumas coisas aparecem pelos canais oficiais e outras por sua rede informal de contatos, você também pode encontrar alguns elementos reutilizáveis no repositório e alguns através de seus amigos.

Reutilização é uma coisa de organização, não algo de projeto. Muitas empresas falham na reutilização porque não entendem seu escopo. Reutilização entre projetos é onde você obtém retorno, não em um único projeto. Muitas empresas desistem prematuramente da reutilização quando não alcançam-na no primeiro projeto, o que é loucura, quando se considera que não a nada a reutilizar no começo. Por isso é importante uma abordagem de desenvolvimento dirigida a arquitetura – porque olha além das necessidades de um único projeto para as necessidades de toda a organização, e um a importante necessidade organizacional é reutilizar o trabalho existente em todo lugar possível.

Uma abordagem de "um-tamanho-cabe-tudo" para reutilização não funcionará. Há muitas abordagens diferentes para reutilização, e as necessidades únicas de cada camada de tipo-classe exigem uma diferente combinação de abordagens. Você precisa reconhecer e planejar de acordo com isso. Parte do "não reinventar a roda" é primeiro compreender que você tem mais de uma roda à sua disposição. Você pode reutilizar código-fonte, componentes, artefatos de desenvolvimento, padrões e modelos. E mais importante, o seu trabalho não é simplesmente reutilizável porque você usa orientação a objetos. Ao contrário, é reutilizável porque você gastou tempo para fazê-lo dessa forma.

Recursos Reutilizáveis

A organizações são freqüentemente massacradas quando tentam aumentar a reutilização em seus projetos – elas rapidamente descobrem que geralmente há pouco disponível para elas internamente com valor para reutilizar. O que elas não imaginam é que o exterior é rico de itens esperando para serem reutilizados. Embora eu o deixe livre para encontrar por si mesmo, quero compartilhar com você diversas fontes representativas de reutilização de modelo, componente, f*ramework*, artefato e padrão (a reutilização de código e de herança são abordagens internas).

***Reutilização de Modelo*.** Se sua empresa ainda não definiu modelos comuns para casos de uso, cronogramas e outros, poderia verificar os modelos do Software Productivity Centre (www.spc.ca). Eles têm vários conjuntos de modelos de documentação disponíveis para compra. Um de seus produtos é chamado de *The Essential Set*, que inclui 54 modelos de documentação cobrindo todo o processo de desenvolvimento de aplicações.

Reutilização de Componente. Companhias que vendem componentes reutilizáveis são muito fáceis de encontrar na Internet. Por exemplo, a Gamelan (www.gamelan.com) é uma grande fonte de *links* para *Java Beans* reutilizáveis e a ActiveX..com (www.activex.com) negocia componentes ActiveX para um ampla faixa de necessidades, incluindo desenvolvimento de *web site* e de acesso a banco de dados.

Reutilização de Framework. O custo de *frameworks* reutilizáveis é geralmente muito alto, mas, de novo, as ofertas são freqüentemente mais altas. A PeopleSoft (www.peoplesoft.com) vende *frameworks* para gerenciamento financeiro, gerência de materiais, distribuição, manufatura e recursos humanos. A SAP (www.sap.com) negocia um produto chamado *Business Framework Architecture* para expansão e incremento do controle do processo de negócios dentro do SAP R/3 e SAP R/4.

Reutilização de Artefato. A reutilização de artefato é diretamente atingida tão logo você esteja disposto a procurar nas fontes externas. Todo desenvolvedor para a plataforma Windows deveria ter uma cópia dos padrões de projeto de interface definidos no livro da Microsoft, *The Windows Interface Guidelines for Software Design* (Microsoft Press, 1995). Desenvolvedores Java podem tirar proveito do *AmbySoft Java Coding Standards* disponível em www.ambysoft.com. Projetistas orientados a objetos podem baixar um cópia do documento da notação da Unified Modeling Language (UML) da Rational (www.rational.com).

***Reutilização de Padrão*.** Você pode obter um bom começo em padrões com os livros *Design Patterns: Elements of Reusable Object-Orientded Software* (Addison-Wesley, 1995), de Erich Gamma e outros, e com o livro de Martin Fowler, *Analysis Patterns: Reusable Object Models* (Addison-Wesley,1997). A página Patterns Home Page, http://www.cs.edu/users/patterns/patrerns.html, é também um excelente recurso para a reutilização de padrões.

Reutilização de Componente de Domínio. Para aprender como construir componentes de domínio reutilizáveis, você precisa dar uma abordagem dirigida a arquitetura para o seu desenvolvimento orientado a objetos. Para mais detalhes, eu sugiro *Software Reuse: Architecture, Process, and Organization for Business Success* de Ivar Jacobson e outros (Addison Wesley, 1997).

E Quanto a Objetos de Negócio?

Por anos, especialistas em objetos têm proclamado que está no horizonte o mercado para objetos de negócio, um mercado onde você pode comprar muitos dos objetos necessários à construção de aplicações de negócios. A idéia é que alguém irá construir um objeto de negócio, documentá-lo e colocá-lo à venda no mercado de objetos. Embora isso seja uma grande visão, eu não vejo acontecer em larga escala

Sim, há muitos objetos de negócio comuns entre as organizações; nós todos lidamos com clientes, que têm endereços e compram produtos ou serviços nossos, mas a realidade é que os meus objetos de negócio são muito diferentes dos objetos de negócios de outros. A maneira como eu lido com os meus clientes, a maneira como eu lhes vendo produtos e serviços é muito diferente da maneira como outros fazem. Isso é o que dá a cada um de nós um lado competitivo. Sim, todos necessitamos dos mesmos objetos de negócios, mas como fazemos negócio diferentemente, necessitamos de diferentes implementações para eles. Portanto, o mercado para objetos de negócio nunca será um sucesso como proclamam os especialistas.

Suspeito que haverá um pequeno mas saudável mercado para objetos de negócio que sejam verdadeiramente comuns entre as organizações. Já que conjuntos comuns de regras se aplicam a todas as organizações, eu poderia ser capaz de comprar uma coleção de classes de tributação, um conjunto de classes que representassem um endereço de superfície e potencialmente um conjunto de classes de contabilidade. O único problema (caveat) é que precisamos de um ambiente comum no qual rodar estes objetos, tais como o Java Virtual Machine ou CORBA (*Commom Object Request Broker Architecture*). Java e CORBA são um bom começo, mas ainda precisamos de uma abordagem comum para as camadas de persistência e de sistema (o CORBA quase já tem isso) para obter sucesso. Dentro de poucos anos, espero que os esforços de padronização no Java e CORBA pavimentem a estrada para um mercado de objetos de negócios. O tempo irá dizer.