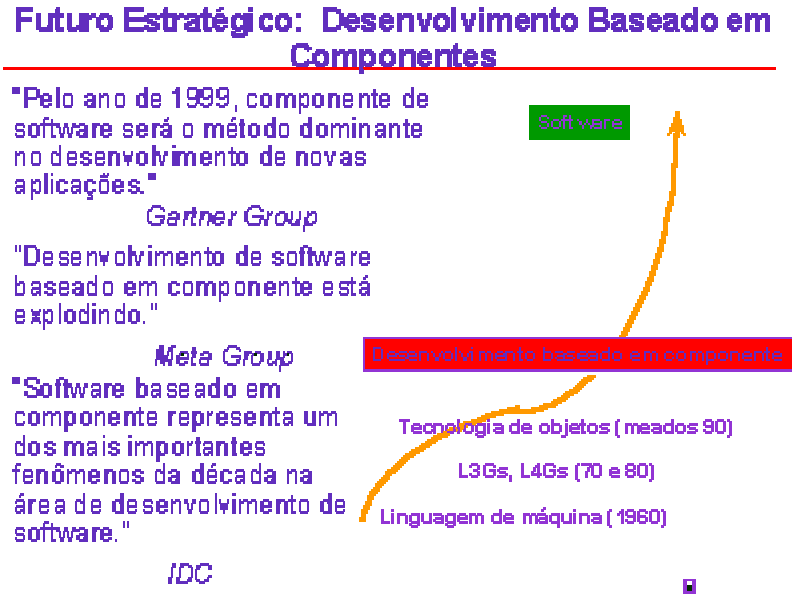
Aplicações Baseadas em Componentes

Guilherme C. Hazan - http://www.guich.com/pf/capitulo3.html

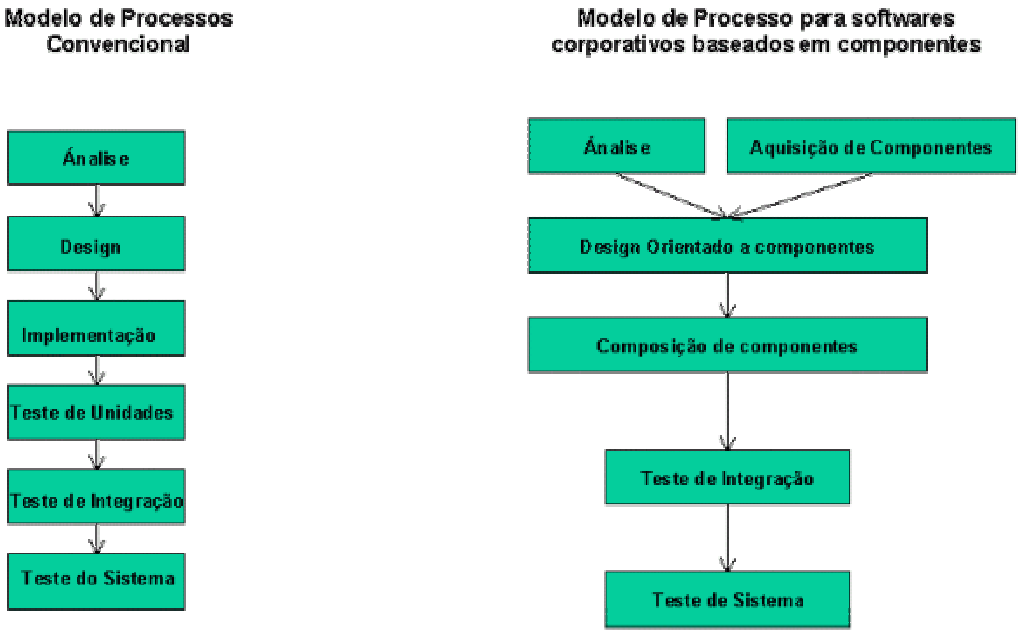
A programação orientada à objeto tem existido por um tempo, mas em anos recentes, construir aplicações através da montagem de componentes de software reutilizáveis emergiu como um modo altamente produtivo para desenvolver aplicações customizadas. Podemos ilustrar o histórico do desenvolvimento de software da seguinte maneira:



Recentemente, um conjunto de fatores despertou um novo interesse em DBC, fornecendo a motivação necessária para se acreditar que possa ser agora mais eficaz e realizado em larga escala. Dentre esses fatores podemos citar o desenvolvimento da WWW(World Wide Web) e da Internet, aumentando o entendimento e preocupação em relação à computação distribuída. A WWW encorajou os usuários a considerarem um conjunto de serviços distribuídos no hiperespaço como sendo, na realidade, um sistema distribuído. Também a mudança de sistemas baseados em mainframes para sistemas baseados na arquitetura cliente/servidor levou os desenvolvedores a considerarem as aplicações não mais sistemas monolíticos, mas sim um conjunto de subsistemas interoperáveis. Por fim, outros fatores foram os padrões de infra-estruturas para construção de aplicações, como as iniciativas do *Object Management Group* (OMG), através da Object Management Architecture (OMA) e dos produtos da Microsoft, como o Component Object Model (COM) e seus derivados, etc.

O DBC mudou o processo da arquitetura de software na medida em que algumas partes do sistema podem ser adquiridas dos fornecedores de componentes. Algumas partes do processo podem ser feitas concorrentemente.

A figura abaixo ilustra a diferença entre os processos *convencional* e os *baseados em componentes*.



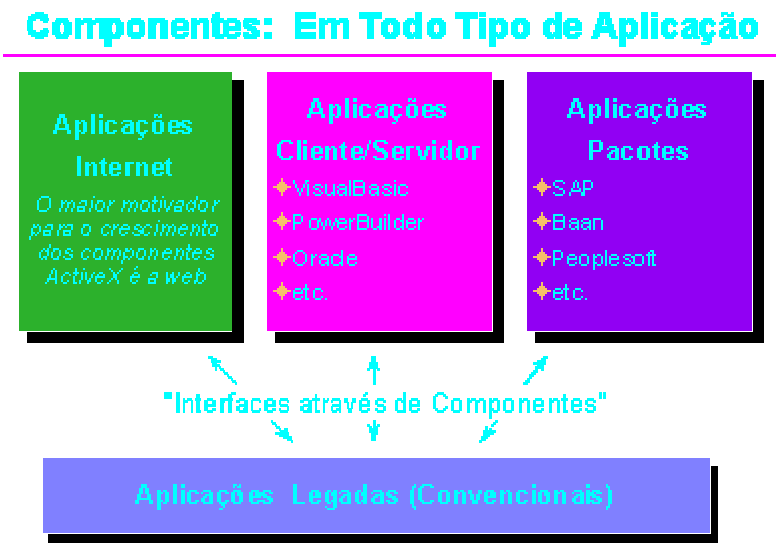
Processos Convencional e da Engenharia de Software baseada em Componentes

O processo de arquitetura de software Baseado em Componentes consiste em dois subprocessos: o desenvolvimento de componentes e a integração de componentes. Desde que esses processos podem ser feitos por duas organizações diferentes eles podem ser concorrentes.

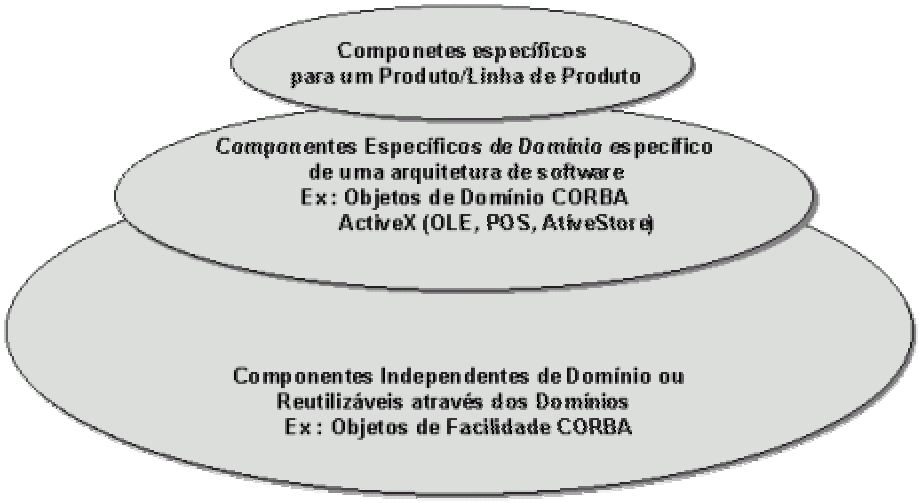
Diferentemente do processo convencional, o processo Baseado em Componentes precisa de um subprocesso de Aquisição de Componentes.

O que são componentes?

Um componente pode ser descrito como pedaços predefinidos de software com interfaces e comportamento bem delineados, que podem ser usados e reutilizados em diferentes aplicações. Componentes são essencialmente objetos em um alto nível abstração que podem se comportar efetivamente como soluções '*caixa preta'* para as necessidades de um sistema específico, fornecendo serviços para uma arquitetura de aplicações bem definida. Com o sucesso das aplicações Visuais RAD (Rapid Aplication Development) os componentes tomaram a forma de entidades gráficas (chamadas 'GUI-based'), com a interface definida por propriedades, métodos e eventos. A figura abaixo dá uma idéia de onde são usados os componentes.



Os componentes podem ser específicos para um produto (ex: Delphi), um domínio (ex: Javabeans, ActiveX) ou independentes de domínio (ex: CORBA que conecta objetos de diferentes domínios como Java, C++ etc). A figura abaixo ilustra esta organização em camadas.



Componentes em Camadas

O que é Desenvolvimento Baseado em Componentes (DBC)?

O Desenvolvimento Baseado em Componentes permite aplicações serem confeccionadas à partir de componentes pré-construídos ao invés de desenvolve-las a partir do zero. É um amadurecimento natural da indústria de software - se baseando na força da orientação à objeto (OO), mas precisando das tecnologias OO e não OO na construção e gerenciamento de aplicações em larga escala. A indústria do software está procurando com o DBC, maximizar a reutilização e aumentar a produtividade do desenvolvimento de aplicações. Os benefícios mencionados desta abordagem apontam

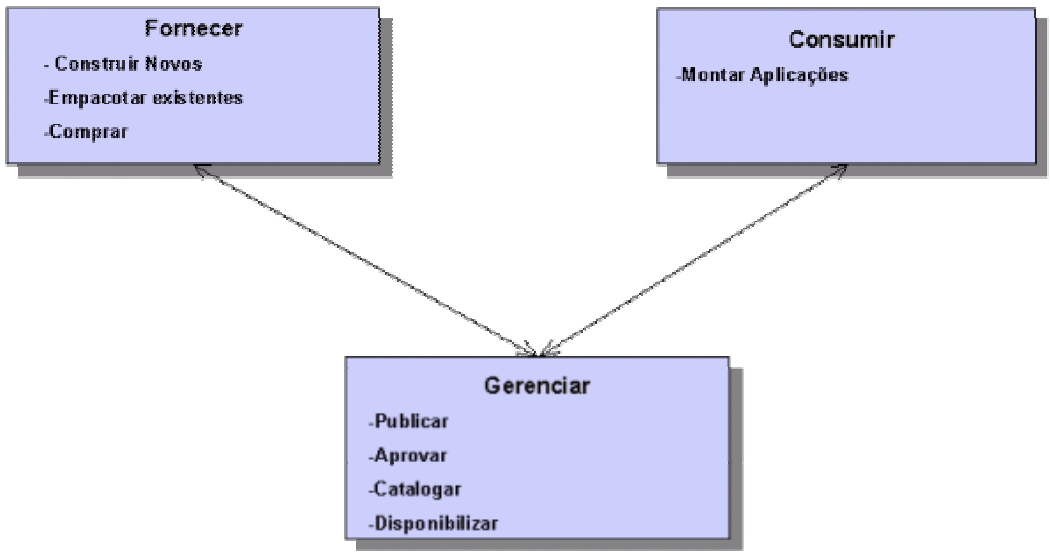
diretamente para a redução de custos de desenvolvimento, precisando para tal, de ferramentas que possuam um enfoque ao desenvolvimento baseado em componentes.

Vantagens de construir componentes

- Reusabilidade: a reutilização da funcionalidade do componente por toda a aplicação
- Produtividade: com o estabelecimento de uma interface bem definida e a redução da complexidade através do encapsulamento, desenvolver aplicações torna-se mais rápido e simples
- Facilidade de Uso e Aprendizado: Através do modelo fornecedor/consumidor, desenvolvedores podem rapidamente se tornar produtivos no DBC sem um extenso treinamento
- Mudanças executadas com facilidade e rapidez: O aumento da modularidade e a ausência de dependências permitem os desenvolvedores modificar, adicionar ou substituir componentes tão rapidamente quanto as necessidades de negócio mudam.
- Melhor foco de negócios: níveis mais altos de abstração permitem desenvolvedores e gerentes de negócios trabalharem juntos para planejar, projetar e construir a aplicação em termos de negócio em alto nível.
- Proteção dos investimentos: Com a criação de novos padrões de interoperacionalidade entre os componentes (JavaBeans, COM/DCOM e CORBA), desenvolvedores poderão ficar certos de que os componentes baseados nesses padrões além de funcionarem imediatamente também funcionarão no futuro.

Fornecimento e Consumo de Componentes

Dentro da indústria DBC existem pessoas ou organizações que desenvolvem componentes para que outras utilizem, são os chamados *fornecedores* de componentes. Esses fornecedores devem ser capazes de publicar componentes com suas respectivas especificações em um formato compreensível e em um lugar onde outras pessoas tenham acesso. Existem também aqueles que utilizam os componentes para construir ou *montar* aplicações, chamados *consumidores*, estes tem que estar aptos a localizar componentes que satisfaçam suas necessidades.



Modelo Fornecedor/Consumidor do Desenvolvimento baseado em componentes

DBC e Desenvolvimento Orientado a Objeto

Como mencionado anteriormente, embora o Desenvolvimento Baseado em Componentes (DBC) e o Desenvolvimento Baseado em Objetos embora busquem os mesmos resultados (desenvolvimento mais rápido, fácil e altamente produtivo de aplicações) e dividam algumas similaridades, a orientação a objeto pura tem provado ser desapontadora. Um grande problema é a dependência de objetos de baixa granularidade, forçando os desenvolvedores a trabalhar em um baixo nível, além de ser difícil para um desenvolvedor determinar o uso apropriado de uma classe escrita por outra pessoa e deste modo tornando-se difícil atingir a reutilização.

No entanto, o atual sucesso da abordagem de desenvolvimento de software baseado em componentes se deve principalmente ao paradigma da orientação a objetos. A abordagem OO(Orientada à Objeto) foi uma tentativa, até então, de reutilização em larga escala. Em Particular, as linguagens de programação OO reconhecem a necessidade de considerar o comportamento de um sistema como sendo um conjunto de ações inter-relacionadas, assim como a abordagem do DBC, centrando foco nas similaridades e diferenças de comportamento dos objetos no sistema e na captura dos relacionamentos entre esses objetos. Assim, no contexto do paradigma OO, quando um novo comportamento deve ser definido, este é geralmente criado através da reutilização ou herança, de um comportamento já existente no sistema, que é então especializado.

Em muitas situações, essa abordagem é benéfica. No entanto, podemos observar que a mesma tem algumas limitações. Por exemplo, quando devemos levar em consideração o aumento da produtividade, a necessidade de que todos os detalhes do comportamento de dado componente do sistema sejam entendidos por um desenvolvedor de aplicações não pode ser considerada muito produtiva. Pior ainda, para sistemas complexos, as hierarquias de comportamento herdado podem se tornar estruturas pesadas, difíceis de serem entendidas e cheias de interdependências que as tornam difíceis de ser modificadas. Se considerarmos que as partes a serem reutilizadas são desenvolvidas por terceiros, teremos, então realmente um ambiente com grandes

problemas de produtividade. A razão é simples: dificilmente entenderemos, de maneira detalhada, o comportamento de cada uma das partes.

Para que superemos essas dificuldades, é preciso que o comportamento e as interfaces desses componentes sejam claramente especificados. Além disso, é preciso que sejam fornecidos mecanismos que permitam a conexão dos componentes de forma flexível e que cada um destes seja substituído por outro similar a qualquer momento, sem a necessidade de modificações internas na estrutura dos componentes relacionados. As características dos Componentes como: alta granularidade, encapsulamento e estabelecimento de padrões - fazem com que sejam significativamente diferentes dos objetos utilizados no desenvolvimento orientado a objeto. A seguir, fornecemos uma comparação das técnicas:

Baseado em Componentes	**Orientado a Objeto**
Trabalha em um alto nível de abstração para a maior produtividade do desenvolvedor	Trabalha em um nível de detalhe muito baixo
Divisão dos desenvolvedores em *desenvolvedores* de aplicação e *experts* através do modelo *Fornecedor/Consumidor*. O encapsulamento mascara a complexidade enquanto API´s bem definidas facilitam o acesso a funcionalidade.	Requer intensivo treinamento e experiência antes que o desenvolvedor se torne produtivo
Apresenta um risco pequeno devido a natureza encapsulada dos componentes. A Herança é contida dentro do encapsulamento dos componentes eliminando as dependências	Desenvolvimento de alto risco devido A complexidade e as dependências
Conta com padrões largamente suportados, garantindo a interoperabilidade dos componentes.	Interoperacionalidade entre diferentes objetos é difícil. Usa ambientes de desenvolvimento altamente proprietários e requer desenvolvimento proprietário ao nível de API.
Larga indústria de suporte oferecendo ferramentas e bibliotecas padrão baseadas em componentes	Indústria de suporte limitada

Os componentes de software não precisam ser necessariamente implementados em uma linguagem orientada a objeto o principal requisito tem haver apenas com a definição da interface não importando se o código encapsulado pelo componente é orientado à objeto ou não.

Gerenciamento dos Componentes

O Gerenciamento dos Componentes é o ponto mais importante do DBC. Alguns desafios para os desenvolvedores que utilizam componentes são:

- Garantir modelos de componentes consistentes e sincronização das especificações, implementações e módulos executáveis.

- Definir claramente procedimentos de suporte e manutenção tanto dos fornecedores de softwares ou dos grupos de desenvolvimento internos que fornecem os componentes
- Rastrear o uso e dependências dos componentes para que as futuras notificações de erro não impactem negativamente o suporte seguro a produção de aplicações.

O DBC, como é implementado no modelo *fornecedor/consumidor*, resolvem muitos pontos nas organizações de um modo geral que previamente impediram o desenvolvimento da orientação à objeto. Como por exemplo a abstração da complexidade de um componente via interfaces bem definidas, que encoraja sua utilização por toda a empresa e aumenta a produtividade, em virtude de um trabalho menos complexo.

A funcionalidade requerida para gerenciar componentes inclui:

- Facilidades de um repositório multi-usuário para armazenar e catalogar componentes
- Capacidades de Browsing para possibilitar desenvolvedores acharem componentes apropriados
- Facilidades de biblioteca para organizar o crescimento da coleção de componentes básicos
- Gerenciamento da configuração para prover uma visão de negócio dos componentes (business-level)
- Análises de impacto quando mudanças são requisitadas.

Definição do Mercado DBC

O Desenvolvimento baseado em componentes, como um mercado de software, é composto de ferramentas e componentes requeridos para desenvolver, construir, montar e gerenciar aplicações utilizando esta tecnologia. Podemos classificar os segmentos desse mercado nos seguintes:

- Ferramentas de Análise, Modelagem e Design: Essas ferramentas permitem na modelagem de aplicações, objetos e/ou componentes a utilização de técnicas visuais e uma ou mais metodologias e notações. Muitas dessas ferramentas foram estendidas em vários níveis para suportar DBC.
- Ferramentas de construção e montagem de componentes: Esse segmento é relativo novo, com produtos disponíveis somente a um ano e meio. Essas ferramentas são usadas por desenvolvedores para construir componentes de softwares e ou montar com esses componentes, aplicações.
- Componentes de Softwares: é o segmento formado pelos próprios componentes que são vendidos para terceiros no propósito dos mesmos serem capazes de montar novas aplicações.
- Ferramentas RAD e Linguagens de Quarta geração (4GL): Empresas que comercializam linguagens de Quarta geração e/ou que fornecem desenvolvimento rápido de aplicações (RAD) para essas linguagens começarão à adicionar o suporte a DBC.
- Linguagens de Terceira Geração (3GL) e programação orientada à objeto (POO): Este mercado inclui ferramentas que facilitam a tarefa de programação usando uma variedade de linguagens. O fato poderem de ser utilizadas na construção de componentes, justificará a inclusão da renda de uma parte deste mercado no DBC.
- Gerenciamento de configuração de software (SCM): Há uma necessidade do desenvolvimento de ferramentas para o Gerenciamento de componentes como o

controle de versão, por exemplo. Este segmento estará desenvolvendo novas ferramentas para suportar o DBC em um ambiente distribuído nos próximos cinco anos.

- Qualidade de software automatizada: Assim como são desenvolvidos os componentes precisam ser testados. Ferramentas que testam a qualidade de software, serão melhoradas para testar também os componentes, incluindo a capacidade de trabalhar com aplicações contendo componentes que residem em diferentes sistemas em uma rede.
- Ferramentas profissionais para desenvolvimento Web: Aplicações Web serão baseadas em implementações componentizadas de aplicações e portanto esse segmento deve ser incluído em qualquer caracterização do mercado DBC.
- Ferramentas de Banco de Dados: Muitas empresas que fornecem SGBD`s estão adicionando capacidades as suas ferramentas para trazer uma arquitetura baseada em componentes aos seus bancos de dados

Além disso, algumas porções do mercado middleware (incluindo o usado para suportar objetos distribuídos, processamento de transações e acesso à dados) estão incluídos. O gerenciamento de aplicações de baseadas em componentes em um ambiente distribuído requer que isto seja desta maneira.

Finalmente, um Ambiente DBC eficiente não pode existir sem o uso dos repositórios. Repositórios representam, apesar de relativamente novo, mas um conceito das ferramentas de bancos de dados que vem crescendo rapidamente. Sua popularidade crescente é atribuída a necessidade de um depósito de dados comum para ferramentas que precisam interagir, uma parte crítica da implementação CBD. O gerenciamento de recursos dos componentes, requer repositórios para acolher e gerenciar esses recursos.