Uma Abordagem sobre o Panorama Atual do Desenvolvimento Baseado em Componentes

Simone Aparecida de Melo[1]
Orientador: Elio Lovisi Filho

Banca examinadora:
Luís Augusto Mattos Mendes **Eduardo Macedo Bhering**

[1]Departamento de Ciência da Computação – Universidade Presidente Antônio Carlos (UNIPAC)
Barbacena – MG – Brasil

melo-sa@uol.com.br

Resumo. *A demanda por sistemas de software cresce muito, e com essa demanda, aumentam também as exigências sobre o desenvolvimento de software: desejam-se sistemas maiores e mais complexos, entregues em menores prazos, com baixo custo e alta qualidade. Um caminho para se atingir essa meta é o desenvolvimento baseado em componentes.*

1. Introdução

A meta de construir aplicações a partir da composição de artefatos de *software* tem mais de três décadas. Em 1968 durante a Conferência Engenharia de Software da OTAN, McIlroy citou sobre a necessidade da indústria de *software* produzir famílias de componentes reusáveis. Segundo seu ponto de vista, desenvolvedores de *software* deveriam poder escolher componentes ajustados às suas necessidades específicas e estes componentes seriam usados em seus sistemas como artefatos caixa-preta.[SILVA E PRICE].

A necessidade de produzir mais, com maior qualidade e com menores riscos tem motivado o desenvolvimento de tecnologias para auxiliar os construtores de software na obtenção de sucesso em seus projetos.

Na década passada a abordagem de orientação a objetos parecia ser o mecanismo adequado para promover reuso e chegou-se a considerar a classe como a unidade de reuso. As aplicações orientadas a objetos têm exigido que os processos contemplem requisitos de reusabilidade, flexibilidade, e adaptação, entre outros. Verificou-se, porém, que a reutilização não é característica inerente da orientação a objetos, mas deve ser obtida a partir do uso de técnicas que produzam software reutilizável.

A abordagem de desenvolvimento orientado a componentes determina que uma aplicação seja constituída a partir de um conjunto de módulos (componentes) interligados. Krajnc vê esta abordagem como uma evolução natural da abordagem de orientação a objetos, em que um componente corresponde a um conjunto de classes inter-relacionadas, com visibilidade externa limitada [SILVA E PRICE].

Apesar das recentes e constantes pesquisas na área de Desenvolvimento Baseado em Componentes (DBC), ainda há carência de métodos, técnicas e ferramentas que suportem tanto o desenvolvimento quanto a reutilização de componentes, em aplicações de um determinado domínio. Outros problemas associados a componentes estão relacionados à busca e seleção de componentes para reuso. A dificuldade de localização de componentes está associada à inexistência de padrões de repositório e mecanismos de busca que permitam a potenciais usuários selecionar componentes que supram suas necessidades. A dificuldade de seleção está associada às deficiências apresentadas pelos mecanismos de descrição de componentes em especificar o que componentes fazem e como interagem.

As razões para o atual interesse por DBC advêm da maturidade das tecnologias que permitem a construção de componentes e a combinação destas para o desenvolvimento de aplicações, bem como o atual contexto organizacional e de negócio que apresenta mudanças quanto a como as aplicações são desenvolvidas, utilizadas e mantidas [SPAGNOLI e BECKER, 2003].

Neste artigo, serão apresentados os conceito de componentes e de DBC, além das vantagens e desvantagens da utilização dos mesmos. Também será apresentada a definição de alguns métodos utilizados, com Catalysis e UML Components e de uma tecnologia para implementação de componentes.

2. Componentes

Componentes de software são artefatos autocontidos, facilmente identificáveis que descrevem e/ou executam funções específicas e tem interfaces claras, documentação apropriada e uma condição de reuso definida [SPAGNOLI e BECKER, 2003].

Um componente é uma parte do sistema independente e substituível, que encapsula, dentro de si, seu projeto e implementação, e oferece serviços, por meio de interfaces bem definidas, para o meio externo.

Autocontido é uma característica desejável dos componentes na qual eles podem ser reusáveis sem a necessidade de depender de outros componentes. E a identificação é uma outra característica do componente de estar contido em um único lugar ao invés de espalhados e misturados com outros artefatos de softwares ou documentação.

A reutilização é um processo de criar sistemas de software a partir de softwares existentes, ao invés de construí-los a partir do zero [GIMENES e HUZITA, 2005].

Segundo Pressman, os componentes de software são criados por meio de uma série de conversões que definem as exigências do cliente para criar um código executável [GABRIEL].

Um componente é uma implementação de software que pode ser executada em um dispositivo físico ou lógico, integrando dados e funções, com alto encapsulamento, alta coesão e baixo acoplamento. Ele deve implementar uma ou mais interfaces – serviços oferecidos a outros componentes, declaradas em suas interfaces providas – e declarar os serviços dos quais ele depende para funcionar, suas interfaces requeridas.

Um modelo das exigências é convertido num projeto, este projeto é convertido numa linguagem que especifica a estrutura de dados do software, os atributos procedimentais e os requisitos relacionados. A linguagem é processada por um tradutor que a converte em instruções executáveis em máquina.

Características do paradigma de programação por componentes incluem a redução nos custos e no tempo de desenvolvimento do software. Além disso, a utilização de elementos independentes e a interação entre eles através de interfaces bem definidas, fazem de componentes unidades de configuração que permitem a implementação e a manutenção de programas distribuídos de maneira eficiente. O desenvolvimento baseado em componentes pode ser realizado empregando modelos de componentes já existentes, tal como Enterprise JavaBeans (EJB) desenvolvido pela Sun Microsystems, .NET da Microsoft e o Modelo de Componentes do CORBA (CCM), especificado pela OMG.

Outra característica importante de um componente de software é a reusabilidade. Um componente deve ser projetado e implementado para que, possa ser reutilizado em outros sistemas. Na década de 1960, falava-se muito em bibliotecas de sub-rotinas que reusavam algoritmos bem definidos, mas tinham um domínio de aplicação limitado [GABRIEL]. Esse tipo de biblioteca, além de economizar esforço de desenvolvimento, permite a produção de programas já testados e amplamente utilizados.

A visão de reuso mudou com o passar dos tempos, hoje além dos algoritmos utiliza-se também estruturas de dados e artefatos de projeto. Um componente reutilizável engloba tanto dados como processamento num único pacote.

O reuso de software implica no aproveitamento de componentes de software já desenvolvido, numa situação diferente daquela para o qual eles foram projetados. A reutilização não se refere apenas ao código fonte, mas também, estruturas do projeto, especificações, planos de testes e documentação.

Os componentes têm pontos de interconexão chamados de interfaces que podem ser classificadas em dois tipos [GIMENES e HUZITA, 2005]:

- Interfaces fornecidas (*provided interfaces*): define os serviços oferecidos pelo componente por meio de operações. Durante a implementação, essas interfaces devem ser identificadas e definidas separadamente.
- Interfaces requeridas (*required interfaces*): define os serviços que o componente necessita de outros componentes. Componentes se conectam por meio da interface requerida de um com a interface fornecida de outro. Se um componente for autocontido, não existirão interfaces requeridas. E elas também devem ser definidas explicitamente.

Cada interface consiste em serviços especificados, mediante uma ou mais operações, sendo cada uma delas separadamente identificada e especificada de acordo com seus parâmetros de entrada e saída e respectivos tipos estabelecidos [GIMENES e HUZITA, 2005]. Essas definições constituem a assinatura (*signature*) da interface.

A única forma de interação de um componente com outros é através de suas interfaces.

A idéia de componentes, usando a notação UML, é ilustrada na Figura 1 . Em UML as interfaces são representadas pelos relacionamentos de dependência.

É um exemplo simples de um componente que processa endereços de diferentes países. Esse componente pode ser utilizado para criar e manter endereços postais em diferentes formatos.

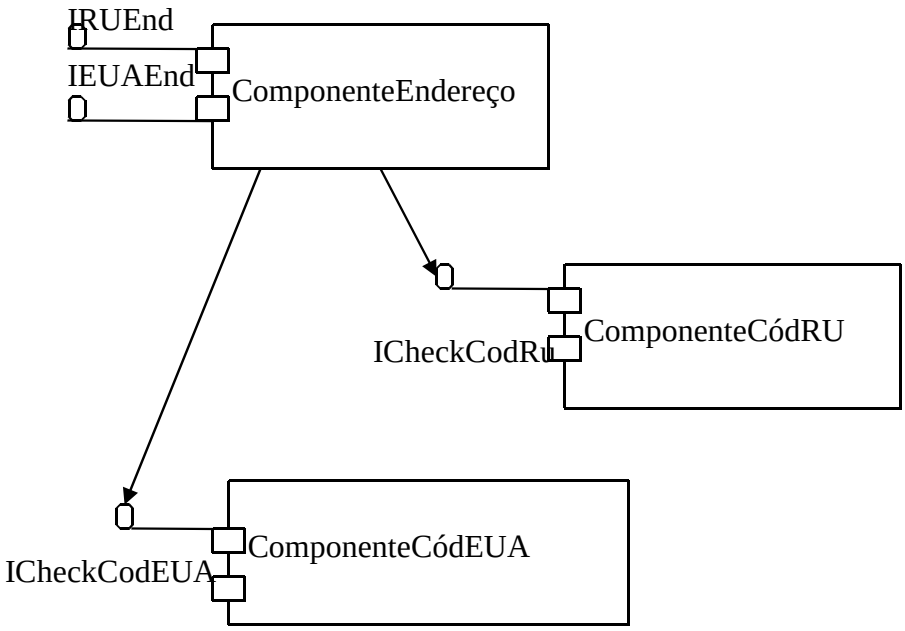

Figura 1. Ilustração do ComponenteEndereço [GIMENES e HUZITA, 2005]

3. Desenvolvimento Baseado em componentes

O desenvolvimento baseado em componentes é uma atividade de Engenharia de Sistemas baseado em componentes que ocorre em paralelo com a engenharia de domínio; que tem como objetivo identificar, construir, catalogar e disseminar um conjunto de componentes de software para que os engenheiros de software possam utilizá-los em novos sistemas ou em sistemas já existentes. [ALMEIDA e BIANCHINI, 2002]

A engenharia de domínio realiza o trabalho necessário para estabelecer um conjunto de componentes de software que podem ser reusados pelo engenheiro de software. Esses componentes são então transportados através da “fronteira” que separa a engenharia de domínio do desenvolvimento baseado em componentes.

A Figura 2 mostra como funciona a engenharia de domínio. Note que existe uma modelagem de domínio, que pode ser desenvolvida independentemente do fato de se pretender utilizar o DBC.

O propósito dessa modelagem é entender os conceitos do domínio e seus relacionamentos, incluindo as tarefas realizadas no domínio, denominadas casos de uso, que representam as diferentes funções.

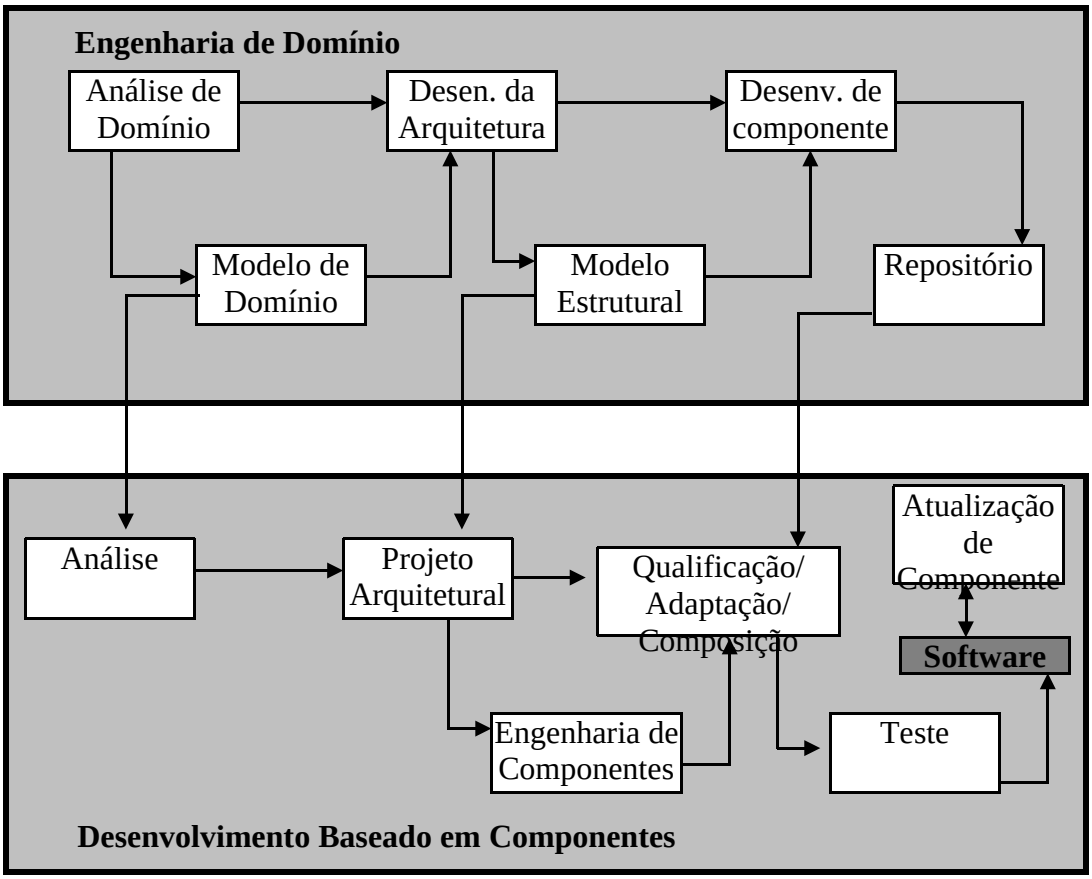

Figura 2. Engenharia de Domínio e DBC [PRESSMAN, 2002]

A Engenharia de Software baseada em componentes é uma disciplina que estuda maneiras para que haja a construção rápida de sistemas a partir de componentes reusáveis onde componentes e frameworks possuem propriedades certificadas, que provêem a base para prever as propriedades dos sistemas construídos a partir dos componentes [TOMITA, 2004]. Ainda, há o interesse no entendimento de como as tecnologias de componentes de software são (ou deveriam ser) utilizadas em serviço de toda uma área de engenharia baseada em componentes de software.

O desenvolvimento de software baseado em componentes tem como objetivo a definição de componentes interoperáveis, com interfaces bem definidas, evidenciando os tipos de relacionamentos permitidos por este componente. Desta forma, a complexidade no desenvolvimento é reduzida, assim como os custos, através de reutilização de componentes exaustivamente testados [GABRIEL].

O software tem que ser de boa qualidade para atender às necessidades dos respectivos usuários. Para que uma empresa consiga desenvolver um software de maneira previsível e em determinado período, com utilização eficiente e eficaz de recursos, ela precisa ter um processo que controle todos os passos, que vão da concepção até a entrega do produto final.

Um dos principais objetivos da construção de software baseado no uso de componentes é reduzir a complexidade, o custo e o tempo de todo o processo de desenvolvimento, além de aumentar a confiabilidade dos sistemas, uma vez que os componentes que já foram testados. Existem estudos que comprovam que a reutilização de componentes é capaz de aumentar tanto a qualidade quanto a produtividade de todo o processo de desenvolvimento de software.

4. Perspectivas de desenvolvimento

O desenvolvimento de software a partir de DBC pode considerar o desenvolvimento de componentes ou o desenvolvimento com componentes. A primeira perspectiva engloba as atividades envolvidas na concepção e implementação de um componente, devendo existir a preocupação em gerar a documentação necessária para que o componente possa ser posteriormente reutilizado. A segunda perspectiva considera a existência de componentes de software implementados e que estejam disponíveis para serem selecionados e reutilizados na composição dos sistemas.

As atividades relacionadas com o desenvolvimento de componentes envolvem atividades de especificação do componente, onde métodos, técnicas, processos como Catalysis e UML Components podem ser empregadas para ajudar na qualidade do processo de desenvolvimento. Nos próximos itens serão explicados estes dois métodos.

4.1. Catalysis

Catalysis (que em Português significa catálise) tem origem na Química por conta do conceito de catalisadores, que são utilizados na aceleração da taxa de um processo ou reação, que normalmente se apresenta em quantidades pequenas e não afetam o término dessa reação. Um catalisador permite que reações ou processos sejam acelerados ou retardados com condições bem controladas.

Vários conceitos foram incorporados pelo método Catalysis para o desenvolvimento de software baseado em componentes. O método utiliza-se de UML (Unified Modeling Language – Linguagem de Modelagem Unificada), que é um método aberto usado para especificar, visualizar, construir e documentar os artefatos de um sistema de software, com algumas alterações, para a especificação dos diferentes diagramas que compõem o processo.

Ele permite que o processo de desenvolvimento seja adaptado às características do projeto, possibilitando assim a definição da melhor seqüência de atividades e artefatos para cada caso [GIMENES e HUZITA, 2005].

Fundamenta-se nos princípios de abstração, precisão e componentes "*plug-in*". O princípio abstração orienta o desenvolvedor na busca dos aspectos essenciais do sistema, dispensando detalhes que não são relevantes para o contexto do sistema. O princípio precisão tem como objetivo descobrir erros e inconsistências na modelagem e o princípio componentes "plug-in" suporta o reuso de componentes para construir outros sistemas.

Outros princípios fundamentais do método Catalysis [GIMENES e HUZITA, 2005] são: construção de um modelo de domínio do sistema que enfatiza a independência entre o

domínio; forte ênfase nos conceitos de abstração e refinamento para representar os relacionamentos, que cria uma série de fatorações, extensões e transformações que visam o rastreamento dos requisitos até o código; ênfase na especificação de pré-condições, pós-condições e invariantes; procedimentos e diagramas que apóiam o particionamento do sistema, o projeto e a conexão dos componentes; forte articulação do processo de desenvolvimento e uso de ciclos de vida rápido, iterativo e incremental.

O padrão cobre os três níveis de Catalysis, e é realizado em quatro passos [CATALYSIS, 1999]:

- Definir problema: onde é dada ênfase no entendimento do problema, especificando-se "o quê" o sistema deve atender para solucionar o problema;
- Especificar Componentes: onde é descrito o comportamento do sistema de uma forma não ambígua;
- Projetar componentes;
- Implementar.

4.2. UML Components

É um processo de desenvolvimento de software baseado em componentes, que trata do problema de especificar e arquitetar sistemas baseados em componentes. Ele não é um processo de desenvolvimento completo, já que não inclui atividades relacionadas ao processo gerencial, e enfatiza as etapas de análise, projeto e, em menor grau, implementação.

Uma característica importante de UML Components é o fato dele ser baseado na linguagem UML. Isto permite que usuários do método utilizem ferramentas já existentes para UML para a construção de seus sistemas baseados em componentes. O processo propõe extensões da linguagem UML através de seu mecanismo de estereótipos para a especificação das interfaces de componentes, de componentes propriamente ditos e de suas implementações.

O método UML Components é iterativo e define um conjunto de passos para que, a partir da especificação de um conjunto de casos de uso, seja construído um sistema de software baseado em componentes que simplifique o trabalho de lidar com mudanças (evolução / substituição de componentes).

As fases definidas pelo método são [TOMITA, 2004]: (1) especificação, onde é gerado um conjunto de especificações de componentes e uma arquitetura de componentes, (2) provisionamento, que garante que os componentes necessários estão disponíveis, seja através de seu desenvolvimento, aquisição ou reuso e (3) montagem, que compõe os diversos componentes entre si e com artefatos de software pré-existentes, incluindo a interface com o usuário.

A fase de especificação ainda é dividida nas seguintes sub-fases: identificação de componentes, cujo objetivo é identificar um conjunto inicial de interfaces (e o nome de seus métodos) e componentes e combiná-los em uma arquitetura inicial; interação de componentes, onde através de modelos de interação, são descobertas mais operações com suas assinaturas completas e as interfaces podem ser refinadas, agrupadas ou divididas; e especificação de componentes, em que é feito uma especificação detalhada das interfaces, incluindo suas restrições, em termos de pré e pós-condições.

5. Vantagens

O uso de componentes traz mudanças no processo de desenvolvimento e com esses processos algumas vantagens.

É utilizado mais "*Information hiding*", onde apenas interfaces são usadas e unidade de *packaging* reforça o encapsulamento (tudo está lá dentro).

Polimorfismo com Very Late Binding [JACQUES]: Software baseado em Componentes é extensível por definição e novos componentes podem ser acrescentados a qualquer momento.

Não tem herança entre componentes para diminuir acoplamento ou pode ter herança internamente.

É seguro, pois a instalação de novos componentes no ambiente do usuário não deve invalidar os componentes já instalados: "Um Componente deve ser aberto à extensão, mas fechado à modificação" [JACQUES]. Portanto, gerenciamento de memória é muito mais importante.

Tem que ter *garbage collection* [JACQUES] porque um componente nunca pode saber quando um outro componente pode ser liberado.

6. Suporte das Linguagens

O componente é uma unidade de composição e precisa de um suporte adequado das linguagens de programação para que ele possa funcionar adequadamente. Aliado a isso é preciso mecanismos de segurança para que os componentes não possam afetar a integridade de outros.

Finalmente, é necessário um mecanismo para definir interfaces: “O que ofereço” (interface de cima), “O que preciso” (interface de baixo), como mostra a Figura 3 em um sistema de locadora de vídeo, onde a interface se encaixa com a necessidade do cliente.

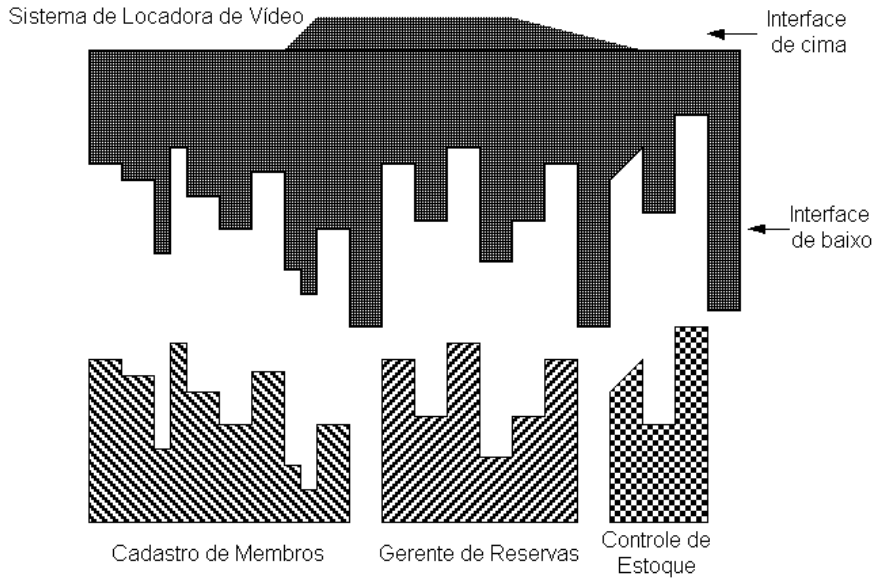

Figura 3. Mecanismo para definir interfaces [JACQUES]

Existem algumas tecnologias que suportam o DBC, como o Enterprise JavaBeans (EJB) que será explicada nesse item.

6.1 Enterprise JavaBens (EJB)

A especificação J2EE (Java 2 *Plataform, Enterprise Edition*) definida pela Sun Microsystems, fornece uma abordagem baseada em componentes para o projeto, desenvolvimento, composição e utilização de aplicações corporativas. A plataforma J2EE define um conjunto de APIs (*Application Programming Interface*) entre as quais está EJB, implementada através dos pacotes e subpacotes `javax.ejb` [SPAGNOLI e BECKER, 2003].

Os Enterprise Java Beans foram criados em Março de 1998, na primeira especificação da Sun com o intuito de promover uma arquitetura de objetos distribuídos pela Internet.

Entretanto, EJB não é apenas uma API e caracteriza-se como uma arquitetura de componentes para o desenvolvimento e utilização de aplicações corporativas baseadas em componentes distribuídos. Compreende entre as suas definições a especificação de um modelo de componentes flexível que fornece um conjunto de convenções, definições e regras para a implementação de componentes utilizando a linguagem Java.

Para se compreender melhor o EJB, é necessário primeiro o entendimento das três principais entidades da arquitetura:

- Enterprise bean ou Bean: são os componentes do EJB que ficam hospedados nos containers EJB;
- Containers EJB: tem a função de gerenciar todos os aspectos do bean durante a execução, incluindo: acesso remoto, segurança, persistência, transações, concorrência e acesso a recursos;
- Servidor EJB: embora não exista uma clara distinção em relação aos *containers* EJB, o papel do servidor é apresentar diferentes *container*, uma para cada tipo de bean, e prover serviços que permitam que os *containers* gerenciem os beans.

A Figura 4 mostra como é o relacionamento entre essas entidades.

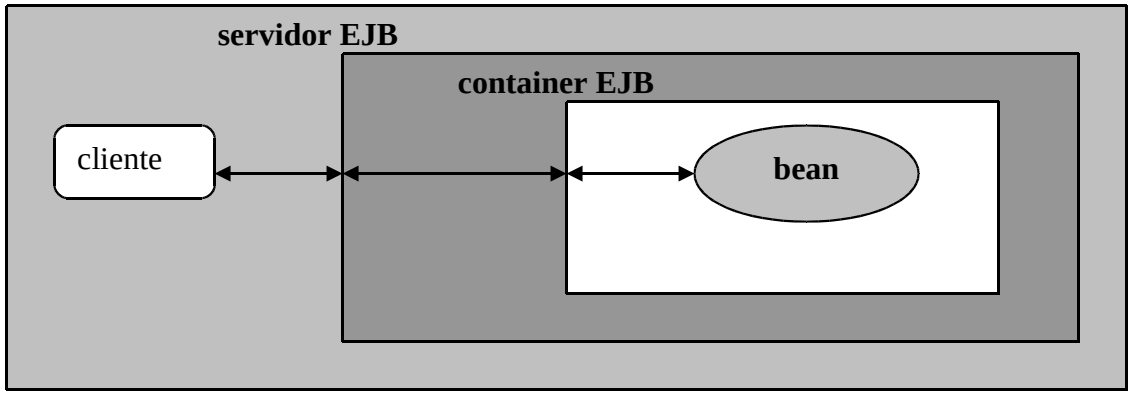

Figura 4. Principais conceitos da arquitetura EJB [SPAGNOLI e BECKER, 2003]

O desenvolvimento dos componentes EJB corresponde à codificação de um conjunto de arquivos, tendo-se a definição classes e interfaces. Os diferentes arquivos gerados implementam o funcionamento do componente e as interfaces necessárias para a execução, bem como as interfaces que identificam os serviços que estes componentes disponibilizam a seus clientes.

Em virtude dos diferentes tipos de beans, existe a possibilidade de se ter diferentes cenários no desenvolvimento com componentes EJB em que diferentes classes e interfaces devem ser codificadas.

Os enterprise beans apresentam-se como componentes de código, os quais implementam serviços e podem ser reutilizados, embora não exista nenhuma preocupação em gerar documentação apropriada para esse reuso. A definição quanto aos tipos de beans existentes, a sua forma de interação e quais recursos estes componentes tem acesso é definido pelo modelo de componentes EJB.

Um componente EJB possui três tipos fundamentais: entidade, sessão, dirigido por mensagem.

6.1.1. Beans de Entidade (Entity Beans)

Modelam conceitos de negócios que podem ser expressos por nomes e esta é uma regra importante quando na sua modelagem de sistema você precisa eleger um candidato a ser um Bean de Entidade. Estes tipos de beans modelam realmente um dado no Banco de Dados onde uma instância representa uma linha na tabela de um Banco de Dados.

Existem muitas vantagens em se usar Beans de Entidade ao invés de acessar a base de dados diretamente. Estes beans promovem um simples mecanismo de acesso e alteração de dados. Quando um novo bean é criado, um novo registro deve ser inserido na base de dados e uma instância do bean associada a este dado. Conforme o bean é usado e seu estado é alterado estas mudanças devem estar sincronizadas com a base de dados. Este processo de coordenação dos dados do database com a instância do bean denomina-se persistência.

Eles são responsáveis pelos dados de negócios (*Core Business Data*), Com relação ao mapeamento com as tabelas, um bean de entidade não necessita estar mapeado para uma única tabela, pode ser representado por várias porque na fase de *design* nem toda tabela de seu esquema de persistência necessita estar mapeada para um bean de entidade.

6.1.2. Beans de Sessão (*Session Beans*)

Os beans de sessão representam as ações de negócios. Por isto os *Session Beans* atendem mais o lado da aplicação cliente que necessita acionar um serviço.

Os *Session Beans* são divididos em 2 tipos básicos: *stateless e stateful*. Um bean de sessão stateless é uma coleção de serviços cada qual sendo representado por um método específico. Stateless tendem para ser de propósitos gerais ou reusáveis como um serviço de software.

Enquanto o bean de sessão *stateful* é uma extensão da aplicação cliente. Quando um cliente obtém uma conexão com o EJB stateful o seu estado é mantido entre as chamadas do mesmo cliente até o cliente remover esta conexão. Isto significa que o estado dos atributos de classe do Stateful permanece inalterado enquanto o cliente mantém a conexão.

Esta característica do Stateful se chama estado conversacional porque representa uma conversação contínua entre o mesmo e o cliente.

6.1.3. Beans Orientados a Mensagens (*Message-Driven Beans*)

Beans orientados a mensagens processam alguma lógica de negócios usando mensagens JMS enviadas para uma destinação particular, ou seja, consomem mensagens JMS através da tecnologia EJB.

A grande diferença entre beans orientados a mensagens e beans de sessão ou de entidade é que os beans orientados a mensagens não possuem interfaces *home* nem interface *remote.* Então podemos deduzir que estes beans são completamente escondidos do cliente. O único meio de clientes comunicarem com os beans orientados a mensagens seria enviando uma mensagem para um destinatário JMS.

7. Problemas Atuais no Desenvolvimento Baseado em Componentes

O DBC se preocupa com a criação de componentes que possam ser reutilizados em outras aplicações. Para que a reutilização possa ser efetiva, deve-se considerá-la em todas as fases do processo de desenvolvimento do *software*. Portanto, o DBC deve oferecer métodos, técnicas e ferramentas que suportem desde a identificação (que é encontrar o componente certo) e especificação dos componentes, referente ao domínio do problema, até o seu projeto e implementação em uma linguagem executável.

Além disso, o DBC deve empregar inter-relações entre componentes já existentes, que tenham sido previamente testados, visando reduzir a complexidade e o custo de desenvolvimento do software. Uma documentação, que inclua conjuntos de testes para cada componente, deve ser oferecida.

O desenvolvimento de software usualmente sofre pressões de tempo e custo. Da mesma forma que a reutilização pode ajudar, também pode requerer esforço adicional, que se deve à possível reutilização futura dos componentes e à necessidade de serem flexíveis, estáveis e corretos. Tal esforço é consideravelmente superior ao necessário para qualquer outro software desenvolvido para uma aplicação específica [GIMENES e HUZITA, 2005].

Apesar das recentes e constantes pesquisas na área de DBC, ainda há carência de métodos, técnicas e ferramentas que suportem tanto o desenvolvimento quanto a reutilização de componentes, em aplicações de um determinado domínio. Outro problema importante relaciona-se com a dificuldade de integração das diferentes técnicas e ferramentas que apóiam o DBC. Considerando o caso de componentes distribuídos, como ocorre na rede Internet, com plataforma cliente e servidor, o problema é ainda maior.

8. Conclusão

Tendo em vista a necessidade de desenvolver software de boa qualidade, para um mercado exigente, percebe-se a necessidade de utilizar algum método que garanta o controle do processo. E um desses métodos (DBC) foi apresentado neste trabalho, sendo mostrado os seus conceitos e suas técnicas.

O processo de software baseado em componente representa um conjunto de ferramentas poderosas para auxiliar as equipes de desenvolvimento, independente da metodologia utilizada, Catalysis ou UML Components. Os benefícios conquistados com a reusabilidade garantem, na maioria das vezes, prazos e custos reduzidos.

Uma receita ideal para a resolução de qualquer problema de desenvolvimento de software, dificilmente será encontrada, a complexidade e inclusão de novos conceitos e possibilidades acontecem a cada dia. Resta aos desenvolvedores buscar estas tecnologias e de posse deste conhecimento, aplicarem o método que melhor atenda os requisitos de qualidade e produtividade.

9. Bibliografia

ALMEIDA, Eduardo Santana de, BIANCHINI, Calebe de Paula (2002). **Um Padrão para o Desenvolvimento de Software: Baseado em Componentes Distribuídos**. Disponível em http://lens.cos.ufrj.br/sugarloafplop/2002/papers/Paper2C.pdf. Consultado em 22 ago 2005.

BATISTA, Thaís**. Arquitetura de Software**. Disponível em http://www.dimap.ufrn.br/~ jair/mes/slides/CORBACCM.pdf. Consultado em 30 out 2005.

CATALYSIS (1999). **Catalysis Enterprise Components with UML**. Disponível em http://www.catalysis.org. Consultado em 27 out 2005.

CIMM. **Centro de Informação Metal Mecânica.** Disponível em http://www.cimm.com.br/construtordepaginas/htm/3_25_2433.htm. Consultado em 22 ago 2005.

GABRIEL, Gláucia. O **Processo de Desenvolvimento Baseado em Componentes: O impulso das novas tecnologias.** Disponível em: https://portaljava.dev.java.net/files/documents/353/13070/DesenvolvimentoComponentes.pdf. Consultado em 30 out 2005

JACQUES. **Aspectos técnicos do desenvolvimento baseado em componentes** http://jacques.dsc.ufcg.edu.br/cursos/map/html/comp/desenvolvimento.htm. Consultado em 30 out 2005.

TOMITA, Rodrigo Teruo (2004). **Um Ambiente para Desenvolvimento de Sistemas Baseados em Componentes** http://www.ic.unicamp.br/~ra992432/msc/projeto_ic.pdf. Consultado em 30 out 2005

SILVA, Ricardo Pereira. PRICE, Roberto Tom. **Suporte ao Desenvolvimento e Uso de Componentes Flexíveis.** Disponível em http://www.inf.ufsc.br/~sbes99/anais/SBES-Completo/02.pdf. Consultado em 22 ago 2005.

SPAGNOLI, Luciana de Araújo. BECKER, Karin (2003). **Um estudo sobre o Desenvolvimento Baseado em Componentes.** Disponível em: http://www.inf.pucrs.br/tr/tr026.pdf. Consultado em 30 out 2005.

GIMENES, Itana Maria de Souza. HUZITA, Hatsue Moriya. **Desenvolvemento Baseado em Componentes: Conceitos e Técnicas**.Ed. Ciência Moderna,Rio de Janeiro: 2005.