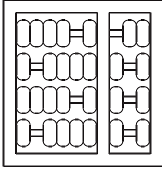

INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Um Processo para o Desenvolvimento Baseado em Componentes com Reuso de Componentes

Patrick Henrique da Silva Brito
Maria Antonia Martins Barbosa
Paulo Asterio de Castro Guerra
Cecília Mary Fischer Rubira

Technical Report - IC-05-022 - Relatório Técnico

September - 2005 - Setembro

The contents of this report are the sole responsibility of the authors.
O conteúdo do presente relatório é de única responsabilidade dos autores.

Um Processo para o Desenvolvimento Baseado em Componentes com Reuso de Componentes

Patrick Henrique da Silva Brito   Maria Antonia Martins Barbosa
Paulo Asterio de Castro Guerra   Cecília Mary Fischer Rubira

Resumo. *O mercado de software atual está cada vez mais sujeito a fortes restrições de prazos e custos com exigência de alta qualidade. Por agilizar o desenvolvimento e torná-lo menos custoso a longo prazo, o desenvolvimento baseado em componentes (DBC) vem sendo adotado largamente. O objetivo deste trabalho é propor um processo de DBC com alta qualidade, que maximize a reutilização de componentes prontos durante o desenvolvimento. Uma característica importante do processo é o fato dele ser genérico, no sentido de ser constituído de atividades gerais, comuns à maioria das metodologias de DBC atuais. Com isso, pretende-se que ele seja facilmente adaptável aos vários processos utilizados, o que facilita sua utilização prática. Além disso, o método proposto foi adaptado ao processo UML Components.*

Palavras-chave: *Processo de desenvolvimento, desenvolvimento baseado em componentes, DBC, desenvolvimento com reuso.*

Abstract. *The current software market each time more have restrictions of stated periods and costs, beyond the requirement of high quality products. Due to speed the development and to become it cheaper, the component-based development (CBD) is being wide adopted. The objective of this work is to present a process of CBD with high quality, that maximizes the components reuse during the development. An important characteristic of the process is its genericity, that facilitates its adaptation to the existing development processes. This genericity facilitates its practical adoption. Moreover, the considered method was adapted to the UML Components process.*

Keywords: *Software development process, component-based development, CBD, software development with reuse.*

1 Introdução

Cada vez mais o desenvolvimento de sistemas computacionais está sujeito a fortes restrições de prazos e custos com exigência de alta qualidade [22]. Em busca do cumprimento de

objetivos tão antagônicos, o desenvolvimento baseado em componentes (DBC) vem sendo adotado atualmente, por proporcionar reutilização em larga escala e conseqüente redução no custo. O objetivo deste trabalho é propor um processo de desenvolvimento de sistemas baseado em componentes com alta qualidade, conciliando as vantagens do DBC com os requisitos de qualidade exigidos para desenvolvimento de sistemas críticos.

O DBC é definido como a construção de software através da integração planejada de componentes reutilizáveis [22]. Um aspecto fundamental do paradigma DBC é a separação entre especificação de um componente de software e sua implementação [24]. Uma especificação de componente define o comportamento observável externamente do componente, com a abrangência e precisão necessárias para sua integração em diferentes sistemas, porém abstraindo detalhes de qualquer implementação específica. Uma implementação de componente fornece um modelo concreto para instanciação, em diferentes ambientes, de componentes que realizam uma dada especificação. A separação entre especificação e implementação de componente visa, fundamentalmente, permitir a divisão da atividade de desenvolvimento de software em duas atividades independentes: a produção de componentes de software e a integração de sistemas [8].

Em um processo de DBC é possível enfatizar dois aspectos distintos [22]: (i) desenvolvimento para reuso, que enfatiza a produção de componentes propícios a serem reutilizados; e (ii) desenvolvimento com reuso, que consiste na composição de sistemas a partir de componentes já existentes. Apesar dessas duas abordagens de desenvolvimento, os processos atuais apresentam limitações no cumprimento desses requisitos. Com o intuito de suprir a carência do desenvolvimento com reuso, este trabalho propõe um processo de DBC que visa maximizar a reutilização de componentes já existentes de forma sistemática. Além do reuso, a sistemática do processo deve melhorar a qualidade final do sistema. Outra característica importante desse processo é o fato dele ser genérico e facilmente adaptável a outros processos de desenvolvimento existentes, tais como processo UML Components [5] e o processo Catalysis [9]. A título de ilustração, o processo proposto neste relatório foi adaptado ao processo de DBC UML Components.

O restante do relatório técnico está organizado como segue. Na Seção 2 são apresentados os fundamentos teóricos necessários para contextualizar o problema do DBC tratado neste trabalho. A Seção 3 apresenta o processo proposto, através do *workflow* de suas atividades gerais, seguido do detalhamento de cada uma. Na Seçãp 4, é apresentada brevemente a forma como o processo deve ser adaptado a outros processos, mostrando sua adaptação ao processo UML Components. Finalmente, na Seção 5 são apresentadas algumas conclusões e trabalhos futuros.

2 Fundamentos de DBC e Arquitetura

2.1 Processos de Desenvolvimento de Software

O objetivo de uma **metodologia** ou **processo** de desenvolvimento de software é sistematizar as atividades de construção de programas, distribuindo a sua complexidade em quatro etapas gerais [19]: (i) definição do problema; (ii) definição da estrutura; (iii) desenvolvimento do sistema; e (iv) atividades de manutenção. O uso de processos disciplinados de

desenvolvimento reduz o número de falhas introduzidas no sistema [3, 2]. A fim de atingir a sistemática necessária para isso, os processos devem possuir um conjunto de métodos estruturados que detalhem e auxiliem o desenvolvimento de sistemas nas suas fases [19]. Esses **métodos** são procedimentos sistemáticos que usam notações bem definidas para alcançar seus objetivos. Assim, os métodos são compostos de atividades e diretrizes de desenvolvimento, normalmente orientadas a um processo específico. De uma maneira geral, os métodos devem incluir informações sobre [22]: (i) os modelos produzidos; (ii) as restrições aplicadas a esses modelos; (iii) diretrizes de projeto; e (iv) a sequência de atividades a serem seguidas.

A maioria dos processos de desenvolvimento de software atuais mapeiam o desenvolvimento do sistema em fases, que são detalhamentos das quatro etapas gerais apresentadas anteriormente:

1. Definição do problema:
   - **Especificação de Requisitos.** O objetivo principal desta fase é a identificação dos serviços que devem ser automatizados pelo sistema. Além disso, outras informações e restições importantes devem ser igualmente documentadas. De acordo com a sua natureza, esses requisitos podem ser classificados em duas categorias: (i)requisitos funcionais, que representam os comportamentos que um sistema deve apresentar diante de certas ações de seus usuários; e (ii) requisitos não-funcionais, que quantificam determinados aspectos do comportamento do sistema.
2. Definição da Estrutura:
   - **Especificação da Arquitetura -** Baseado principalmente nos requisitos não-funcionais especificados na fase anterior, se dá início à especificação da arquitetura do sistema. Normalmente, essa fase consiste na escolha do estilo arquitetural que melhor ofereça as propriedades arquiteturais desejadas.
3. Desenvolvimento do Sistema:
   - **Análise -** Nesta etapa, os desenvolvedores se concentram na identificação das entidades principais para a implementação dos requisitos do sistema. As entidades identificadas são representadas a partir das suas informações de estado (dados internos) e das principais operações a serem oferecidas.
   - **Projeto -** A etapa de projeto consiste no refinamento dos modelos gerados na análise. Esse refinamento visa a aplicações de padrões, por exemplo, os padrões de projetos (do inglês *design patterns*) sugeridos por Gamma *et.al*. [12]. Ainda nessa etapa, as interações entre as entidades são revistas e detalhadas.
   - **Implementação -** Esta etapa consiste na materialização dos modelos especificados e detalhados no projeto em uma linguagem de programação. Normalmente é feito um mapeamento direto das estruturas da modelagem para a linguagem de programação. Quando isso não é possível, se faz necessário a utulização de um modelo de mapeamento, por exemplo, os modelos EJB [17], CCM [16], DCOM [21] e COSMOS [14].

- **Execução dos Testes -** Na etapa de testes, o sistema é verificado para se certificar de que os requisitos especificados estão implementados de maneira satisfatória.
- **Distribuição** (do inglês *deployment*) - Esta etapa consiste na entrega e instalação do sistema nos diversos ambientes onde serão utilizados pelos clientes.

4. Atividades de Manutenção:

- Corretivas - As atividades desta etapa consistem de ações com o objetivo de corrigir iconsistências entre os requisitos especificados e o sistema implementado;
- Evolutivas - As ações de manutenções evolutivas adicionam novas funcionalidades aos sistemas já desenvolvidos. Sendo assim, com a aplicação dessas atividades, o tempo de vida dos sistemas costumam aumentar.

Essas fases apresentadas constituem as etapas de execução do desenvolvimento de um software. Porém, além do processo de desenvolvimento, que especifica e implementa o sistema a partir dos seus requisitos, para se desenvolver um sistema de software é necessário seguir ao mesmo tempo um processo gerencial [5], que esquematiza atividades, planeja liberações, aloca recursos e monitora progressos. Na literatura encontramos alguns exemplos de abordagens que incluem um dos processos ou ambos. Por exemplo, o processo Catalysis [9] e o processo UML Components [5] são basicamente processos de desenvolvimento. Enquanto isso, o IBM-Rational Unified Process (RUP) [13] trata tanto do processo gerencial quanto do processo de desenvolvimento.

Este trabalho está concentrado em processos de desenvolvimento de software e para facilitar a leitura, quando o termo simples processo for utilizado, se trata de uma referência a um processo de desenvolvimento.

2.1.1 Processos de Desenvolvimento Baseado em Componentes

Com a popularização do DBC, a necessidade de novos processos voltados para esse paradigma é uma realidade. Isso acontece porque os processos de desenvolvimento tradicionais não se adequam totalmente ao desenvolvimento de sistemas baseados em componentes. Mais especificamente, esses processos devem conter fases e métodos que também ofereçam técnicas que permitam o empacotamento de componentes com o objetivo específico de serem reutilizados [22]. Essa reutilização sistemática deve levar em consideração a captura de abstrações que facilitem o entendimento do sistema tanto para seu desenvolvimento, quanto para a reutilização [23]. Os métodos também devem auxiliar na definição de como os componentes devem ser conectados uns aos outros para atender aos requisitos especificados, isto é, auxiliar na construção da arquitetura do software [5].

Uma técnica bastante utilizada para a estruturação de componentes propícios a serem reutilizados é a de análise de domínio[1] [15, 11]. O fundamento básico dessa técnica é a análise da lógica do negócio no qual os componentes se inserem. Sendo assim, os componentes que

[1] do inglês *domain analysis*

contêm as funcionalidades das entidades básicas do domínio são considerados candidatos à reutilização.

As principais particularidades dos processos de DBC podem ser observadas tanto no acréscimo de estágios técnicos ao processo convencional, quanto no enfoque dado a algumas práticas já realizadas. Um processo de desenvolvimento de software baseado em componentes geralmente inclui a definição de estratégias para [13, 5]:

- **Definição explícita da arquitetura do sistema**. A explicitação da arquitetura tem o objetivo principal de enfatizar os aspectos de interação entre os componentes do sistema, com os seus fluxos e restrições;

- **Separação de contextos a partir do modelo de domínios.** Com essa separação, pretende-se classificar os componentes mais propícios à reutilização, de acordo com a lógica do negócio de cada sistema em desenvolvimento;

- **Identificação das interfaces dos componentes.** Um dos objetivos principais do DBC é a construção de sistemas facilmente modificáveis [5]. O baixo acoplamento proporcionado pela definição de interfaces providas e requeridas é um meio de alcançar esse objetivo;

- **Identificação do comportamento interno dos componentes.** A etapa de projeto, comum a todos os processos de desenvolvimento, consiste na modelagem dos serviços oferecidos pelo componente. Por ser uma estrutura altamente encapsulada, deve haver transparência em relação à tecnologia utilizada para a sua implementação interna;

- **Montagem dos componentes do sistema.** Nesta etapa ocorre a materialização da configuração arquitetural do sistema final. Devido à sua autonomia, um componente de software implementa seus serviços utilizando unicamente as suas interfaces requeridas. Sendo assim, a fase de montagem consiste na indicação dos objetos reais que implementam essas interfaces; e

- **Manutenção de um repositório de componentes.** O objetivo principal da utilização de repositórios é maximizar a reutilização de componentes. Isso acontece através do oferecimento de mecanismos de busca sistemáticos que auxiliam o desenvolvimento do software. Normalmente, essas técnicas são utilizadas no início da especificação e antes do projeto interno dos componentes do sistema.

Do ponto de vista dos aspectos do desenvolvimento do software, pode-se dar dois enfoques principais ao desenvolvimento [22]: (i) desenvolvimento para reuso, que enfatiza a produção de componentes propícios a serem reutilizados; e (ii) desenvolvimento com reuso, que consiste na composição de sistemas a partir de componentes já existentes. As Seções 2.1.2 e 2.1.3 detalham cada uma dessas abordagens.

2.1.2 Processos de DBC para Reuso

Esses processos de DBC visam maximizar o grau de reutilização dos componentes desenvolvidos. Por essa razão, durante a especificação interna dos componentes do sistema, deve-se ter em mente a otimização da identificação e da busca dos componentes candidatos a serem reutilizados. Como apresentado na Seção 2.1.1, uma técnica bastante utilizada para a construção de componentes mais propícios para serem reutilizados é a análise do domínio no qual ele se insere. Normalmente, componentes desenvolvidos para uma aplicação específica necessitam ser generalizados para que possam ser reutilizados em um número maior de sistemas.

As principais características desejáveis de um componente reutilizável são [22]:

- Reflete alguma abstração de um domínio estável e bem definido;
- Encapsula seu estado e sua implementação interna;
- Deve reduzir sua dependência de outros componentes;
- Suas interfaces devem ser completas, indicando inclusive as exceções propagadas pelo componente.

Por esse motivo, para que essas características sejam satisfeitas, um processo de DBC voltado para o reuso de componentes deve conter pelo menos as seguintes atividades [22]:

1. Remover os métodos específicos da aplicação;
2. Refatorar os nomes das operações e componentes para torná-lo mais geral;
3. Adicionar novas operações para complementar as necessidades do domínio;
4. Refinar as interfaces do componente para indicar as exceções propagadas;
5. Especificar uma interface para adaptação do componente entre variações do domínio;
6. Integrar componentes requeridos, com o intuito de reduzir as deoendências do componente.

Deve ser feita uma análise da relação custo x benefício, para analisar a viabilidade de se construir o componente com essas características que maximizam o reuso. Essa análise é justificada pelo fato do custo de desenvolvimento de um componente reutilizável ser normalmente maior, quando comparado a componentes de software desenvolvidos sem essa preocupação extra [22]. Porém, esse custo extra do requisito de reusabilidade[2] deve ser considerado um custo organizacional, ao invés de um custo do projeto. Em outras palavras, esse custo deve ser distribuído em todos os projetos que potencialmente vão reutilizar o componente.

[2] do inglês *reusability*

2.1.3 Processos de DBC com reuso

O reuso de componentes é indicado para sistemas de domínio estável, isto é, que são bem definidos e possuem um histórico relativamente pequeno de mudanças. Exemplos de domínios estáveis podem ser: sistemas hospitalares, acadêmicos e alguns sistemas comerciais específicos, tais como sistemas de videolocadoras e de supermercados. Isso se deve ao fato dos componentes que implementam as funcionalidades comuns ao domínio serem o foco principal do reuso. Para sistematizar a reutilização de componentes, um processo de DBC deve ter pelo menos cinco atividades: (i) pesquisar de componentes; (ii) modificar os requisitos de acordo com os serviços providos pelos componentes disponíveis; (iii) buscar novamente componentes com serviços mais adequados aos requisitos alterados; (iv) adaptar as incompatibilidades entre as interfaces providas e requeridas dos componentes reutilizados; e finalmente (v) compor os componentes do sistema.

Referente à atividade de adaptação de componentes, existem três incompatibilidades mais comuns [22]:

1. **Incompatibilidade de parâmetros.** Operações com mesmo nome mas tipos de parâmetros diferentes.

2. **Incompatibilidade de operações.** Operações com a mesma semântica mas com nomes diferentes em interfaces providas e requeridas.

3. **Oferecimento parcial de serviço.** Interface provida de um componente fornece um sub-conjunto de operações da interface requerida por outro componente.

A atividade de composição de componentes para formar o sistema envolve integração entre os componentes e destes com a arquitetura. Sendo assim, se faz necessário construir conectores para integrar os componentes materializados, de modo a constituir o sistema propriamente dito.

2.2 O Processo UML Components

De uma maneira geral, o processo UML Components é um processo de DBC voltado para o desenvolvimento de sistemas de informação. Por esse motivo, ele é um processo simples e de fácil utilização prática. Devido à sua simplicidade, o processo UML Components adota uma arquitetura pré-definida em quatro camadas. Duas dessas camadas são destacadas durante o desenvolvimento: camada de **sistema** e camada de **negócio**. A camada de sistema é responsável por agrupar os componentes que implementam as funcionalidades especificadas para o sistema. Porém, para que essas funcionalidades possam ser implementadas, esses componentes podem necessitar de algumas funcionalidades comuns ao domínio. Os componentes que implementam essas funcionalidades gerais são posicionados na camada de negócio. As quatro camadas da arquitetura e o papel de cada uma podem ser vistos na Figura 1.

Como mostrado na Figura 2, no processo UML Components, o desenvolvimento é dividido em 6 fases: (i) especificação de requisitos (*requirements definition*); (ii) especificação dos componentes (*specification*); (iii) provisionamento dos componentes (*provisioning*); (iv)

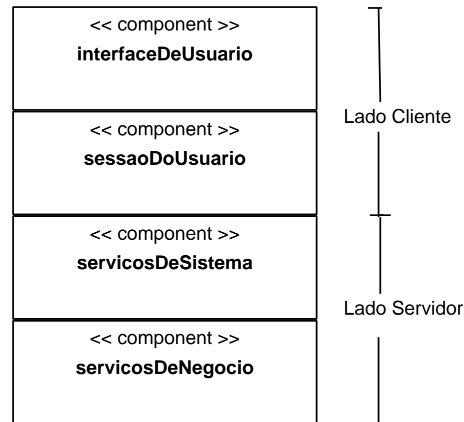

Figura 1: Arquitetura Adotada pelo processo UML Components [5]

montagem do sistema (*assembly*); (v) testes (*test*); e (vi) implantação (*deployment*). Esse processo é iterativo e enfatiza basicamente a fase de especificação dos componentes. Por essa razão essa fase é detalhada em três sub-fases: (i) identificação dos componentes (*component identification*); (ii) interação entre os componentes (*component interaction*); e (iii) especificação final (*component specification*).

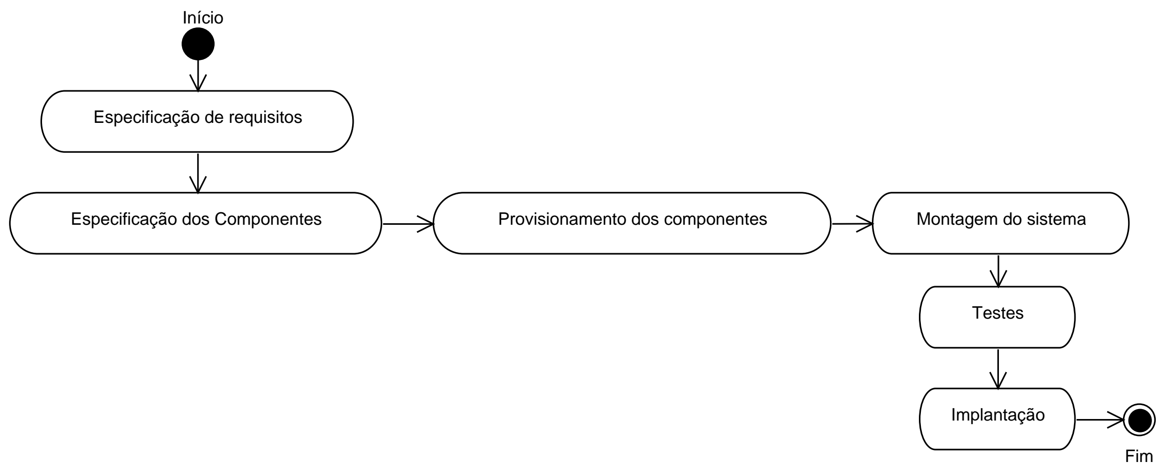

Figura 2: Fases do processo UML Components [5]

2.3 Fases do Processo

A visão geral do processo UML Components foi mostrada no início na Seção 2.2. Agora, cada uma das atividades do processo é explicada brevemente.

2.3.1 Especificação de Requisitos

Apesar deixar bem claro o papel desta fase, inclusive descrevendo os artefatos que devem ser produzidos nela, o processo UML Components não detalha as atividades da sua execução. Na fase de especificação dos requisitos, as funcionalidades do sistema são representadas através de casos de uso. Além disso, é especificado o modelo conceitual do negócio[3], que representa as entidades básicas da lógica do mesmo. Dadas as suas importâncias, esses artefatos são consideradas as principais saídas desta fase.

O modelo conceitual do negócio representa o domínio do problema. Por essa razão ele necessita ser entendido e firmado entre os interessados no sistema[4]. O propósito principal desse modelo é proporcionar um vocabulário comum entre os envolvidos no projeto. Além disso, esse modelo é a base para a identificação dos componentes da camada de negócio.

Os casos de uso, são utilizados como notação para representar os requisitos funcionais do sistema. O modelo inicial de casos de uso pode ser constuído com as principais funcionalidades do sistema. Após o início da especificação, esse modelo pode ser refinado para contemplar os demais requisitos funcionais.

2.3.2 Especificação dos Componentes

A fase de especificação dos componentes é a fase do desenvolvimento que o processo UML Components mais entra em detalhes. Como o próprio nome indica, essa fase é responsável por especificar os componentes do sistema. Essa especificação acontece tomando como entrada os artefatos produzidos na fase de especificação de requisitos: (i) modelo conceitual do negócio; e o (ii) modelo de casos de uso.

Os principais artefatos de saída da fase de especificação dos componentes são três: (i) especificação das interfaces (refinamento das assinaturas); (ii) especificação dos componentes (interfaces providas e requeridas); e (iii) a arquitetura do sistema, como conseqüência da definição das dependências entre os componentes. Como pode ser visto na Figura 3, para produzir esses artefatos, o processo UML Components divide essa fase em três estágios: (i) identificação dos componentes; (ii) interação entre os componentes; e (iii) especificação final. A Seguir, é explicado o papel de cada um dos estágios de especificação.

1. O estágio de **identificação dos componentes** recebe como entrada o modelo conceitual do negócio e a especificação dos casos de uso do sistema, ambos provenientes da fase de especificação de requisitos. O objetivo básico dessa etapa da especificação é identificar um conjunto inicial de interfaces. Além disso, essas interfaces devem ser classificadas em duas categorias: (i) interfaces de sistema, relativas à funcionalidade do sistema; e (ii) interfaces de negócio, que conterão as operações básicas do negócio.

[3]do inglês *business concept model*
[4]do inglês *system stakeholders*

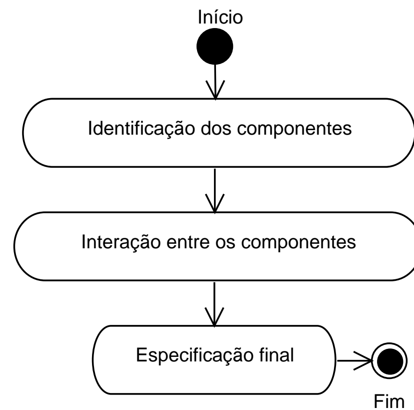

Figura 3: Especificação dos Componentes no processo UML Components [5]

Devido às características já citadas de cada grupo de interfaces, as interfaces de sistema são identificadas a partir dos casos de uso e as interfaces de negócio a partir do modelo conceitual fornecido.

Em seguida, para cada uma dessas interfaces, é criado um componente. Devido à classificação da interface, o componente já pode ser posicionado na arquitetura, nas camadas correspondentes à classificação da interface (sistema ou negócio). Em relação às interfaces de sistema, além da identificação da interface propriamente dita, neste estágio também são identificadas as primeiras operações. Apesar disso, são definidos apenas os seusnomes; a assinatura completa é refinada apenas durante o próximo estágio, de interação entre os componentes.

2. Já durante a interação entre os componentes, a preocupação é voltada para o detalhamento das estruturas identificadas anteriormente. Por esse motivo, nesse estágio o projetista examina como cada operação do sistema será implementada. Em outras palavras, a preocupação é saber quais componentes do sistema deverão interagir para o serviço ser implementado. Para isso, o processo UML Components utiliza diagramas de colaboração UML. Através dessas colaborações, as operações requeridas do negócio são descobertas e as assinaturas das operações de sistema podem ser refinadas.

   Com a definição das dependências entre os componentes do sistema, a estrutura da arquitetura vai sendo definida. No final da execução do estágio de interação entre os componentes, a arquitetura do sistema já está definida. Essa definição possui um nível de detalhes alto, com uma definição clara das dependências entre as operações dos componentes.

3. Finalmente, o estágio de espeficifação final dos componentes é onde as especificações são refinadas e, se possível, representadas de uma maneira formal. Durante a especificação final, a arquitetura do sistema não deve variar. Dessa forma, o detalhamento da

especificação só deve ser feito no momento em que a arquitetura do sistema já está considerada estável. A definição formal de assertivas é uma tarefa custosa, do ponto de vista de esforço de trabalho, e por isso deve-se evitar retrabalho nessa tarefa.

Com a especificação dos componentes finalizada, é chegada a hora de providenciar cada um dos componentes para em seguida compor o sistema, testá-lo e entregá-lo ao cliente para utilização. Apesar da importância dessas fases, o processo UML Components não detalha a sua execução. As sub-seções seguintes descrevem o papel de cada uma dessas fases do desenvolvimento.

2.3.3 Provisionamento dos componentes

A etapa de provisionamento deve garantir a materialização dos componentes especificados. Essa materialização pode ocorrer de duas formas: (i) reutilização de componentes já existentes; e (ii) implementação de componentes novos. Normalmente, os componentes da camada de negócio são candidatos à reutilização, uma vez que representam características básicas para sistemas de um mesmo domínio. Já os componentes da camada de sistema oferecem implementacões específicas, peculiares às necessidades de um sistema em particular. Por esse motivo, apesar de também poderem ser reutilizados, as chances para isso são reduzidas.

Para cada uma das formas de materialização, há diferentes atividades a serem realizadas. Para o caso de reutilização de componentes, devido às possíveis inconsistências entre as interfaces especificadas e reutilizadas, pode ser necessário implementar adaptadores [22]. No caso dos componentes implementados, é necessário especificar as suas estruturas internas. Para isso, deve-se utilizar algum modelo de componentes que possibilite a sua especificação através das estruturas existentes nas linguagens de programação, tais como objetos e classes.

2.3.4 Montagem do sistema

A fase de montagem é responsável pela materialização da configuração da arquitetura do sistema. Sendo assim, ela consiste na integração dos componentes para construção da aplicação como um todo. Basicamente, existem duas atividades desempenhadas durante esta fase: (i) a construção dos conectores, que implementam um código de mapeamento entre os componentes do sistema; e (ii) a construção do programa principal, que deve conter o código de instanciação dos componentes, dos conectores, além da "ligação" entre eles, que é efetuada dinamicamente.

2.3.5 Testes

Teste é a atividade de executar um software com o objetivo de revelar falhas [19]. Apessar de não comprovar a ausência de *bugs*, a sua execução aumenta consideravelmente a confiabilidade do sistema [19]. Porém, para uma maior eficácia, os testes devem ser considerados desde as fases iniciais do desenvolvimento [19, 22]. Apesar da importância dos testes, tanto o processo UML Components quanto o método MDCE+ se restringem ao aspecto puramente de desenvolvimento, não contendo atividades sistemáticas de testes que tratem essa

questão desde a especificação dos requisitos. Em parte isso é uma vantagem, uma vez que possibilita a utilização conjunta com algum processo de testes distintos.

De uma forma geral, o papel de um processo de testes é realizar verificações entre o sistema e os requisitos especificados. Dessa forma, os testes tentam encontrar respostas para as seguintes perguntas: (i) os requisitos estão corretos? (ii) os requisitos estão consistentes (não-contraditórios entre si)? (iii) o sistema atende minimanente a todos os requisitos? Como pode ser percebido, as respostas a essas questões são de fundamental importância para a garantia da qualidade do sistema.

2.3.6 Implantação

Após o seu desenvolvimento e uma garantia maior da satisfação dos seus requisitos, o sistema pode ser implantado para sua utilização. A conclusão da fase de implantação não significa que o ciclo de desenvolvimento do sistema foi finalizado. Pelo contrário, muito provavelmente serão necessários alguns ciclos de manutenções corretivas para que o sistema possa ser considerado estável [19]. Além disso, mesmo com a estabilidade do sistema, nada impede que novas mudanças de requisitos sejam solicitadas por parte do cliente. As atividades de evolução da especificação do sistema, decorrentes dessas mudanças de requisitos, são conhecidas como manutenções perfectivas (ou evolutivas) [22, 19].

Nesse momento, a descrição do processo UML Components está finalizada. As seções seguintes apresentam cada uma das fases do processo adaptado, resultante da união entre o processo UML Components e o método MDCE+.

3 O Processo Proposto

Como visto na Seção 2.1.3, o principal objetivo dos processos de desenvolvimento com reuso é reduzir os esforços de implementação. Essa redução é alcançada pela maximização do número de componentes reutilizados. O processo descrito nessa seção é um processo genérico, no sentido de ser constituído de atividades gerais, comuns à maioria das metodologias de DBC atuais [5, 9]. Com isso, pretende-se que ele seja mais facilmente adaptavel a vários processo específicos adotados por diferentes organizações. Essa facilidade de utilização é decorrente da não imposição de um processo específico. Mudanças radicais costumam afetar a estruturação do desenvolvimento e, por isso, essa ação é considerada arriscada [22, 19].

Por se tratar de um processo de desenvolvimento de software, o objetivo do processo proposto é desenvolver um sistema a partir de descrições textuais do que se espera dele. Dessa forma, a partir de refinamentos sucessivos, são gerados artefatos cada vez mais específicos, a ponto de no final se ter os componentes do sistema especificados e implementados em uma linguagem de programação. Além disso, o foco no reuso obriga o processo a adicionar atividades que antecipem e maximizem o número de componentes reutilizados.

Como mostrado na Figura 4, o processo de desenvolvimento apresentado aqui é composto de oito passos básicos: (i) especificação de requisitos; (ii) projeto arquitetural; (iii) reutilização de componentes prontos; (iv) implementação de componentes novos; (v) imple-

mentação dos conectores e adaptadores; (vi) integração dos componentes do sistema; (vii) validação do sistema; e (viii) correções do sistema.

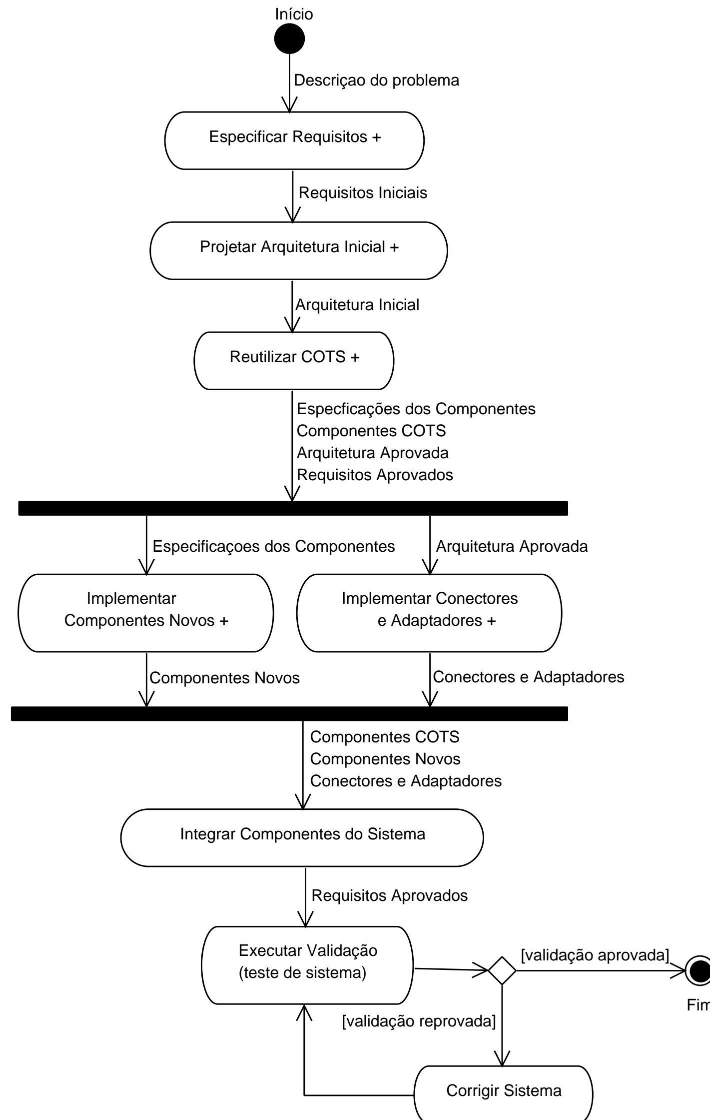

Figura 4: Visão Geral do Processo

A partir das descrições textuais iniciais, os requisitos do sistema são especificados. Em seguida, a visão funcional abstrata já começa a dar lugar a uma visão mais específica de implementação, através da estruturação inicial da arquitetura de software. Essa especificação precoce da arquitetura deve modularizar a estrutura do sistema através da identificação dos seus principais componentes abstratos. Com os componentes arquiteturais identificados, o processo passa por uma fase de busca por candidatos a desempenhar o papel de cada um

deles. Vale a pena salientar o fato de que o principal fator a ser considerado na escolha dos componentes adequados é o equilíbrio entre a qualidade das funcionalidades implementadas (proximidade dos requisitos) e o custo de implementação do sistema.

Após esse etapa, os componentes do sistema já estarão especificados. Dessa forma, aqueles que não foram reutilizados podem dar seqüência ao processo de desenvolvimento. Dependendo da granularidade do componente, pode ser necessário executar o processo recursivamente, no intuito de reutilizar as partes que o constituem. O fato do processo ser recursivo o adequa tanto ao desenvolvimento de sistemas de grande porte, quanto ao desenvolvimento de sistemas pequenos.

As seções seguintes detalham cada uma das oito atividades do processo, que foram apresentadas na Figura 4.

3.1 Especificação de Requisitos

A especificação de requisitos é a primeira fase a ser executada durante o desenvolvimento de um software. Em geral, os requisitos podem ser vistos como os objetivos esperados para o sistema, assim como as condições e capacidades necessárias para tal. Por se tratar de um processo voltado para o reuso de componentes, já nessa fase devem ser construídos protótipos a partir de componentes prontos. A construção de protótipos tem o objetivo de melhorar a compreensão dos requisitos e ao mesmo tempo auxiliar o entendimento da sua arquitetura [1].

Como apresentado na Seção 2.1, de acordo com as suas características, um requisito pode ser classificado como **funcional** ou **não-funcional** [19]. Os requisitos funcionais podem ser mapeados para os serviços, tarefas ou funções que o sistema deve oferecer. Já os requisitos não-funcionais, apesar de não representarem funcionalidades diretamente, eles podem interferir na maneira como o sistema deve executá-las. Dessa forma, a organização estrutural do sistema, materializada na arquitetura de software, é diretamente relacionada a esses requisitos.

Conforme mostrado na Figura 5, para a especificação das funcionalidades e requisitos de qualidade do sistema, a fase de especificação de requisitos abrange basicamente sete atividades: (i) análise e representação do domínio do problema; (ii) identificação inicial dos requisitos funcionais; (iii) construção do protótipo; (iv) refinamento dos requisitos funcionais; (v) especificação dos requisitos de qualidade; (vi) especificação das restrições de desenvolvimento; e (vii) definiçã das restrições de reutilização.

De um modo particular, as especificações de restrições (de desenvolvimento e reutilização) devem ter um cuidado especial. Esse cuidado é decorrente do fato do desenvolvimento de um software sempre representar um risco [19, 4]. As principais restrições de desenvolvimento a serem definidas estão representadas na Figura 6. Além disso, do ponto de vista do reuso de componentes, essas restrições serão úteis para descartar alguns dos possíveis candidatos. Alguns exemplos comuns desse tipo de restrição são [22]: (i) existência de certificações específicas; (ii) custo do produto e forma de pagamento; (iii) Prazo de entrega; e (iv) garantia/manutenção do componente.

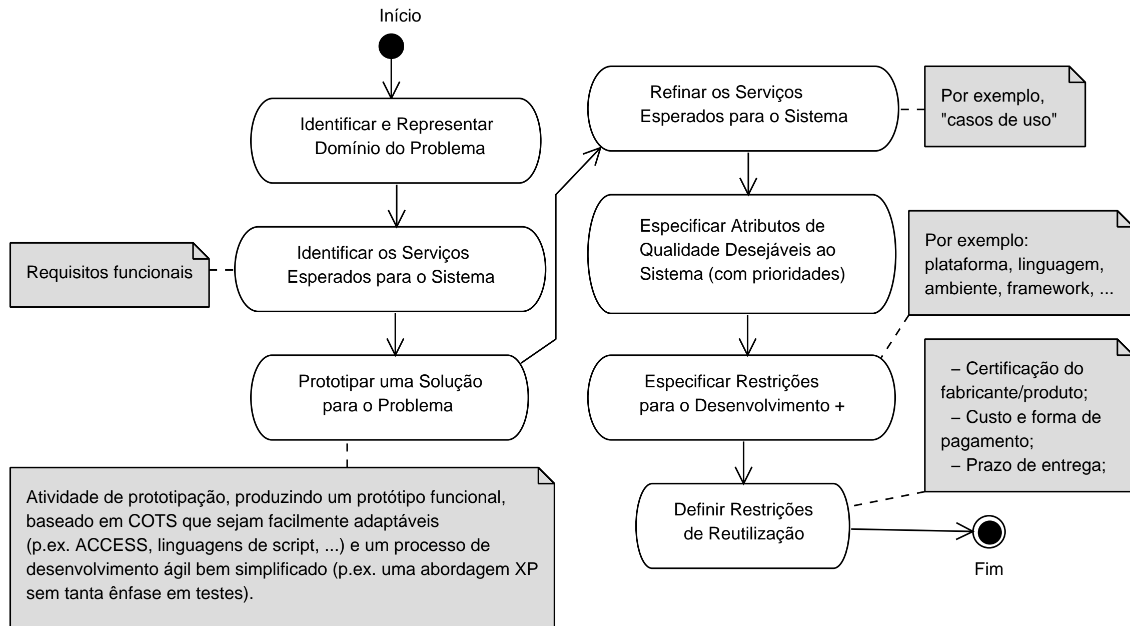

Figura 5: Especificação dos Requisitos

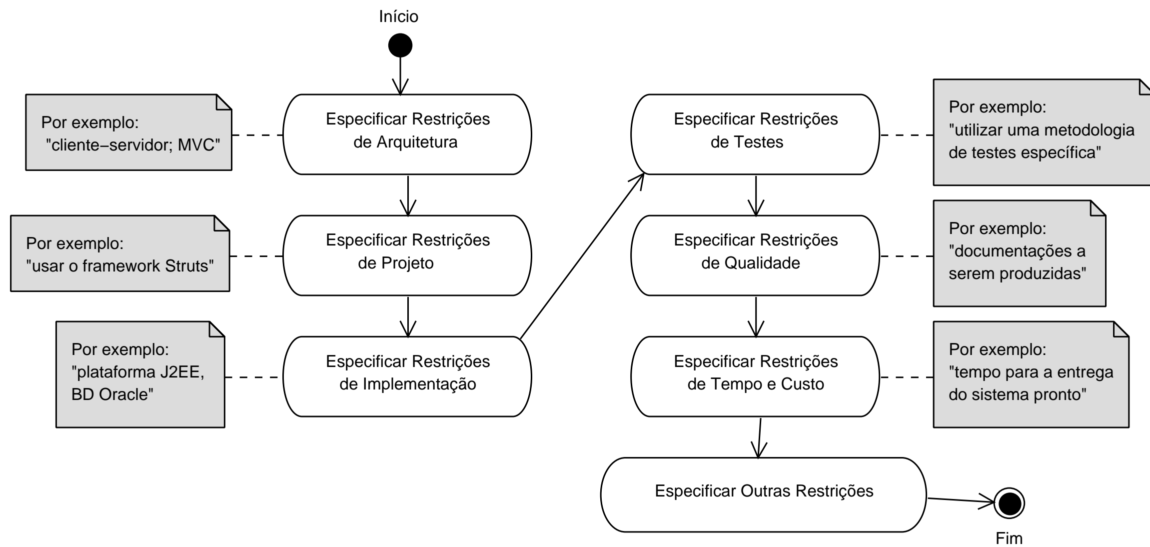

Figura 6: Especificação de Restrições de Desenvolvimento

3.2 Projeto Arquitetural

Com o aumento crescente do tamanho e principalmente da complexidade dos sistemas de software, o projeto arquitetural assumiu um papel decisivo para o sucesso ou falha no

atendimento dos requisitos, principalmente dos requisitos não-funcionais [22, 6].

As principais vantagens de se enfatizar o papel da arquitetura do sistema são: (i) ter uma estruturação abstrata, que facilita o entendimento e faz da arquitetura um eficiente veículo de comunicação entre as partes interessadas[5] do sistema; (ii) ter uma granularidade alta de reutilização através dos componentes arquiteturais, que aliado ao DBC pode ser determinante para a redução do tempo de desenvolvimento, o que atende a um requisito importante do mercado desenvolvedor de software atual [20]; (iii) ter um mecanismo para lidar com a estruturação de sistemas de softwae complexos; e (iv) ter uma facilidade maior para adaptar, manter, evoluir e portar o sistema para outras plataformas, como conseqüências do baixo acoplamento proporcionado pelo uso de conectores arquiteturais, que explicitam a comunicação entre os componentes da arquitetura. Essa comunicação explícita facilita a substituição dos componentes, uma vez que torna possível a adaptação das mensagens que fluem entre eles [7, 25].

Durante a escolha do estilo arquitetural a ser adotado, o arquiteto de software pode inclusive reutilizar arquiteturas de sistemas anteriores, baseado nas semelhanças entre as suas restrições e os seus requisitos de qualidade. Seguindo essa tendência, pesquisadores estudam formas de sistematizar essa reutilização estrutural do sistema como forma de agilizar o desenvolvimento do software. Um exemplo muito em voga atualmente é a abordagem de desenvolvimento em linhas de produção de software[6] [18, 10]. Devido à importância da arquitetura em todo o desenvolvimento do sistema, a Figura 7 detalha o *workflow* de escolha do estilo arquitetural e de descoberta dos componentes iniciais.

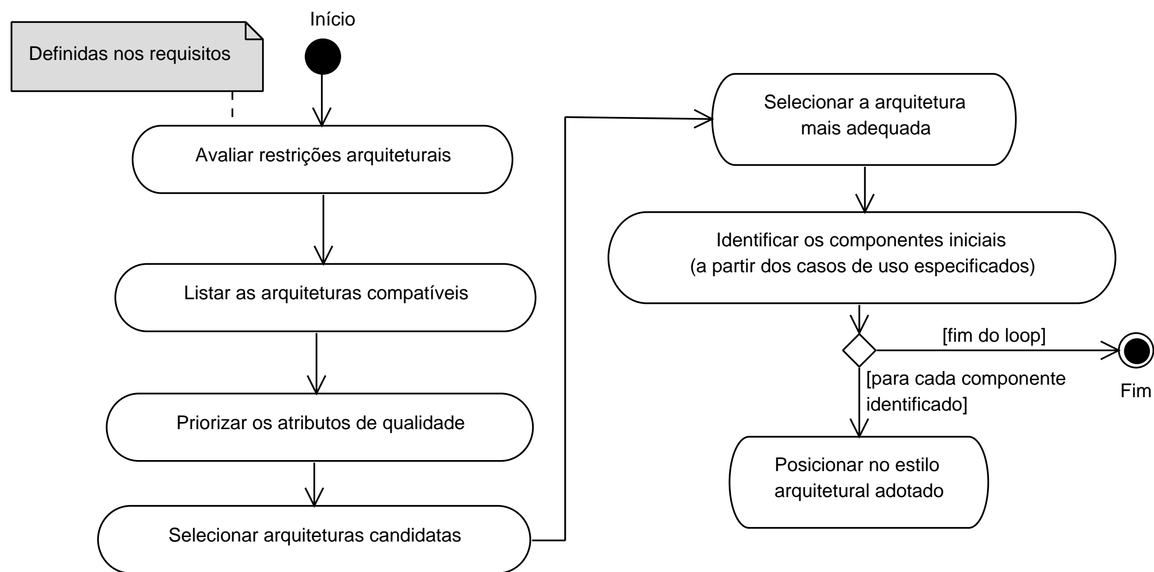

Figura 7: Escolha da Arquitetura do Sistema

[5] do inglês *stakeholders*
[6] do inglês *software product lines*

3.3 Reutilização de Componentes Prontos

O método proposto define três formas de se materializar um componente: (i) reutilização de um componente já utilizado pela organização; (ii) aquisição do componente a partir de um catálogo de terceiros; e (iii) implementação do componente. Para maximizar a redução do tempo e do custo de desenvolvimento, é necessário priorizar a reutilização de componentes prontos. O *workflow* de execução dessa atividade de busca é apresentado na Figura 8. Nele, após a identificação dos componentes candidatos, é iniciado um processo de negociação e ajuste dos requisitos. Dessa forma, espera-se conciliar todas as vantagens de se reutilizar um componente e a satisfação máxima dos requisitos especificados. Devido à importâcia do processo de identificação dos candidatos iniciais, a Figura 9 expande essa atividade.

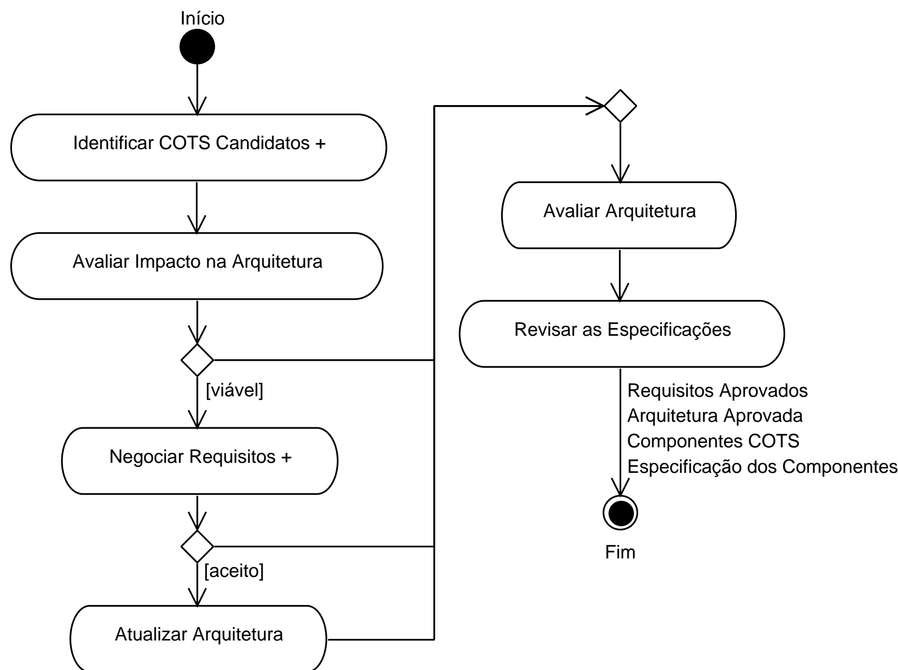

Figura 8: Reutilização de Componentes Prontos

Em termos de estratégia de reuso, antes de mais nada é necessário analisar os componentes existentes em repositórios internos da própria organização, à procura de componentes já utilizados em outros projetos. Por representarem características comuns de vários sistemas, os componentes associados ao domínio do negócio são os mais propícios para serem reutilizados.

Após a identificação dos componentes que já foram utilizados no domínio da organização, a busca pode ser estendida para catálogos externos de componentes. Esses catálogos são constituídos de componentes de softwares que podem ser adquiridos de terceiros, quer seja através de uma compra, quer seja através da adoção de componentes do tipo software livre (*open source* ou *freeware*). Nesses casos, a busca de componentes nem sempre pode levar

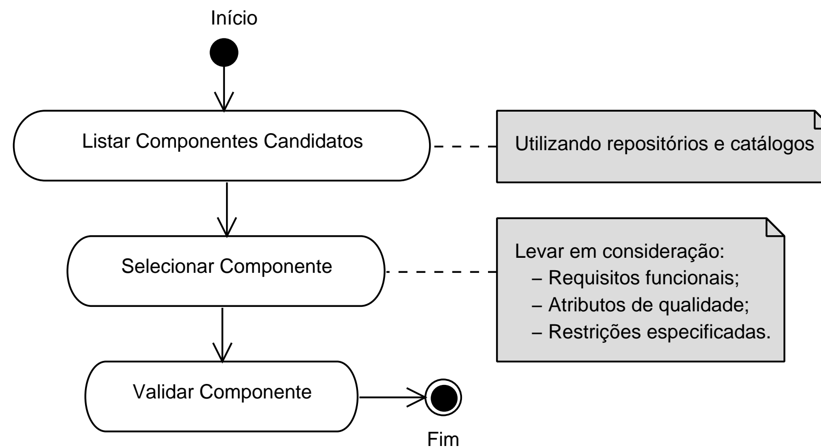

Figura 9: Identificar Componentes Candidatos

Modo de Materialização	Custo
Reutilização Interna	Custo da integração de componentes (se existe mais de um componente) + custo da adaptação + custo da materialização dos outros serviços (se não foram materializados todos)
Aquisição Externa	O mesmo da reutilização + o custo da aquisição
Implementação	Estimativa do custo da implementação dos componentes. Podem ser utilizadas técnicas tradicionais, tais como pontos por função e estimativas baseadas em casos casos de uso

Tabela 1: Custos das Formas de Materialização

em consideração as entidades do domínio, uma vez que além das particularidades próprias de cada organização, não se pode esperar uma padronização na representação dos modelos.

Com a lista dos possíveis candidatos a serem reutilizados, deve-se proceder a escolha do modo como o componente será materializado. Essa escolha consiste basicamente numa análise do custo x benefício entre as três formas existentes: reutlizar, adquirir ou implementar. A análise de custos deve abranger limitações de tempo e recursos [22]. Na reutilização de componentes prontos, os componentes reutilizados nem sempre obedecem aos mesmos contratos especificados nos requisitos. Por esse motivo, nesses casos pode ser necessário adaptar o componente reutilizado a fim de satisfazer os contratos estabelecidos [7, 22]. Os custos de construção desses adaptadores também devem ser contabilizados no cálculo do custo total do componente. A Tabela 3.3 mostra as principais variáveis que devem ser levadas em consideração para a estimativa do custo. Outras vantagens, tais como a disponibilidade de código e o oferecimento de serviços extra, tais como garantia e manutenção também devem ser levados em consideração.

3.4 Implementação de Componentes Novos

Na ausência de componentes prontos que motivem a reutilização, os componentes devem ser implementados de acordo com a especificação produzida no decorrer do desenvolvimento. Devido à característica recursiva desse processo de desenvolvimento, para a implementação de um componente com granularidade alta, a especificação pode ser executada recursivamente, a partir da fase de especificação dos requisitos (Seção 3.1). Dessa forma, apesar de não ter sido possível a reutilização do componente como um todo, ainda existe a chance de reutilizar suas partes internas, dado que ele é um componente complexo.

Como pode ser visto no *workflow* de execução da etapa de implementação (Figura 10), além da possibilidade de aplicar o processo recursivamente, o desenvolvedor pode optar por desenvolver o componente "do zero". Essa opção é válida para os casos onde a granularidade do componente já atingiu o limite da relação custo x benefício. Em outras palavras, o custo de tentar reutilizar partes do componente pode ser superior à sua implementação.

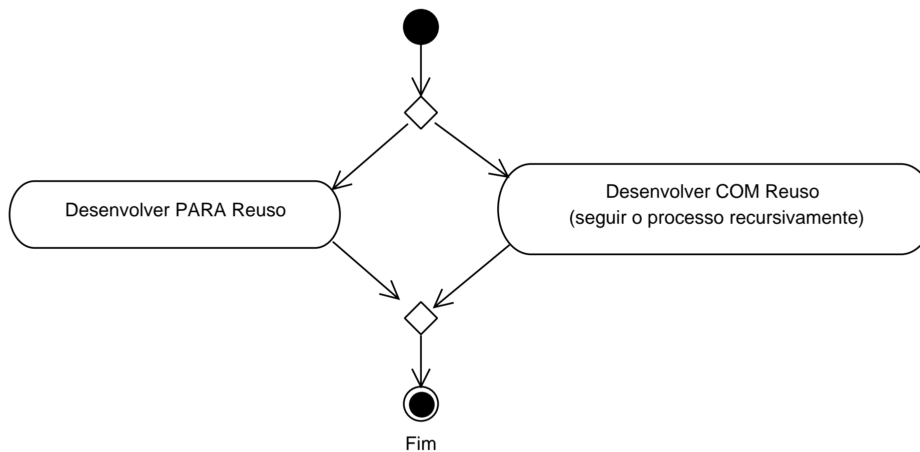

Figura 10: Implementação dos Componentes Novos

Devido à inexistência de linguagens de programação que utilizem o paradigma do desenvolvimento baseado em componentes, o primeiro passo para a criação de um novo componente é escolher um modelo que possibilite o mapeamento das suas abstrações para as estruturas das linguagens de programação atuais. Alguns exemplos de modelos de componentes existentes na literatura são: EJB [17], DCOM [21], CCM [16] e COSMOS [14].

Após a escolha do modelo, deve-se adaptar a especificação de acordo com as suas restrições. Um exemplo comum de uma restrição a ser verificada é a impossibilidade de um componente oferecer mais de uma interface. Conhecendo-se as limitações do modelo adotado, o projetista deve procurar formas de contorná-las. No exemplo da inviabilidade de se oferecer mais de uma interface, poderia ser criada uma nova interface a ser oferecida pelo componente. Em seguida, essa interface deveria herdar de todas as interfaces providas especificadas. Finalmente, após todas essas etapas prévias, o projetista pode proceder a especificação interna do componente. O nosso processo propõe que essa especificação seja feita

utilizando alguma metodologia orientada a objetos, como por exemplo o *Rational Unified Process* (RUP) [13], da IBM.

3.5 Integração dos Componentes do Sistema

Apesar de neste ponto do desenvolvimento já se ter todos os componentes do sistema, ainda nos falta materializar as conexões entre eles. Essas conexões são realizadas através da ligação das interfaces requeridas de um componente às respectivas interfaces providas de outros. Porém, como mostrado na Figura 11, dependendo da similaridade ou não entre as duas interfaces, pode ser necessário implementar algum procedimento de adaptação entre elas. Nos casos onde a adaptação é necessária, o conector passa a ser chamado de adaptador [22], já que é o responsável por esse processamento.

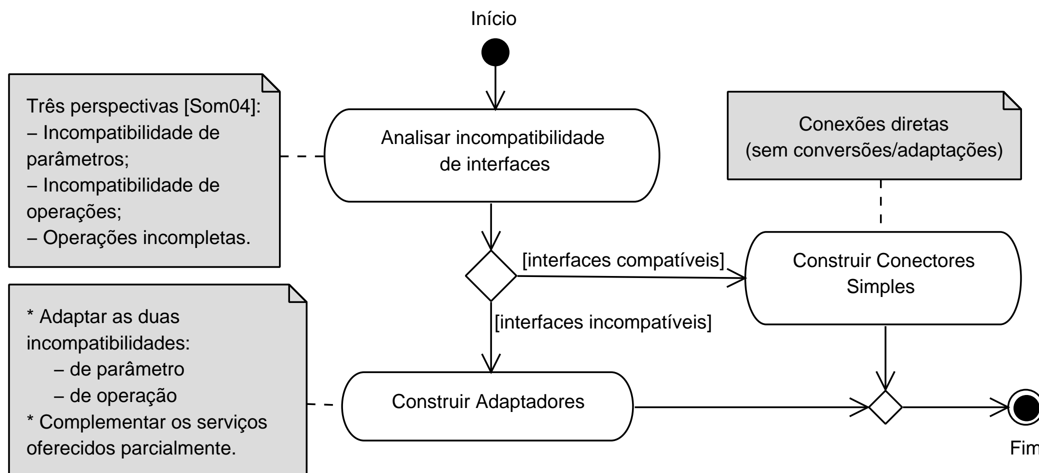

Figura 11: Composição dos Componentes

Por representar o elo de comunicação entre componentes arquiteturais, os conectores são os únicos componentes do sistema que possuem o conhecimento dos seus fluxos interativos. Dessa forma, como discutido no decorrer desse relatório, eles são considerados os locais ideais para a implementação dos requisitos não-funcionais do sistema (Seção 2.1), tais como tolerância a falhas, distribuição e disponibilidade[7].

Após a finalização da especificação dos conectores do sistema, deve-se prosseguir com a sua implementação, conforme o modelo de componentes adotado. Além disso, é necessário implementar as rotinas de "ligação" dos componentes. Esse código é responsável por instanciar cada um dos pares de componentes cliente e servidor, assim como o seu respectivo conector. Em seguida, as interfaces requeridas do componente cliente devem ser inicializadas com a implementação da interface provida do conector. Finalmente, a interface requerida do conector deve ser inicializada com a classe que implementa a interface provida

[7] do inglês *availability*

do componente servidor. Para ficar mais claro, a Figura 12 mostra a estruturação de dois componentes (`A` e `B`), unidos por um conector `AB`.

Esse código de ligação dinâmica dos componentes faz parte da implementação do programa principal do sistema.

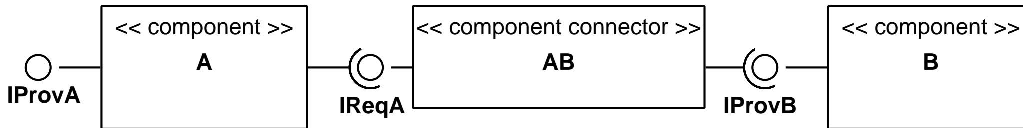

Figura 12: Um Exemplo de Configuração Arquitetural

3.6 Validação & Correções do Sistema

A atividade de validação tem o objetivo de assegurar que a implementação do produto final corresponde aos requisitos especificados [19]. Apesar de não ser garantia de sucesso, essa verificação final do sistema é essencial para o aumento da sua qualidade e da garantia de satisfação do cliente [22, 19].

A validação do software consiste em uma seqüência de testes, elaborados a partir dos documentos de requisitos. Em sistemas reais, devido ao grande número de serviços que podem ser oferecidos, essa atividade deve ser executada preferencialmente de forma automática, com o auxílio de ferramentas CASE.

Resultados negativos nos testes de validação do sistema representam a insatisfação de algum requisito. Por essa razão, a não aprovação dessas atividades implicam necessariamente que o sistema deve ser revisto, a fim de serem feitas as devidas correções. Dessa forma, o processo de desenvolvimento não pode ser finalizado antes que o produto final esteja validado. Em outras palavras, sua reprovação nos testes de validação, indica que o sistema não se encontra nas condições mínimas de ser entregue ao cliente.

4 Adaptação do Processo Proposto

O procedimento de adaptação do processo apresentado na Seção 3 consiste na substituição de algumas atividades desse processo, por atividades de outros processos de desenvolvimento. Esse refinamento, além de tornar o processo instanciável, o torna facilmente utilizado por quem já utiliza o processo cujas atividades foram introduzidas. A seguir, a Seção 4.1 apresenta a adaptação feita ao processo UML Components.

4.1 Adaptação ao Processo UML Components

Como mostrado na Seção 2.2, o processo UML Components é constituído basicamente de seis fases: (i) especificação de requisitos; (ii) identificação dos componentes; (iii) interação

entre os componentes; (iv) especificação final dos componentes; (v) provisionamento dos componentes; e (vi) montagem do sistema.

Dessa forma, para adaptar o processo apresentado na Seção 3 ao UML Components, é necessário acomodar cada uma das fases do segundo nas grandes fases do primeiro. Essa adaptação facilita a adoção da abordagem de maximização do reuso em empresas que já adotam o UML Components como processo de desenvolvimento. Utilizando um raciocínio semelhante, seria possível adaptá-lo a outros processos de desenvolvimento existentes. A seguir, é mostrado para cada fase do UML-Components, o local onde ela se encaixa nas atividades do outro processo.

1. **Especificação de requisitos** ⇒ Especificação de Requisitos (Seção 3.1).
2. **Identificação dos componentes** ⇒ Identificar os componentes iniciais, que faz parte do Projeto Arquitetural (Seção 3.2 - Figura 7).
3. **Interação entre os componentes** ⇒ Revisar as Especificações, que faz parte da Reutilização de COTS (Seção 3.3 - Figura 8).
4. **Especificação final dos componentes** ⇒ A mesma atividade da fase anterior.
5. **Provisionamento dos componentes** ⇒ Desenvolver PARA Reuso, que faz parte da Implementação dos Componentes Novos (Seção 3.4 - Figura 10).
6. **Montagem do sistema** ⇒ É distribuída em duas atividades: Implementação dos Conectores e Adaptadores e Integração dos Componentes do Sistema, ambas apresentadas na Seção 3.5.

5 Conclusões e Trabalhos Futuros

Este relatório descreveu um processo de desenvolvimento baseado em componentes que visa maximizar a reutilização de componentes prontos durante o desenvolvimento. Uma característica importante do processo é o fato dele ser genérico, no sentido de ser constituído de atividades gerais, comuns à maioria das metodologias de DBC atuais [5, 9]. Com isso, pretende-se que ele seja facilmente adaptável aos vários processos utilizados, o que facilita sua utilização prática.

Devido à ênfase na reutilização de componentes prontos, esse processo valoriza a análise do domínio, através da identificação de componentes de acordo com as suas entidades especificadas. Essa identificação acontece de uma maneira iterativa e sistemática, distribuída em todas as fases do desenvolvimento. A seleção dos componentes reutilizados é baseada em um processo de triagem, a partir da relação de compromisso entre os requisitos do sistema e os requisitos oferecidos pelos componentes reutilizados.

Uma outra característica importante do processo é a ênfase dada ao projeto arquitetural do sistema, acompanhada da reutilização dessas abstrações. Através dessa ênfase na arquitetura, o processo visa proporcionar reutilização de componentes de maior granularidade.

Após a identificação dos componentes a serem reutilizados e a implementação dos demais componentes, deve-se proceder a materialização dos conectores arquiteturais, cujo papel é

compor o sistema através da integração dos seus componentes. Por representar o elo de comunicação entre os componentes arquiteturais, os conectores são considerados os locais ideais para a implementação dos requisitos não-funcionais do sistema, tais como tolerância a falhas, distribuição e disponibilidade[8]. Além disso, ao compor componentes reusáveis, pode ser necessário especificar ações de adaptação entre as inconsistências das diferentes interfaces dos componentes. Quando implementa essas adaptações, o conector recebe o *status* de adaptador.

Com a o processo definido, o próximo passo é avaliá-lo através do desenvolvimento de um estudo de caso em uma empresa que adota algum processo de DBC. Idealmente, essa empresa deve possuir um domínio estável e bem definido, além de um repositório dos seus componentes já desenvolvidos anteriormente. Um outro passo é adaptar o processo proposto a outras metodologias de DBC já existentes.

6 Agradecimentos

O desenvolvimento desse trabalho recebu apoio do projeto MCT/FINEP/Ação Transversal - Biblioteca de Componentes - 05/2004. Patrick Brito é também apoiado pelo curso de Especialização em Eng. de Software do Instituto de Computação – Unicamp. Maria Antonia Barbosa é também apoiada pelo CNPq, processo no. 381805/2005-0. Cecília Rubira é apoiada parcialmente pelo CNPq, processo no. 351592/97-0.

Referências

[1] Carina Alves, João Bosco Pinto Filho, and Jaelson Castro. Analysing the tradeoffs among requirements, architectures and cots components. In *WER*, pages 20–31, 2001.

[2] T. Anderson and P. A. Lee. *Fault Tolerance: Principles and Practice*. Springer-Verlag, 2nd edition, 1990.

[3] Algirdas Avizienis. Towards systematic design of fault-tolerant systems. *IEEE Computer*, 30(4):51–58, April 1997.

[4] Barry W. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.

[5] John Chessman and John Daniels. *UML Components*. Addison-Wesley, 2000.

[6] Paul Clements and Rick Kazman. *Software Architecture in Practices*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[7] Paulo Asterio de C. Guerra, Fernando Castor Filho, Vinicius Asta Pagano, and Cecília M. F. Rubira. Structuring exception handling for dependable component-based software systems. In *EUROMICRO*, pages 575–582, 2004.

[8] do ingês *availability*

[8] Paulo Asterio de Castro Guerra. *Uma Abordagem Arquitetural para Tolerância a Falhas em Sistemas de Software Baseados em Componentes.* PhD thesis, IC, Unicamp, June 2004.

[9] Desmond D'Souza and Alam Cameron Wills. *Objects, Components, and Frameworks with UML The Catalysis Approach.* Addison-Wesley, 2nd edition, 1999.

[10] L. Feijs. Architecture visualisation and analysis: Motivation and example. In *Intl. Workshop on Development and Evolution of Software Architectures for Product Families*, 1996.

[11] Robert B. France and Thomas B. Horton. Applying domain analysis and modeling: an industrial experience. In *SSR '95: Proceedings of the 1995 Symposium on Software reusability*, pages 206–214, New York, NY, USA, 1995. ACM Press.

[12] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlisside. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading, Massachusetts, 1995.

[13] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The unified software development process.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[14] Moacir C. Silva Jr., Paulo A. C. Guerra, and Cecília Rubira. A java model for evolving software systems. *IEEE International Conference on Automated Software Engineering(ASE'03)*, October 2003.

[15] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical report CMU/SEI-90-TR-021, Pittsburgh, PA, USA, 1990.

[16] Raphael Marvie and Philippe Merle. Vers un modèle de composants pour CESURE - le CORBA Component Model. Technical Report 3, Projet RNRT 98 CESURE, Novembre 2000. http://www.gemplus.fr/cesure/.

[17] Richard Monson-Haefel. *Enterprise JavaBeans.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1999.

[18] Robert L. Nord, editor. *Software Product Lines, Third International Conference, SPLC 2004, Boston, MA, USA, August 30-September 2, 2004, Proceedings*, volume 3154 of *Lecture Notes in Computer Science*. Springer, 2004.

[19] Roger S. Pressman. *Software Engineering: a Practitioner's Approach.* McGraw-Hill, 5th edition, 2001.

[20] Brian Randell. Dependable pervasive systems. *23rd IEEE International Symposium on Reliable Distributed Systems (SRDS'04)*, October 2004.

[21] Roger Sessions. *COM and DCOM: Microsoft's vision for distributed objects.* John Wiley & Sons, Inc., New York, NY, USA, 1998.

[22] Ian Sommerville. *Software Engineering*. Addison-Wesley, 6th edition, 2001.

[23] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1998.

[24] Clemens Szyperski. Component software and the way ahead. In Gary T. Leavens and Murali Sitaraman, editors, *Foundations of Component-Based Systems*, chapter 1, pages 1–20. Cambridge University Press, 2000.

[25] Zhang You-Sheng and He Yu-Yun. Architecture-based software process model. *Software Engineering Notes*, 28(2), March 2003.