

UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO
DESENVOLVIMENTO DE SISTEMAS PARA WEB

Desenvolvimento do Componente *WorkflowArchitectureMgr* da
Linha de Produto para Sistemas de Gerenciamento de *Workflow*

FÁBIO GABRIEL ZAUPA

Maringá - PR
Novembro / 2004

FÁBIO GABRIEL ZAUPA

Desenvolvimento do Componente *WorkflowArchitectureMgr* da
Linha de Produto para Sistemas de Gerenciamento de *Workflow*

Monografia apresentada como requisito parcial à obtenção do grau de Especialista pelo curso de Pós-Graduação em Desenvolvimento de Sistemas para Web, da Universidade Estadual de Maringá.

Orientadora: Profa. Dra. Itana Maria de Souza Gimenes

Co-Orientador: Edson A. Oliveira Junior

Maringá - PR
Novembro / 2004

TERMO DE APROVAÇÃO

FÁBIO GABRIEL ZAUPA

Desenvolvimento do Componente *WorkflowArchitectureMgr* da
Linha de Produto para Sistemas de Gerenciamento de *Workflow*

Monografia aprovada como requisito parcial para obtenção do grau de Especialista pelo curso de Pós-Graduação em Desenvolvimento de Sistemas para Web, Departamento de Informática, Universidade Estadual de Maringá, pela seguinte banca examinadora:

Profa. Dra. Itana Maria de Souza Gimenes
Departamento de Informática, UEM.

Profa. Dra. Elisa Hatsue Moriya Huzita
Departamento de Informática, UEM.

Profa. Dr. Marcelo Morandini
Departamento de Informática, UEM.

Maringá - PR
Novembro / 2004

AGRADECIMENTOS

Agradeço primeiro a Deus.

Agradeço a toda minha família e a minha namorada Andrea pela paciência e apoio nesse período.

Agradeço aos meus amigos César, João Carlos, Nelson e Ana Carolina, que por um período de um ano foram meus companheiros de viagem e verdadeiros amigos.

Agradeço a todos os professores do curso de especialização, que usaram um pouco do tempo de seus sábados para passar para nós um pouco de seus conhecimentos.

Agradeço aos integrantes do projeto ExpPSEE da UEM, principalmente ao Fabrício Lazilha pelas dicas importantes e ao Edson Alves de Oliveira Junior, que foi uma pessoa fundamental para a realização desse projeto, passando muitos conhecimentos e demonstrando ser uma pessoa altamente qualificada e de grande futuro.

E por fim, agradeço a minha orientadora, professora Dra. Itana Gimenes, pela excelente orientação, por ter acreditado no meu potencial e por ter me dado a chance de participar desse ótimo projeto ExpPSEE.

SUMÁRIO

LISTA DE ABREVIATURAS	5
LISTA DE FIGURAS.....	6
RESUMO	7
ABSTRACT	8
CAPÍTULO 1 - INTRODUÇÃO	9
CAPÍTULO 2 – FUNDAMENTAÇÃO TEÓRICA	11
2.1. CONCEITOS BÁSICOS	11
2.2. ELEMENTOS DE UMA LINHA DE PRODUTO DE SOFTWARE.....	12
2.2.1 <i>Variabilidade</i>	14
2.2.2. <i>Desenvolvimento do Núcleo de Artefatos</i>	18
2.2.2.1. Contexto da Linha de Produto	19
2.2.2.2. Núcleo de artefatos.....	19
2.2.2.3. Plano de Produção	19
2.2.3. <i>Desenvolvimento do Produto</i>	20
2.2.4. <i>Gerenciamento de Linha de Produto</i>	21
2.3. DESENVOLVIMENTO BASEADO EM COMPONENTES	22
2.4. O PROCESSO UML COMPONENTS	23
2.5. CONSIDERAÇÕES FINAIS	28
CAPÍTULO 3 - A TECNOLOGIA DE <i>WORKFLOW</i>.....	29
3.1. DEFINIÇÃO DE <i>WORKFLOW</i>	29
3.2. CONCEITOS E TERMOS RELACIONADOS A <i>WORKFLOW</i>	29
3.3. SISTEMAS DE GERENCIAMENTO DE <i>WORKFLOW</i> (<i>WFMS</i>).....	31
3.3.1. ARQUITETURA GENÉRICA PARA <i>WFMS</i>	31
3.3.2. MODELO DE REFERÊNCIA	33
3.4. A ARQUITETURA DE LINHA DE PRODUTO PARA <i>WFMS</i>	35
3.5. CONSIDERAÇÕES FINAIS	37
CAPÍTULO 4 – O COMPONENTE <i>WORKFLOWARCHITECTUREMGR</i>.....	38
4.1. PROJETO DO COMPONENTE <i>WORKFLOWARCHITECTUREMGR</i>	38
4.2. ESPECIFICAÇÃO DO COMPONENTE.....	40
4.2.1. <i>Workflow de Requisito (Requirements Workflow)</i>	41
4.2.2. <i>Workflow de Especificação (Specification Workflow)</i>	43
4.2.2.1. Component Identification	44
4.2.2.2. Interação dos Componentes (Component Interaction).....	47
4.2.2.3. Especificação dos Componentes (Component Specification)	50
4.3. GERAÇÃO DO CÓDIGO.....	51
4.4. AVALIAÇÃO DO COMPONENTE <i>WORKFLOWARCHITECTUREMGR</i>	52
4.4.1. <i>Análise dos Resultados</i>	54
4.5 CONSIDERAÇÕES FINAIS	54
CAPÍTULO 5 – CONCLUSÕES E TRABALHOS FUTUROS	56
REFERÊNCIAS BIBLIOGRÁFICAS.....	57

LISTA DE ABREVIATURAS

DBC	Desenvolvimento Baseado em Componentes
ExPSEE	<i>Experimental Process-centered Software Engineering Environment</i>
LP	Linha de Produto
RUP	<i>Rational Unified Process</i>
UML	<i>Unified Modeling Language</i>
WfMC	<i>Workflow Management Coalition</i>
WfMS	<i>Workflow Management System</i>

LISTA DE FIGURAS

FIGURA 2.1 - O PROCESSO DE DESENVOLVIMENTO DE LINHA DE PRODUTO (GIMENES, 2002)	13
FIGURA 2.2 - EXEMPLO DE REPRESENTAÇÃO DE VARIABILIDADE EM CASOS DE USO (LAZILHA, 2002)	18
FIGURA 2.3 - DESENVOLVIMENTO DO PRODUTO (HALMEMAN, 2003)	21
FIGURA 2.4 - WORKFLOW DE REQUISITO (CHEESMAN, 2000)	24
FIGURA 2.5 - WORKFLOW DE ESPECIFICAÇÃO (CHEESMAN, 2000)	24
FIGURA 2.6 - ARTEFATOS PRODUZIDOS PELOS WORKFLOWS DO UML COMPONENTS (CHEESMAN, 2000)	25
FIGURA 2.7 - MODELO CONCEITUAL DE NEGÓCIO DE UM SISTEMA DE RESERVA (CHEESMAN, 2000)	26
FIGURA 2.8 - MODELO DE TIPOS DE NEGÓCIO DE UM SISTEMA DE RESERVA (CHEESMAN, 2000)	27
FIGURA 2.9 - ARQUITETURA DE COMPONENTES DE UM SISTEMA DE RESERVA (CHEESMAN, 2000)	28
FIGURA 3.1 - RELAÇÃO DA TERMINOLOGIA ASSOCIADA A UM WORKFLOW (WFMC, 1995)	30
FIGURA 3.2 - ARQUITETURA GENÉRICA DE UM SISTEMA DE WORKFLOW (WFMC, 1995)	32
FIGURA 3.3 - MODELO DE REFERÊNCIA PARA WfMS (WFMC, 1995)	33
FIGURA 3.4 - ARQUITETURA DE LINHA DE PRODUTO PARA WfMS (OLIVEIRA JUNIOR, 2004)	35
FIGURA 4.1 - DEFINIÇÕES DE INSPEÇÃO E VALIDAÇÃO DE ARQUITETURAS DE <i>WORKFLOW</i>	40
FIGURA 4.2 - <i>FRAMEWORK</i> DE PROCESSO PARA O <i>UML COMPONENTS</i>	41
FIGURA 4.3 - DIAGRAMA DE ATIVIDADE DO COMPONENTE <i>WORKFLOWARCHITECTUREMGR</i>	42
FIGURA 4.4 - MODELO DE CASO DE USO DO COMPONENTE <i>WORKFLOWARCHITECTUREMGR</i>	43
FIGURA 4.5 - MODELO CONCEITUAL DE NEGÓCIO DO COMPONENTE <i>WORKFLOWARCHITECTUREMGR</i>	43
FIGURA 4.6 - MODELO DE TIPOS DO COMPONENTE <i>WORKFLOWARCHITECTUREMGR</i>	44
FIGURA 4.7 - OPERAÇÕES DA INTERFACE DE SISTEMA <i>ICREATEARCHITECTUREMGT</i>	46
FIGURA 4.8 - DIAGRAMA DE RESPONSABILIDADES DE INTERFACE DO COMPONENTE <i>WORKFLOWARCHITECTUREMGR</i>	46
FIGURA 4.9 - DIAGRAMA DE COLABORAÇÃO DA OPERAÇÃO <i>CREATEARCHITECTURE()</i> DA INTERFACE <i>ICREATEARCHITECTUREMGT</i>	47
FIGURA 4.10 - DIAGRAMA DE COLABORAÇÃO DA OPERAÇÃO <i>SETBASELINEARCHITECTURE()</i> DA INTERFACE <i>ICREATEARCHITECTUREMGT</i>	48
FIGURA 4.11 - DIAGRAMA DE SEQÜÊNCIA DO COMPONENTE <i>WORKFLOWARCHITECTUREMGR</i>	49
FIGURA 4.12 - ASSINATURA DAS OPERAÇÕES DA INTERFACE <i>ICREATEARCHITECTUREMGT</i>	50
FIGURA 4.13 - ARQUITETURA DE COMPONENTES	50
FIGURA 4.14 - PARTE DO CÓDIGO JAVA DO COMPONENTE <i>WORKFLOWARCHITECTUREMGR</i> GERADO PELA RATIONAL ROSE	52
FIGURA 4.15 - CLASSE <i>DRIVER</i> PARA TESTES	53
FIGURA 4.16 - RESULTADOS DA SIMULAÇÃO	54

ZAUPA, Fábio G. **Desenvolvimento do Componente *WorkflowArchitectureMgr* da Linha de Produto para Sistemas de Gerenciamento de *Workflow***. Maringá: UEM, 2004 (Monografia de Especialização).

RESUMO

Há muito vem se formando uma consciência na engenharia de *software* para obtenção de produtos com alta qualidade e que sejam economicamente viáveis. Para tal, é necessário um conjunto organizado de processos, técnicas e ferramentas. Reutilização está entre as técnicas mais relevantes desse conjunto. Parte-se do princípio que reutilizando partes bem especificadas, desenvolvidas e testadas pode-se construir *software* em menor tempo e com maior confiabilidade. Porém, ainda falta nesse contexto uma maneira sistemática e previsível de realizar a reutilização. A abordagem de linha de produto de *software* surgiu como uma proposta de construção sistemática de *software* baseada em famílias de produtos. Um grupo do Departamento de Informática da UEM vem trabalhando na elaboração de uma linha de produto baseada em componentes para Sistemas de Gerenciamento de *Workflow* que visa gerar produtos desta categoria com maior facilidade e confiabilidade a partir de uma infra-estrutura predefinida. Os componentes desta arquitetura vêm sendo desenvolvidos nas diversas pesquisas realizadas pelo grupo. Assim, o objetivo deste trabalho é especificar e desenvolver o componente *WorkflowArchitectureMgr* desta arquitetura. Este componente visa apoiar a definição e a manutenção de arquiteturas de *Workflow* que permitem a definição de modelos predefinidos de *workflow*. A contribuição principal deste trabalho é o povoamento da linha de produto existente, permitindo, assim, a sua evolução incremental e avaliação.

Palavras Chaves: Arquitetura de *software*, Componente, Linha de produtos, Reutilização e Sistema de Gerenciamento de *Workflow*

ZAUPA, Fábio G. **WorkflowArchitectureMgr Component Development for the Workflow Management Systems Product Line** . Maringá: UEM, 2004 (Monograph of Specialization).

ABSTRACT

The need for software products of high quality and economically feasible has become evident in the software engineering community. Software quality requires well defined processes, techniques and tools. Reuse is considered as one of the techniques that can contribute to increase software quality. The principle is that by reusing well specified and tested software parts it is possible to reduce development time as well as production costs. However, there is still a lack of a systematic approach to make reuse more effective. The product line approach emerged in this context as a means to develop software as a family of products. A component-based product line for workflow management systems has been developed by a group in the Department of Informatics of UEM. The main infrastructure of the product line consists of its architecture and components. The components have been progressively developed within research projects by the members of the group. The objective of this work is to design the component *WorkflowArchitectureMgr* of this product line. The component aims at supporting the definition and maintenance of workflow architecture that allow the specification of predefined workflow models. Thus, this work contributes to the population and consolidation of the proposed product line.

Keywords: Software Architecture, Component, Product Line, Reuse and Workflow Management Systems

CAPÍTULO 1 - INTRODUÇÃO

Há muito vem se formando uma consciência na Engenharia de *Software* para obtenção de produtos com alta qualidade e que sejam economicamente viáveis. Para tal, é necessário um conjunto organizado de processos, técnicas e ferramentas. A reutilização está entre as técnicas mais relevantes desse conjunto. Parte-se do princípio que reutilizando partes bem especificadas, desenvolvidas e testadas pode-se construir *software* em menor tempo e com maior confiabilidade. Porém, ainda falta, nesse contexto, uma maneira sistemática e previsível de realizar a reutilização. A abordagem de linha de produto de *software* (LP) surgiu como uma proposta de construção sistemática de *software* baseada em famílias de produtos (GIMENES e TRAVASSOS, 2002).

Esta abordagem é aplicável aos domínios em que se faz necessária à construção de sistemas similares, mas com algumas características diferentes. Dessa maneira, nota-se que os Sistemas Gerenciadores de *Workflow* (WfMS – *Workflow Management Systems*) são favoráveis à aplicação dessa abordagem, uma vez que o *workflow* é a automatização total ou parcial do processo de negócio de uma organização, sendo que os WfMS são utilizados para criar, definir e gerenciar *workflows*.

Assim, desde 2000, o grupo de Engenharia de Software da UEM coordenado pela Profa. Dra. Itana Maria de Souza Gimenes vem trabalhando na elaboração de uma arquitetura baseada em componentes de linha de produto para WfMS.

Lazilha (2002) especificou uma arquitetura de LP para WfMS, a partir da arquitetura genérica e do modelo de referência da WfMC (*Workflow Management Coalition*) (1995). Este trabalho se concentrou na concepção da arquitetura. Os componentes da arquitetura foram especificados e alguns aspectos de variabilidade foram analisados, mas nenhum componente foi implementado. O desenvolvimento dos componentes para essa arquitetura é importante tanto para povoar a arquitetura, complementando assim o núcleo de artefatos de LP, quanto para validar a arquitetura proposta.

Assim, iniciativas vêm sendo realizadas pelo grupo para completar a arquitetura. Halmeman (2003) especificou o componente *WorkflowExecutionMgr* e Oliveira Junior (2004) especificou o componente *WorkflowMgr*. O desenvolvimento desses componentes implicou em revisões da arquitetura proposta por Lazilha (2002).

Dando seqüência a essas iniciativas, este trabalho visa realizar o projeto e a implementação do componente "*WorkflowArchitectureMgr*". Este componente visa apoiar a definição e a manutenção de arquiteturas de *workflow*. O componente permite definições de *workflow* mais flexíveis, que podem ser reutilizadas. Dessa forma, os tipos de objetos da arquitetura não são pré-fixados.

O desenvolvimento do componente seguiu o processo *UML Components* (CHEESMAN, 2000) para Desenvolvimento Baseado em Componentes (DBC),

No Capítulo 2 estão os conceitos fundamentais para o desenvolvimento deste trabalho, que incluem: a abordagem de LP, DBC e o processo *UML Components*. No Capítulo 3 são apresentados os conceitos da tecnologia de *workflow* e a arquitetura de LP para WfMS proposta por Lazilha (2002) e revisada por Halmeman (2003) e Oliveira Junior (2004), contextualizando o componente proposto. No Capítulo 4 é apresentado o projeto do componente *WorkflowArchitectureMgr* e, finalmente, no Capítulo 5 estão as conclusões e as propostas de trabalhos futuros.

CAPÍTULO 2 – FUNDAMENTAÇÃO TEÓRICA

Este Capítulo apresenta uma introdução à abordagem de linha de produto de *software*. Na Seção 2.1, são apresentados os conceitos básicos sobre a abordagem. Na Seção 2.2 é mostrado os elementos de uma LP, ressaltando a importância da representação das variabilidades e as atividades essenciais para o desenvolvimento de uma LP. Na Seção 2.3, estão os conceitos de DBC, seguido no desenvolvido deste trabalho. Finalmente, na Seção 2.4, o método *UML Components* é descrito sumariamente.

2.1. Conceitos Básicos

De acordo com Clements (2001) uma linha de produto de *software* é definida como uma coleção de sistemas que compartilham um conjunto gerenciado de características que satisfazem necessidades específicas de um segmento ou missão e que são desenvolvidos a partir de um núcleo de artefatos seguindo um plano previamente definido.

Segundo Lazilha (2002) a abordagem de LP é aplicável aos domínios em que há necessidade de produtos específicos, mas com um conjunto de semelhanças e pontos de variabilidade bem definidos, formando assim uma família de produtos.

Entende-se por família de produtos de *software* um conjunto de produtos de *software* com características suficientemente similares para permitir a definição de uma infra-estrutura comum de estruturação dos itens que compõem os produtos e a parametrização das diferenças entre os produtos.

A abordagem de LP visa a redução de custos e o aumento da qualidade dos produtos. Uma LP pode ser desenvolvida a partir de uma arquitetura genérica comum, a arquitetura de LP, e um conjunto de componentes que povoam a arquitetura.

Para se obter sucesso em uma LP de *software* é necessário um gerenciamento sistemático e uma cuidadosa especificação das variabilidades enquanto se explora os pontos semelhantes. As semelhanças garantem a reutilização dos artefatos de *software* e o gerenciamento das variabilidades permite que produtos com características diferentes sejam gerados a partir de uma mesma arquitetura (HALMEMAN, 2003).

2.2. Elementos de uma Linha de Produto de *Software*

Segundo Clements e Northrop (2001), as três atividades essenciais da construção de uma linha de produtos são:

- o desenvolvimento do núcleo de artefatos,
- o desenvolvimento do produto, e
- o gerenciamento da LP.

Essas três atividades estão intrinsecamente relacionadas de tal forma que a alteração em uma delas implica em analisar o impacto nas demais (GIMENES, 2002).

Clements e Northrop (2001) também se referenciam à atividade de desenvolvimento do núcleo de artefatos como engenharia de domínio e à atividade de desenvolvimento do produto como engenharia de aplicação. Weiss (1999) também incorpora engenharia de domínio e engenharia de aplicação em sua abordagem denominada FAST (*Family-Oriented Abstraction, Specification and Translation*). Isto se deve às semelhanças entre os conceitos de engenharia de domínio e LP.

Segundo Gimenes e Travassos (2002), o processo de desenvolvimento de uma LP pode ser visto como dois modelos de ciclo de vida, conforme mostra a Figura 2.1.

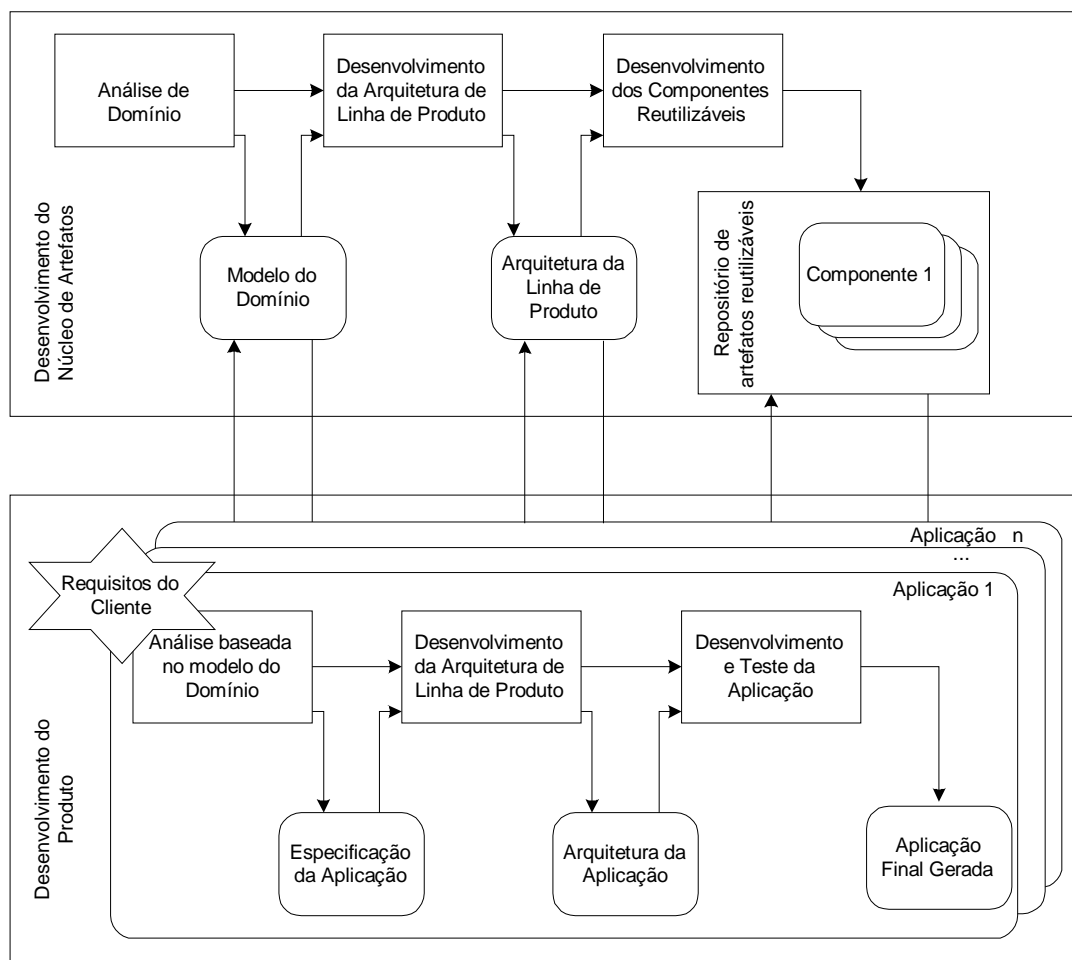


Figura 2.1 - O processo de desenvolvimento de linha de produto (GIMENES, 2002)

Nesta figura os retângulos representam as etapas de desenvolvimento, enquanto os retângulos arredondados representam os artefatos produzidos. A estrela representa os requisitos da aplicação a ser desenvolvida (produto). O modelo de desenvolvimento do núcleo de artefatos é composto de três etapas: análise de domínio, desenvolvimento da arquitetura e desenvolvimento de componentes reutilizáveis. Estas produzem um modelo de domínio, uma arquitetura e um conjunto de componentes reutilizáveis e geradores de *software* para a LP.

O desenvolvimento do produto é composto das seguintes etapas: análise baseada no modelo de domínio, desenvolvimento da arquitetura de LP,

desenvolvimento e teste da aplicação. Estas produzem a especificação da aplicação, arquitetura da aplicação e a aplicação final gerada.

2.2.1 Variabilidade

Para obter sucesso no desenvolvimento de uma LP, é necessário abstrair e representar as variabilidades relacionadas a uma arquitetura e aos seus componentes.

Os produtos de uma LP compartilham um conjunto de características, mas podem se diferenciar em termos de comportamento (conjunto de ações), atributos de qualidade, plataforma, configuração física, fatores de escala, entre muitos outros (CLEMENTS, 2001).

De acordo com Gimenes e Travassos (2002), variações são diferenças tangíveis que podem ser reveladas e distribuídas entre os artefatos da LP, sejam elas a arquitetura, os componentes, as interfaces entre os componentes ou as conexões entre componentes. As variações podem ser identificadas em qualquer fase do ciclo de desenvolvimento de uma LP, a começar pela análise de requisitos. Por exemplo, a variação mais conhecida em construção de sistemas é a existência de várias implementações para uma mesma especificação.

A representação de uma variação pode ser realizada por meio da introdução de parâmetros instanciáveis em tempo de construção, que associados aos componentes, subsistemas ou coleção de subsistemas permitem a configuração do produto (CLEMENTS, 2001). Um tipo de variação simples de ser representado é a escolha de componentes diferentes para uma mesma arquitetura. Assim, produtos podem ter maior ou menor capacidade, ou características diferentes dependendo do tipo de componente escolhido para a arquitetura (GIMENES e TRAVASSOS, 2002).

Existem vários mecanismos para tratamento de variabilidade. Por tratamento de variabilidade, entende-se desde o momento de reconhecimento de um ponto de variação até o mapeamento do ponto para uma instância de variação (GIMENES e TRAVASSOS, 2002).

Segundo Bachmann e Bass (2001), é necessário representar diferentes possibilidades de variação nos artefatos de uma arquitetura quando a representação está sendo gerada e não se sabe qual alternativa será escolhida no momento de se gerar um novo produto.

Segundo Halmeman (2003), para representar variações deve-se considerar as causas de variação e os tipos de variação. Conforme discutido a seguir:

- **Variação funcional:** acontece quando uma função existe em alguns produtos, mas não existe em outros. Por exemplo, considere um carro com um dispositivo que seja rádio e sistema de navegação. Podemos ter um carro apenas com rádio, outro apenas com sistema de navegação e outro com ambos;

- **Variação nos dados:** a estrutura dos dados pode variar de um produto para outro. Por exemplo, se considerarmos uma aplicação em que dois componentes compartilham informações sobre clientes. As informações dos clientes possuem, entre outras coisas, o endereço do cliente armazenado como uma cadeia de caracteres não estruturada. Para permitir que uma característica diferente seja implementada em uma nova versão (mostrar o endereço de forma estruturada, por exemplo) então o formato para o endereço do cliente deverá ser diferente;

- **Variação no fluxo de controle:** um padrão de interação pode variar de um produto para outro. Por exemplo, considere um mecanismo de notificação entre componentes, o qual informe aos componentes interessados que houve mudanças em alguns valores de dados. Uma possível solução para informar os demais componentes seria que todos os outros componentes fossem notificados em seqüência dentro de um fluxo de controle. No entanto, se considerarmos um ambiente distribuído, pode ser que alguns dos componentes não sejam notificados, o que causa um comportamento imprevisível do sistema. Para resolver este problema, poderia ser modificado o fluxo de controle de tal modo que os componentes enviassem um código de controle informando se recebeu a notificação;

- **Variação na tecnologia:** A tecnologia pode variar sob muitos aspectos. Se considerarmos a plataforma que é composta do sistema operacional,

hardware, interfaces com o usuário entre outras coisas. Percebe-se que um desses elementos pode variar, do mesmo modo que a variação funcional. Por exemplo, um determinado elemento pode ser necessário em um produto, mas não em outro;

- ***Variação nas metas de qualidade:*** as metas de qualidade podem variar de um produto para outro. Por exemplo, se considerarmos a forma como é feita a comunicação entre fabricantes e clientes. Esta comunicação pode ser realizada por meio de um mecanismo que envia mensagem ao cliente (não *on-line*) ou através de uma conexão permanente, a última opção oferece auxílio imediato ao cliente. A escolha de uma destas formas de comunicação implica em variação nas metas de qualidade, que podem variar de um produto para outro;

- ***Variação no ambiente:*** a maneira como um produto interage com o ambiente pode variar. Por exemplo, um determinado elemento pode ser invocado por diferentes linguagens.

Independente da causa de uma variação ela sempre estará enquadrada em um tipo de variação; os tipos estão descritos abaixo:

- ***Opcional:*** a variação é opcional se, por exemplo, a funcionalidade está em um produto, mas não em outro;

- ***Instância de várias alternativas:*** a arquitetura permite que várias funcionalidades alternativas sejam inseridas. Por exemplo, um automóvel disponibiliza um conector para onde pode ser adicionado um dispositivo de controle de viagem. Esse dispositivo pode ser de alta tecnologia e alto custo para automóveis submetidos a condições extremas ou de baixa tecnologia e baixo custo para automóveis para uso em condições normais;

- ***Conjunto de instâncias para várias alternativas:*** a arquitetura fornece a possibilidade de várias instâncias nas quais múltiplas alternativas podem ser inseridas. Por exemplo, um produto de *software* capaz de se comunicar com o mundo externo utilizando protocolos de comunicação diferentes em paralelo. Esse produto é composto de um conjunto de protocolos de comunicação enquanto outro

produto pode ter outro conjunto de protocolos. Para cada um dos protocolos, pode-se ainda, considerar diferentes funcionalidades.

Depois de identificadas as causas e o tipo da variação é possível representá-las. Um estudo mais detalhado de como realizar a representação pode ser encontrado em Bachmann e Bass (2001). Outra maneira de representar os pontos de variação é através do conceito de *features*. Este conceito tem origem na engenharia de domínio (KANG, 1990). Uma *feature* é uma característica de um produto que usuários e clientes consideram importantes na descrição e na distinção de membros de uma família de produto (GRISS, 1998). Uma *feature* pode ser um requisito específico, uma seleção entre os requisitos opcionais e alternativos; ou pode estar relacionada a certas características do produto como funcionalidade, usabilidade e desempenho, ou pode estar relacionada às características de implementação como o tamanho, a plataforma de execução ou compatibilidade com certos padrões.

Um modelo de *features* consiste da representação dos tipos de *features* que podem ocorrer em uma família de produtos e seus inter-relacionamentos através de grafos.

O modelo de *features* e o modelo de casos de uso apresentam alguma relação, mas algumas diferenças devem ser observadas conforme Griss (1998), Gimenes e Travassos (2002). São elas:

- o modelo de *features* é orientado para o reutilizador e o modelo de casos de uso é orientado para o usuário;
- o modelo de casos de uso descreve os requisitos do usuário em termos de funcionalidades do sistema enquanto o modelo de *features* organiza o resultado da análise dos aspectos comuns e variáveis com o objetivo de preparar a base para a reutilização;
- o modelo de casos de uso deve cobrir todos os requisitos de um sistema de domínio. Já no modelo de *features*, devem estar representadas as características que o analista do domínio considera importantes, pois este resume apenas os itens essenciais relativos aos objetivos do domínio;

- a notação utilizada para representar casos de uso é diferente daquela utilizada para representar *features*, pois nem todas as *features* podem ser automaticamente relacionadas como casos de uso;
- mesmo existindo um grupo de *features* básicas que o usuário vê como capacidades do sistema, nem todas essas *features* aparecem nos casos de uso. Algumas *features* surgem: no detalhamento da implementação, nas opções de configuração do sistema, ou por sugestão de especialistas do domínio. Tais *features* podem aparecer somente nas fases de projeto e implementação.

Uma discussão sobre *features* relacionadas com casos de uso é feita por Jacobson, Griss e Jonsson (1997). Uma *feature* é definida como um caso de uso ou parte deste. Neste contexto é sugerida a utilização do estereótipo <<extend>> para representar aspectos de variação em casos de uso. O caso de uso estendido recebe uma marca, representada por um círculo preenchido para indicar um ponto de variação, como mostrado na Figura 2.2.

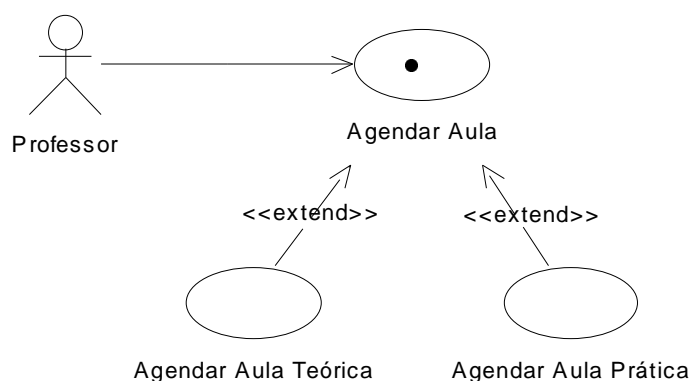


Figura 2.2 - Exemplo de representação de variabilidade em casos de uso (LAZILHA, 2002).

2.2.2. Desenvolvimento do Núcleo de Artefatos

No desenvolvimento do núcleo de artefatos, três aspectos devem ser considerados: a definição do contexto da LP, o núcleo de artefatos e o plano de produção. Um artefato é um item reutilizável de *software* utilizado como bloco de construção de uma LP. Pode ser uma arquitetura de *software*, um componente,

um *framework*, ou alguma documentação relativa à arquitetura de *software* ou a algum componente.

2.2.2.1. Contexto da Linha de Produto

O domínio da LP é a descrição dos produtos que constituirão a LP ou quais produtos a linha será capaz de incluir. O domínio da LP deve ser cuidadosamente definido, pois uma definição muito ampla pode comprometer a identificação das variações.

2.2.2.2. Núcleo de artefatos

O núcleo de artefatos, representado na Figura 2.3 como repositório de artefatos reutilizáveis, é a base para a construção de produtos em uma LP. É composto da arquitetura e do conjunto de componentes de *software* que são desenvolvidos para reutilização na LP. Inclui também modelos de desempenho, resultados da avaliação da arquitetura, documentação de projeto, especificação de requisitos, modelos de domínio e componentes COTS (*Commercial Off-The-Shelf*).

2.2.2.3. Plano de Produção

O plano de produção descreve como os produtos são produzidos a partir do núcleo de artefatos. Em uma LP o plano de produção é o que direciona a reutilização. O plano de produção descreve os requisitos de variação entre os produtos, como e quais ferramentas específicas serão utilizadas, qual a ordem de utilização das ferramentas e a maneira de promover a junção dos artefatos produzidos (HALMEMAN, 2003).

O plano de produção gerará as seguintes entradas para a atividade de desenvolvimento do núcleo de artefatos:

- **Restrições do produto:** descreve os elementos em comum e as variações existentes entre os produtos que irão constituir a LP, as características de comportamento, os padrões que devem ser seguidos, os limites de desempenho que devem ser observados, os sistemas com os quais o produto deverá interagir, as restrições de *hardware*, os requisitos de qualidade que devem ser observados, as características de mercado e as novas tecnologias que poderão beneficiar a LP no futuro;
 - **Estilos de arquiteturas, frameworks e padrões:** estes elementos são utilizados para projetar as entradas necessárias para o núcleo de artefatos;
 - **Restrições de produção:** determinam quais os padrões específicos das organizações deverão ser aplicados no desenvolvimento da LP;
 - **Estratégia de produção:** abrange tudo que se refere à concretização do núcleo de artefatos;
 - **Repositório de artefatos já existentes:** o repositório inclui todos os possíveis artefatos já existentes em sistemas legados e através de uma análise cuidadosa é possível identificar quais serão os mais adequados para compor o núcleo de artefatos.

2.2.3. Desenvolvimento do Produto

A atividade de desenvolvimento do produto, também é conhecida como engenharia da aplicação, e depende dos seguintes elementos:

- modelo do domínio da LP;
- núcleo de artefatos, especificamente um *framework* de arquitetura de LP que é utilizado como base para especificar uma arquitetura para um membro da família;
- conjunto de componentes reutilizáveis a partir do qual um subconjunto de componentes será integrado à arquitetura para gerar o produto;
- plano de produção;
- requisitos específicos para os produtos individuais.

A Figura 2.3 mostra este relacionamento.

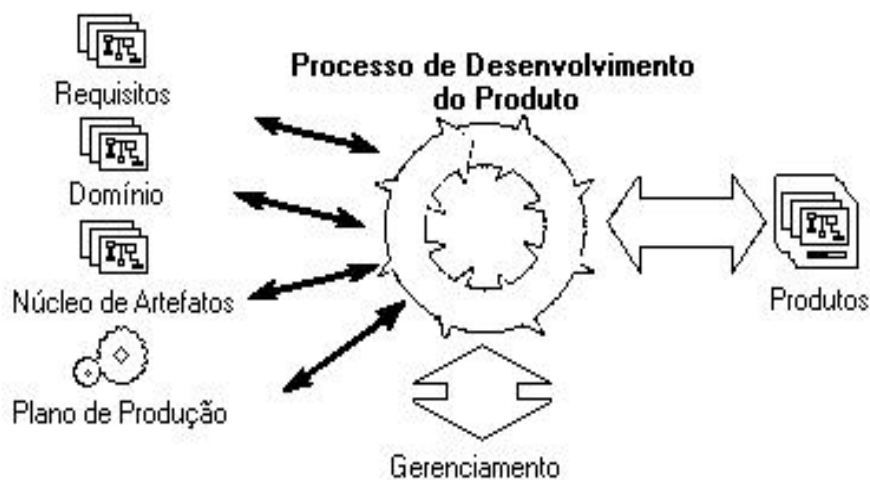


Figura 2.3 - Desenvolvimento do Produto (HALMEMAN, 2003).

2.2.4. Gerenciamento de Linha de Produto

O gerenciamento é extremamente importante para o desenvolvimento de uma LP. Para as atividades devem-se oferecer artefatos, coordenação e supervisão.

Para a construção de uma LP, deve-se exercer um forte gerenciamento, tanto em nível técnico quanto em nível organizacional. O gerenciamento é importante no período de desenvolvimento e também para manter a integridade da LP.

O gerenciamento técnico incide diretamente sobre o desenvolvimento do núcleo de artefatos e no desenvolvimento de produtos. Ele deve garantir que estes elementos estejam dentro dos requisitos da LP, de acordo com os planos de produção e coletar dados para verificar o progresso das atividades.

O gerenciamento organizacional está voltado para a estrutura da organização, deve verificar e garantir que os produtos e recursos produzidos estejam de acordo com as necessidades da organização.

As etapas de Desenvolvimento do Produto e Gerenciamento estão fora do escopo deste trabalho.

2.3. Desenvolvimento Baseado em Componentes

O crescente amadurecimento dos métodos de Desenvolvimento Baseados em Componentes (DBC) tem contribuído com a abordagem de LP, pois quebra blocos monolíticos em componentes interoperáveis, reduz a complexidade no desenvolvimento e os custos através da reutilização de componentes que podem ser adequados a outras aplicações.

Os componentes são os elementos que preenchem a arquitetura de uma LP. Dessa forma, a disponibilidade de mecanismos para especificar e construir componentes de forma a permitir variações entre estes e que possam facilitar a geração de aplicações com características diferentes com menor esforço de desenvolvimento é essencial para o enfoque de LP.

Segundo Werner (2000), até algum tempo atrás, o desenvolvimento de grande parte dos produtos de software disponíveis no mercado era baseado em uma abordagem de desenvolvimento em blocos monolíticos, formados por um grande número de partes inter-relacionadas, estes relacionamentos estavam na maioria das vezes implícitos.

Segundo D'Souza (1999), um componente pode ser definido como uma unidade de software independente, que encapsula dentro de si seu projeto e implementação, e oferece interfaces bem definidas para o meio externo, para que este componente possa se unir a outros componentes e dar origem aos sistemas baseados em componentes. As interfaces são chamadas de Interfaces Fornecidas (*Provided Interfaces*) ou Interfaces Requisitadas (*Required Interfaces*). A interface fornecida define as operações que o componente oferece a outros componentes. A interface requisitada define as operações que o componente requisitará de outros componentes. As interfaces servem apenas para a comunicação entre componentes, ocultando dos usuários os detalhes de implementação. A especificação de um componente é, normalmente, publicada separadamente de seu código fonte por meio da especificação das interfaces oferecidas por ele (GIMENES, 2000).

Segundo Bosch (2000), componentes podem ser classificados em:

- componentes desenvolvidos pelo próprio cliente;
- componentes adquiridos para um determinado domínio; e
- os chamados componentes *Commercial-Of-The-Shelf* (COTS) ou

componentes de prateleira, que são componentes genéricos encontrados no mercado.

O DBC visa fornecer um conjunto de ferramentas e notações que possibilitem que, ao longo do processo de software, ocorra tanto a produção de novos componentes quanto a reutilização de componentes existentes. Para Werner (2000) é primordial o entendimento dos conceitos que estão por trás das funcionalidades disponibilizadas pelos componentes e seus relacionamentos.

O amadurecimento dessas técnicas também contribui muito com o enfoque de LP, oferecendo mecanismos para especificar e construir componentes de forma a permitir variações entre estes e que possam facilitar a geração de aplicações com características diferentes reduzindo o esforço no desenvolvimento. Estes métodos estão focados nos seguintes elementos: componentes, interfaces e construção de componentes. A notação utilizada é a *Unified Modeling Language* (UML) (BOOCH; RUMBAUGH; JACOBSON, 1999).

O processo *UML Components* será descrito na próxima seção, pois foi escolhido para guiar o processo de desenvolvimento do componente *WorkflowArchitectureMgr*.

2.4. O Processo *UML Components*

O *UML Components* (CHESSMAN, 2000) é um processo que visa projetar arquiteturas e especificar sistemas de grande porte baseados em componentes. Este processo apresenta de forma clara e direta como especificar os componentes que formam um sistema. O processo considera tanto o desenvolvimento com reuso quanto o desenvolvimento para reuso (OLIVEIRA JUNIOR, 2004).

O *UML Components* é dividido em dois *workflows*: *Workflow de Requisito* (*Requirements*) (Figura 2.4) e *Workflow de Especificação* (*Specification*) (Figura 2.5).

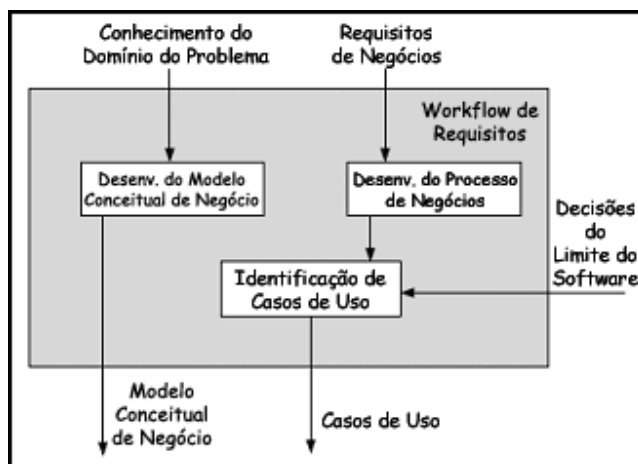


Figura 2.4 - Workflow de Requisito (CHEESMAN, 2000).

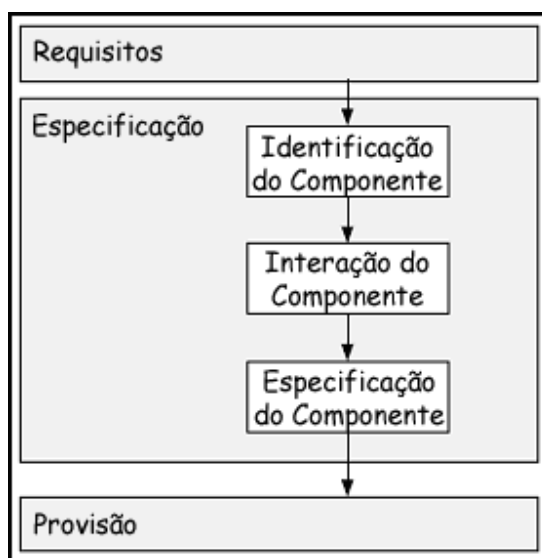


Figura 2.5 - Workflow de Especificação (CHEESMAN, 2000).

A Figura 2.6 apresenta a organização dos artefatos de acordo com os dois *workflows* do *UML Components*.

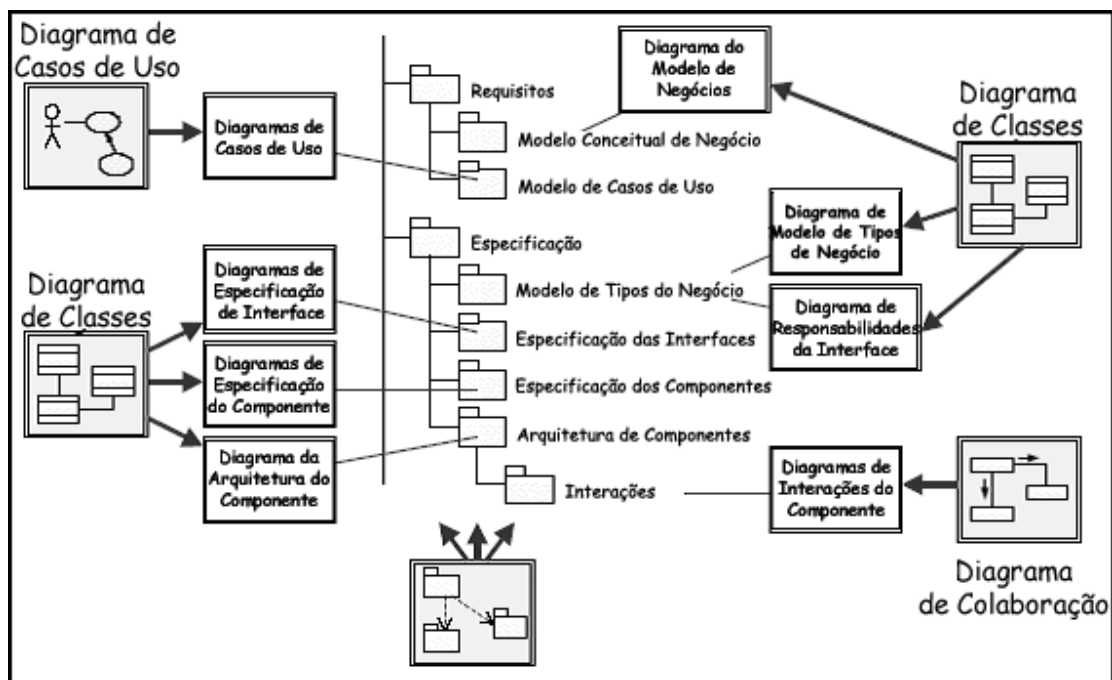


Figura 2.6 - Artefatos Produzidos pelos Workflows do UML Components (CHEESMAN, 2000).

A seguir é apresentada uma descrição de cada um dos principais artefatos produzidos pelos *workflows* (CHEESMAN, 2000). Também são apresentados alguns exemplos destes artefatos.

Workflow de Requisito (*Requirements Workflow*)

- Modelo Conceitual de Negócio (*Business Concept Model*):** representa o modelo conceitual de informações do domínio. O principal objetivo deste modelo é capturar as classes conceituais e identificar as suas relações. Este modelo é representado por um diagrama de classes da UML, porém é um modelo independente de software. Cada classe identificada pode, opcionalmente, possuir o estereótipo <<concept>>. As relações entre as classes podem ou não ter suas multiplicidades especificadas. O mesmo acontece para os atributos relevantes às classes, porém estes não precisam ter seus tipos especificados. A Figura 2.7

apresenta um exemplo deste tipo de diagrama para um sistema de reserva (CHEESMAN, 2000);

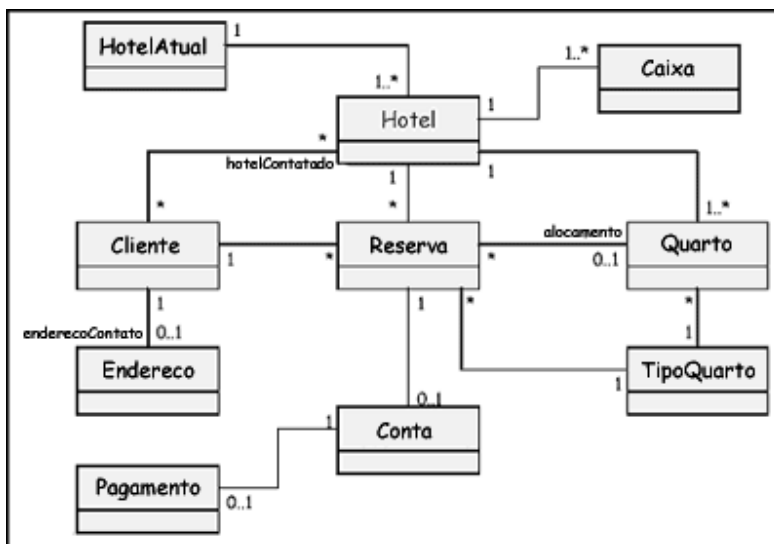


Figura 2.7 - Modelo Conceitual de Negócio de um Sistema de Reserva (CHEESMAN, 2000).

- **Modelo de Caso de Uso (Use Case Model):** representa a modelagem semiformal da interação usuário-sistema. Este modelo é formado por diagramas de caso de uso da UML e a descrição destes.

Workflow de Especificação (Specification Workflow)

- **Modelo de Tipos do Negócio (Business Type Model):** representa a modelagem lógica precisa das informações relevantes ao escopo do sistema. Este modelo é formado por diagramas de classes da UML. As classes do modelo de tipos podem possuir os seguintes estereótipos: <<type>>, <<datatype>>, <<interface type>> ou <<interface>>. Neste modelo, os atributos são especificados e os seus tipos são definidos. Neste modelo também é produzido o diagrama de responsabilidades de interface, onde são identificados os tipos <<core>> e <<Interface type>>. Esses estereótipos servem para identificar a classe associada a uma interface de negócio e a própria interface de negócio

respectivamente. A Figura 2.8 apresenta um exemplo deste tipo de diagrama para um sistema de reserva (CHEESMAN, 2000);

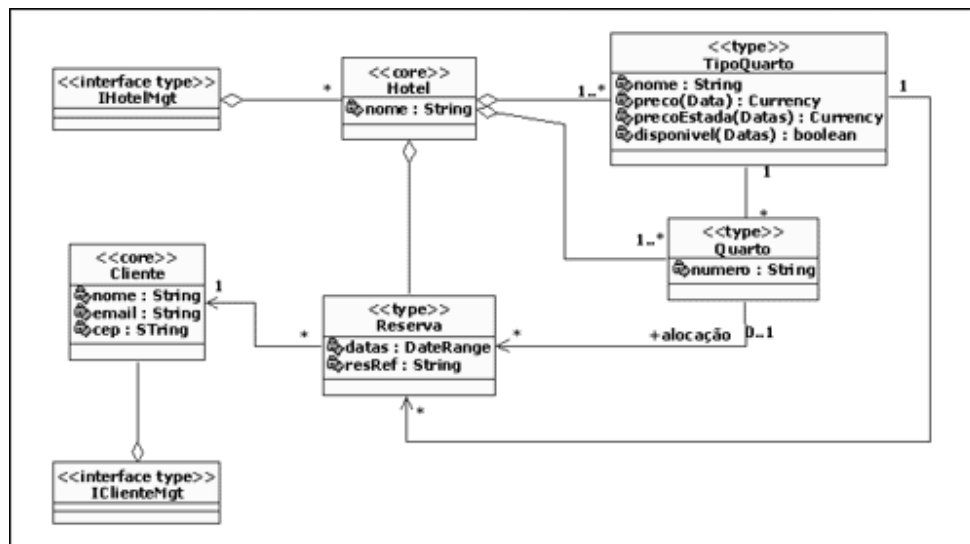


Figura 2.8 - Modelo de Tipos de Negócio de um Sistema de Reserva (CHEESMAN, 2000).

- **Especificação das Interfaces (*Interface Specification*):** representa a especificação das interfaces. Uma especificação de interface consiste das seguintes partes: a própria interface; o modelo de informação da interface contendo os seus atributos e as suas associações; a especificação das suas operações (pré e pós-condições); e a especificação de alguma invariante adicional. Este modelo é representado por diagramas de classes da UML;

- **Especificação dos Componentes (*Component Specification*):** representa a especificação da realização das interfaces fornecidas por um componente. Os componentes possuem o estereótipo `<<comp spec>>`. Este modelo é representado por diagramas de classe da UML;

- **Arquitetura de Componentes (*Component Architecture*):** representa um conjunto de componentes, suas interfaces (serviços fornecidos) e as suas relações. A arquitetura de componentes é especificada através de diagramas de componentes da UML. Os componentes podem possuir os estereótipos `<<comp spec>>` ou `<<existing>>`. A Figura 2.9 apresenta um exemplo de arquitetura de componentes para um sistema de reserva (CHEESMAN, 2000).

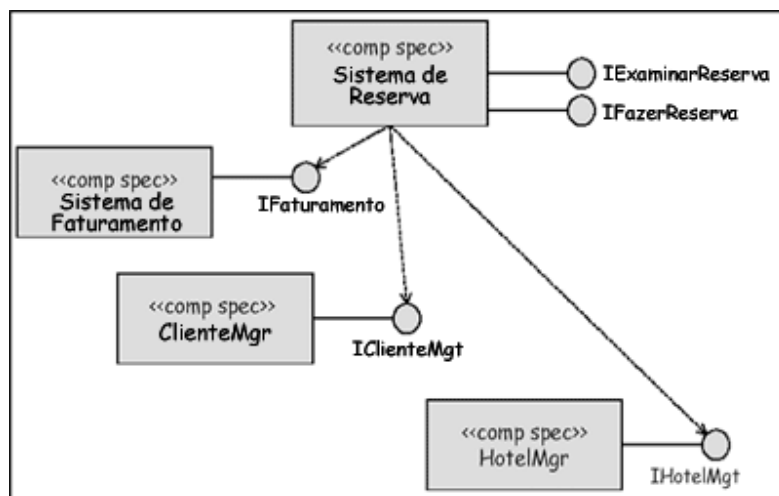


Figura 2.9 - Arquitetura de Componentes de um Sistema de Reserva (CHEESMAN, 2000).

2.5. Considerações Finais

Este capítulo apresentou os conceitos básicos de LP e os seus elementos. Dentre os elementos foram citados os conceitos de variabilidades e as atividades essenciais para a produção de uma LP como o Desenvolvimento do Núcleo de Artefatos, o Desenvolvimento do Produto e o Gerenciamento. Foram conceituados o Desenvolvimento Baseado em Componentes e o processo *UML Components*.

No próximo capítulo serão apresentados os conceitos para a Tecnologia de *Workflow* e a arquitetura para LP proposta por Lazilha (2002).

CAPÍTULO 3 - A TECNOLOGIA DE *WORKFLOW*

Este capítulo apresenta os conceitos e técnicas de *workflow* fundamentais para o desenvolvimento deste trabalho. A seção 3.1 apresenta uma definição da tecnologia de *workflow*. Na seção 3.2, é mostrado os conceitos e termos relacionados a *workflow*. Já nas seções 3.3 são apresentados os WfMS com a arquitetura genérica proposta pela WfMC e o modelo de referência para WfMS. Para finalizar, a Seção 3.4, apresenta a arquitetura de LP para WfMS proposta por Lazilha (2002) e revisada por Halmeman (2003) e Oliveira Junior (2004). O componente *WorkflowArchitectureMgr*, desenvolvido neste trabalho, faz parte dessa arquitetura.

3.1. Definição de *Workflow*

Um processo de negócio é um conjunto de procedimentos e atividades que conduzem à realização de uma meta de negócio de uma organização. A WfMC (1995) define *workflow* como sendo uma coleção de atividades organizadas para realizar um processo de negócio. Dessa forma, um *workflow* estabelece a ordem de execução das atividades e as condições em que cada atividade pode ser iniciada, assim como a sincronização das atividades, o fluxo de informações e os participantes das atividades.

Um *workflow* é definido como a automação total ou parcial de um processo de negócio, de modo que se possa, de uma maneira mais eficiente controlar, simular e maximizar a eficiência dos processos de negócio (WFMC, 2001).

3.2. Conceitos e Termos Relacionados a *Workflow*

Os principais conceitos e terminologia associados a um *workflow* são ilustrados na Figura 3.1.

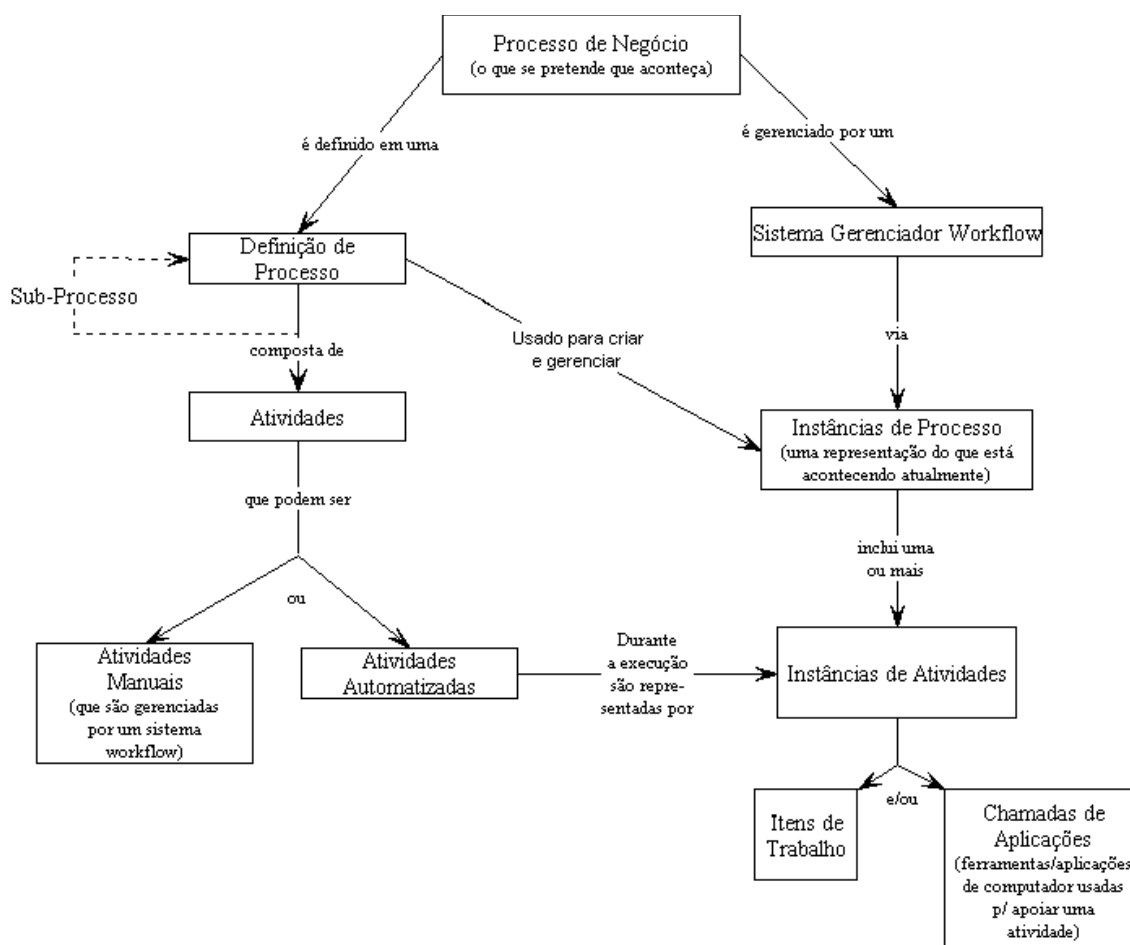


Figura 3.1 - Relação da Terminologia Associada a um Workflow (WFMC, 1995).

Esses termos, além do processo de negócio já definido na seção 3.1, compõem um Sistema de Gerenciamento de *Workflow* que é o responsável por fornecer os mecanismos para a automação dos processos de negócios, incluindo acesso a recursos humanos ou de máquinas de acordo com as atividades; definição de processo que representa um processo de negócio de modo a permitir manipulações automatizadas, como simulação, ou execução por um sistema de gerenciamento de *workflow*; sub-processo que é um processo chamado de outro processo hierarquicamente superior; atividade ou tarefa que é uma parte ou porção de trabalho que forma um passo lógico dentro de um processo, podendo ser manual ou automatizada; instância de processo que é a representação de uma única execução de um processo; instância de atividade que é a representação de uma atividade em uma única execução de um processo; item de trabalho que é

uma partição de uma atividade; chamada de aplicação que é uma aplicação de *workflow* invocada pelo WfMS para automatizar, totalmente ou parcialmente, uma atividade, ou auxiliar um participante do *workflow* no processamento de um item de trabalho; ator ou participante que é responsável pela execução total ou parcial de uma determinada instância de atividade; e finalmente, cargo ou papel, que pode ser definido como um atributo de participante que determina suas responsabilidades e direitos na execução de um *workflow*.

3.3. Sistemas de Gerenciamento de *Workflow* (WfMS)

WfMS são os sistemas que permitem a definição, criação e gerenciamento da execução de *workflows* com apoio computacional. Eles executam uma ou mais máquinas de *workflow*, interpretam a definição de processos, interagem com os participantes do *workflow* e, quando necessário, invocam o uso de ferramentas externas (WfMC, 1995).

3.3.1. Arquitetura Genérica para *WfMS*

A WfMC, mesmo com a grande quantidade de produtos no mercado, identificou a viabilidade da construção de um modelo de implementação genérico de um sistema de *workflow* que pode ser tomado como ponto de partida para a construção da maioria dos produtos do mercado (LAZILHA, 2002). Este modelo identifica os principais componentes funcionais e suas respectivas interfaces dentro de um WfMS. A arquitetura genérica de um WfMS é apresentada na Figura 3.2.

- **Dados de controle do sistema:** são dados utilizados pelos componentes de *software*, tais como dados de controle de *workflow*, dados relevantes do *workflow* e o modelo organizacional;
- **Aplicativos e suas bases de dados:** são os aplicativos que não fazem parte dos WfMS, mas podem ser invocados por ele como parte do sistema de *workflow*.

3.3.2. Modelo de Referência

Após a análise dos elementos que compõem a arquitetura genérica, e das interfaces, a WfMC propôs o Modelo de Referência para WfMS que consiste em padronizar o desenvolvimento de aplicações baseadas na tecnologia de *workflow* propondo uma arquitetura básica para o desenvolvimento das aplicações.

A Figura 3.3 ilustra os componentes e interfaces que compõem o modelo de referência.

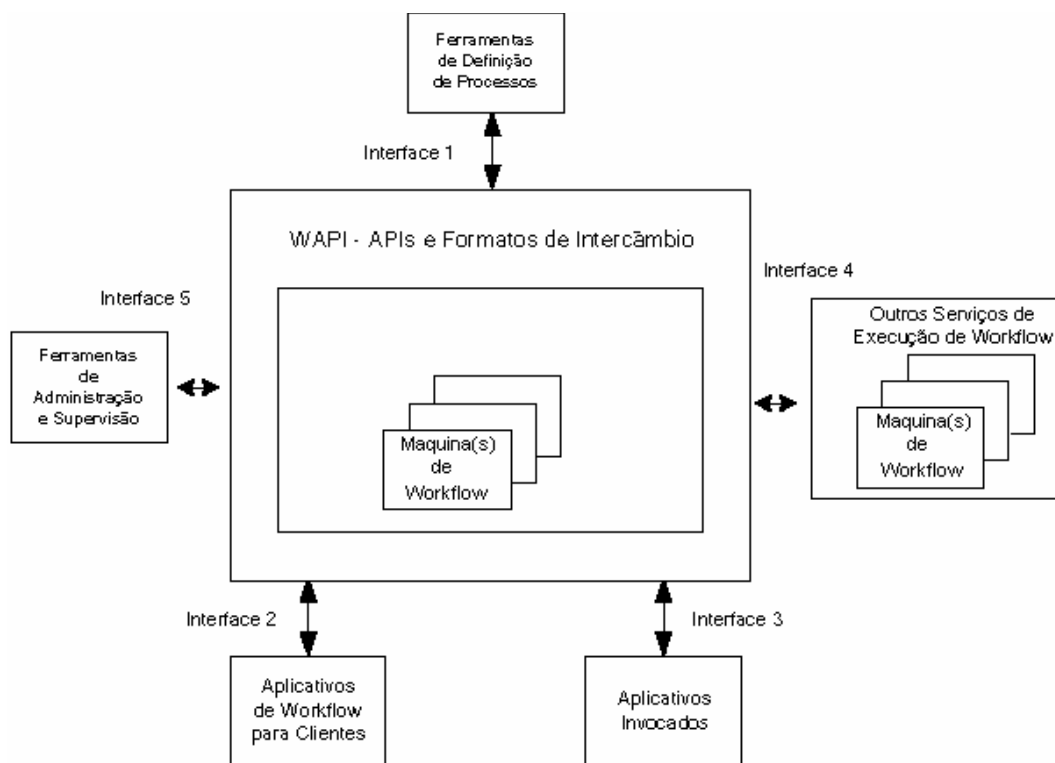


Figura 3.3 - Modelo de Referência para WfMS (WfMC, 1995).

A seguir estão as descrições dos componentes do Modelo de Referência (WfMC, 1995).

- **Serviço de execução de workflow:** este serviço fornece o ambiente e os mecanismos de execução nos quais as ativações e instanciações dos processos acontecem. O serviço de *software* que consiste de uma ou mais máquinas de *workflow* para criar, gerenciar e executar instâncias de *workflow*. A interação com o serviço de execução ocorrerá via interface de aplicações de *workflow* para clientes (Interface 2) ou interface de aplicações invocadas (Interface 3);
- **Definição de processos:** uma variedade de ferramentas podem ser utilizadas para analisar, modelar, descrever e documentar um processo de negócio (Interface 1). É importante que ao final do projeto e modelagem, o processo de definição possa ser interpretado e executado pelas máquinas *workflow*;
- **Aplicações de workflow para clientes:** é o *software* gerenciador de listas de trabalhos que interage com o usuário final nas atividades que envolvem recursos humanos (Interface 2);
- **Aplicações invocadas:** o sistema gerenciador de *workflow* necessita invocar a aplicação com o objetivo de automatizar uma atividade que faça parte de um item de trabalho de um participante (Interface 3). Existe a necessidade de mecanismos de integração que garantam a comunicação entre a máquina de *workflow* e a aplicação invocada;
- **Interoperabilidade entre serviços de execução de workflow:** permite a troca de itens de trabalho entre diferentes WfMSs (Interface 4). Logo, diferentes máquinas *workflow* podem trabalhar em conjunto. O foco de padronização neste item está na forma de cooperação;
- **Ferramentas de supervisão e administração:** estas ferramentas de supervisão e administração permitem a avaliação do estado geral e extração de métricas do sistema, o que é importante para as organizações (Interface 5).

3.4. A Arquitetura de Linha de Produto para WfMS

A partir da análise da arquitetura genérica e do modelo de referência para WfMS (WfMC, 1995), Lazilha (2002) propôs uma arquitetura para WfMS segundo a abordagem de LP.

A arquitetura foi revisada nos estudos realizados por Halmeman (2003) e Oliveira Junior (2004), em que foram encontrados e corrigidos alguns problemas.

A Figura 3.4 apresenta a arquitetura atual.

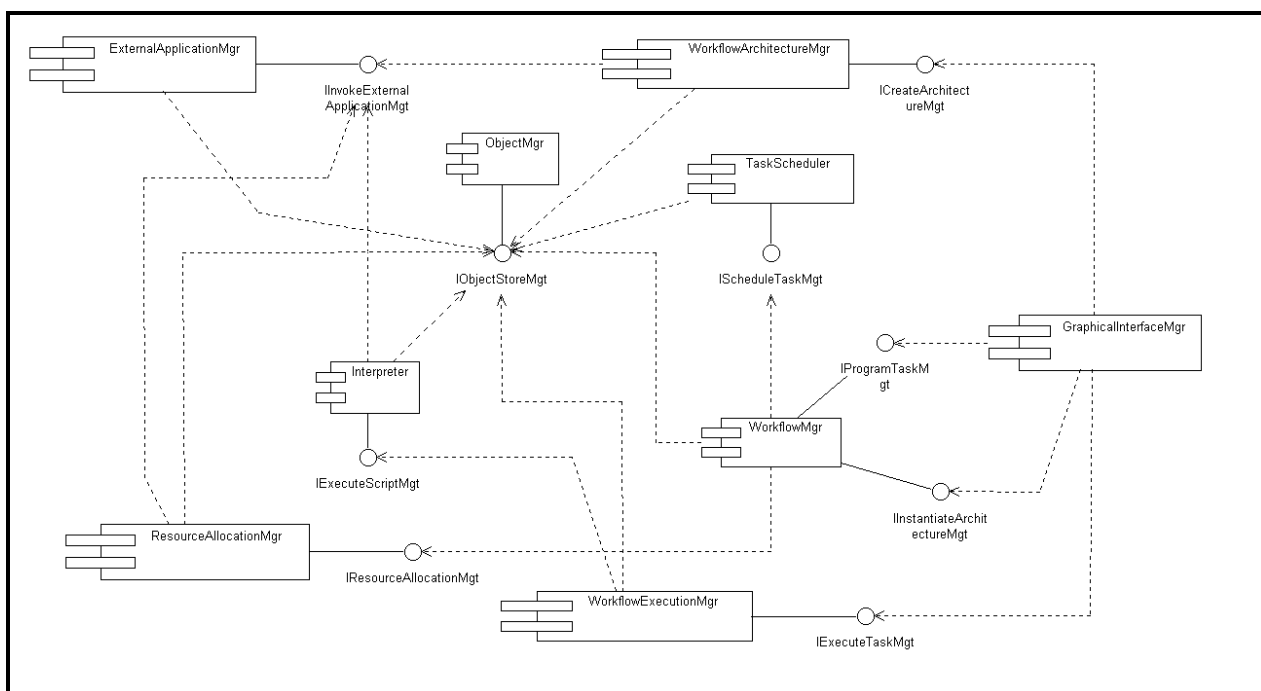


Figura 3.4 - Arquitetura de Linha de Produto para WfMS (OLIVEIRA JUNIOR, 2004).

A seguir são descritas as principais finalidades de cada um dos componentes da arquitetura:

- **GraphicalInterfaceMgr**: responsável pelo gerenciamento da interface gráfica com o usuário do sistema. A interface pode ser uma interface gráfica “convencional” ou via *browser*;
- **WorkflowArchitectureMgr**: responsável pelo controle e gerenciamento da construção e manutenção de arquiteturas de *workflow*. A implementação deste componente suporta as funções relacionadas à definição das arquiteturas de *workflow* e os tipos de objetos relacionados a esta. A utilização deste componente

torna a definição de *workflows* mais flexíveis, visto que, dessa forma, os tipos de objetos não são pré-fixados;

- **WorkflowMgr:** responsável pela criação e gerenciamento de projetos que incorporam *workflows*. Os projetos incluem a instanciação e execução de arquiteturas de *workflow*. A cada projeto existe um *workflow* associado. Para cada tipo de objeto existente na arquitetura, instancia-se um objeto no *workflow*. Em seguida, as tarefas do projeto são programadas e gerenciadas, fazendo-se a alocação de recursos e demais tomadas de decisão. O projeto deste componente foi realizado nos estudos realizados por Oliveira Junior (2004);
- **WorkflowExecutionMgr:** responsável pelo controle, gerenciamento e execução das tarefas a serem realizadas no *workflow*. Sua principal característica é apoiar a interação com os usuários do WfMS. É através dele que os usuários identificam as suas tarefas no *workflow*. O projeto deste componente foi feito nos estudos realizados por Halmeman (2003);
- **TaskScheduler:** responsável pelo controle, gerenciamento e escalonamento das tarefas e ações a serem realizadas pelos usuários de *workflow*. Esse componente permite que os usuários identifiquem suas tarefas geradas pelo Gerenciador de *Workflow*;
- **ResourceAllocationMgr:** responsável pelo controle e alocação de recursos (atores, ferramentas ou material);
- **ExternalApplicationMgr:** responsável pelo gerenciamento das aplicações externas invocadas durante a definição do *workflow* e execução das tarefas;
- **ObjectMgr:** responsável pelo relacionamento com os mecanismos de persistência de objetos manipulados pelo sistema. Este componente trabalha com objetos que vão desde dados de controle até instâncias de *workflow* e tarefas. Todos os componentes pertencentes à arquitetura de componentes proposta utilizam seus serviços direta ou indiretamente. Sua presença torna o restante dos componentes independentes de uma implementação particular de um sistema gerenciador de objetos, garantindo flexibilidade e portabilidade para toda a coleção de componentes existentes;
- **Interpreter:** responsável pela execução dos scripts.

3.5. Considerações Finais

Neste capítulo foram apresentados os conceitos básicos da tecnologia de *workflow* tais como definições, conceitos e termos relacionados. Também foram citados a Arquitetura Genérica para WfMS, os Sistemas de Gerenciamento de *Workflow* e o Modelo de Referência.

A arquitetura de LP para WfMS, proposta por Lazilha (2002) e revisada por Halmeman (2003) e Oliveira Junior (2004), é fundamental na elaboração deste trabalho, pois é a arquitetura da qual o componente *WorkflowArchitectureMgr* faz parte.

O desenvolvimento do componente *WorkflowArchitectureMgr* contribui para o povoamento da arquitetura de LP para WfMS. No próximo capítulo será apresentada a especificação deste componente.

CAPÍTULO 4 – O COMPONENTE *WORKFLOWARCHITECTUREMGR*

O objetivo deste trabalho é completar o Núcleo de Artefatos de LP para WfMS. Para tanto, este capítulo apresenta o projeto do componente *WorkflowArchitectureMgr*

O processo seguido para projetar o *WorkflowArchitectureMgr* é o *UML Components* para DBC (CHEESMAN, 2000).

Para a avaliação do componente *WorkflowArchitectureMgr* foi simulada a sua execução acionando o componente e simulando a definição de uma arquitetura de *workflow*, com base em cenários definidos nos diagramas de seqüência da LP.

A Seção 4.1 apresenta uma introdução sobre as características e as funcionalidades do componente *WorkflowArchitectureMgr*. Na Seção 4.2, é apresentada a especificação do componente. Na Seção 4.3, é apresentado o código gerado pela ferramenta IBM/Rational Rose 2002, a partir das especificações da seção 4.2. Para finalizar, na Seção 4.4 é apresentada a avaliação do componente.

4.1. Projeto do Componente *WorkflowArchitectureMgr*

O componente *WorkflowArchitectureMgr* é responsável pelo controle e gerenciamento da construção e manutenção de arquiteturas de *workflow*. Este componente apóia as funções relacionadas à definição de arquiteturas de *workflow* e dos respectivos tipos de objetos relacionados a esta. A utilização deste componente torna a definição dos *workflows* mais flexível, visto que, dessa forma, os tipos de objetos não são pré-fixados, pois uma mesma arquitetura pode ser utilizada para definir vários *workflows*.

Este componente está diretamente relacionado com outros três componentes da arquitetura de LP para WfMS (LAZILHA, 2002): *GraphicalInterfaceMgr*, *ObjectMgr* e *ExternalApplicationMgr* (ver arquitetura na seção 3.4).

Através da interface gráfica (componente *GraphicalInterfaceMgr*) que interage com o componente *WorkflowArchitectureMgr*, o gerente de projeto poderá instanciar uma arquitetura de *workflow*. Os procedimentos a serem seguidos para construir uma arquitetura estão descritos a seguir:

1. criar uma nova arquitetura ou abrir uma arquitetura já existente mais não finalizada;
2. adicionar os tipos de objetos que a arquitetura terá, como tarefas, artefatos, papéis, recursos que podem ser especializados em ator, material e ferramenta. O tipo de ferramenta também pode ser especializado em interna ou externa. O componente *ExternalApplicationMgr* será usado para verificar se existem ferramentas externas. Nenhum objeto será especificado nessa fase;
3. inspecionar a arquitetura para validá-la. Nesse passo são verificados se os tipos de objetos envolvidos na definição da arquitetura estão de acordo com as regras impostas. Estas regras definem que um tipo de tarefa pode ser precedido e/ou composto por zero ou mais tipos de tarefas. Esses tipos de tarefas devem utilizar um ou mais tipos de artefatos para a produção de um ou mais tipos de artefatos. Sendo assim, cada tipo de artefato pode depender de zero ou mais tipos de artefatos. Para esta ação de transformação, os tipos de tarefas devem utilizar um ou mais tipos de ferramentas e requerer a utilização de um ou mais tipos de cargos. Um tipo de cargo pode ser liderado por um ou mais outros tipos de cargos. Além disso, os tipos de cargos devem manipular um ou mais tipos de ferramentas e acessar um ou mais tipos de artefatos, segundo um conjunto de tipos de direitos (leitura, escrita e/ou execução), enquanto os tipos de ferramentas devem manusear um ou mais tipos de artefatos segundo um conjunto de tipos de ações válidas sobre estes tipos de artefatos. Para cada tarefa, deve ser alocado um ator responsável pela sua execução. Os atores devem ocupar um ou mais cargos, enquanto que os cargos devem ser ocupados por um ou mais atores, definindo assim quais são os direitos e deveres que cada ator possui. A figura 4.1 resume essas definições;

Arquitetura	Tarefa (1 ou *)
Tarefa	Tarefa (0 ou *)
	Artefato (1 ou *)
	Recurso (1 ou *)
Artefato	Artefato (0 ou *)
	Ferramenta/Ações (1 ou *)
	Cargo/Direito (1 ou *)
Cargo	Cargo (0 ou *)
	Ator (1 ou *)

Figura 4.1 - Definições de Inspeção e Validação de Arquiteturas de *Workflow*

4. verificar se a arquitetura pode ser finalizada (uma vez finalizada não poderá mais ser alterada);
5. salvar a arquitetura, enviado-a para o componente responsável (*ObjectMgr*).

4.2. Especificação do Componente

A especificação do componente *WorkflowArchitectureMgr* foi realizada seguindo o processo *UML Components* (CHEESMAN, 2000). Foi elaborado, inicialmente, um *framework* de processo para o *UML Components* com o auxílio da ferramenta CASE IBM/Rational Rose. Os termos foram mantidos em inglês para serem identificados mais facilmente às etapas do desenvolvimento, além de manter o trabalho padronizado ao processo *UML Components*. A Figura 4.2 apresenta o *framework* elaborado.

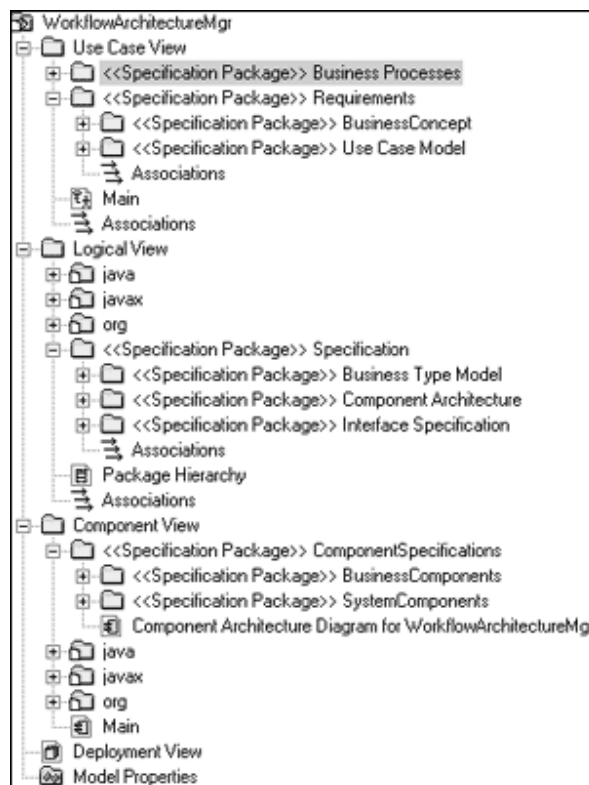


Figura 4.2 - *Framework* de Processo para o *UML Components*

O estereótipo <<Specification Package>> indica que os pacotes contêm os artefatos produzidos durante a especificação do componente. As seções a seguir apresentam os artefatos produzidos durante a especificação do componente *WorkflowArchitectureMgr*.

4.2.1. *Workflow* de Requisito (*Requirements Workflow*)

No *workflow* de requisito foram produzidos os diagramas de atividades do componente, que auxilia na identificação dos elementos do modelo conceitual de negócios; o Modelo de Casos de Uso, que auxilia na identificação das interações usuário-sistema; e o Modelo Conceitual de Negócio, que serve para capturar as classes conceituais e identificar as suas relações.

Esses artefatos formam a base para a identificação e especificação dos componentes, suas interfaces e métodos, nos próximos estágios.

Os artefatos produzidos neste *workflow* são apresentados nas Figuras 4.3, 4.4 e 4.5.

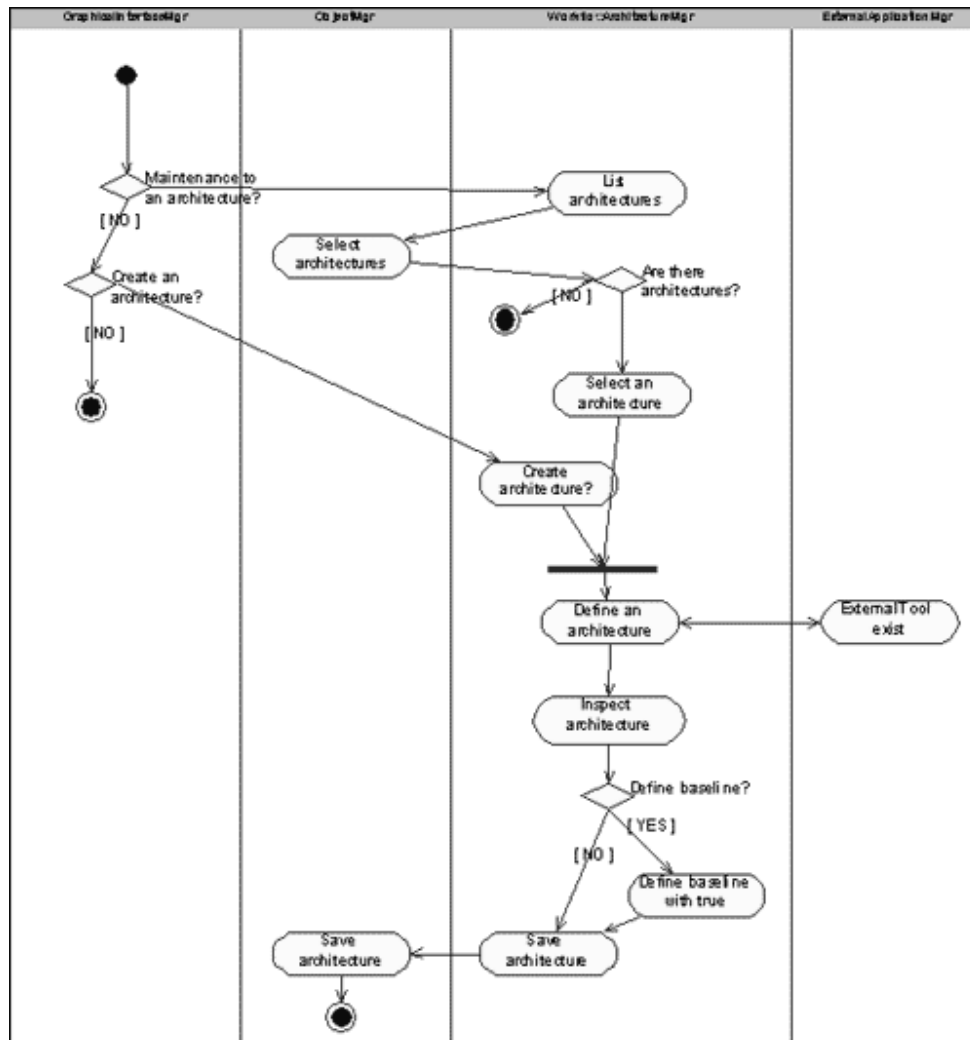


Figura 4.3 - Diagrama de Atividade do Componente *WorkflowArchitectureMgr*

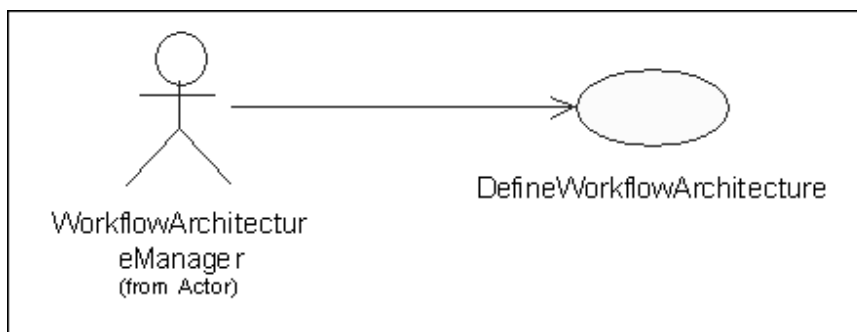


Figura 4.4 - Modelo de Caso de Uso do Componente WorkflowArchitectureMgr

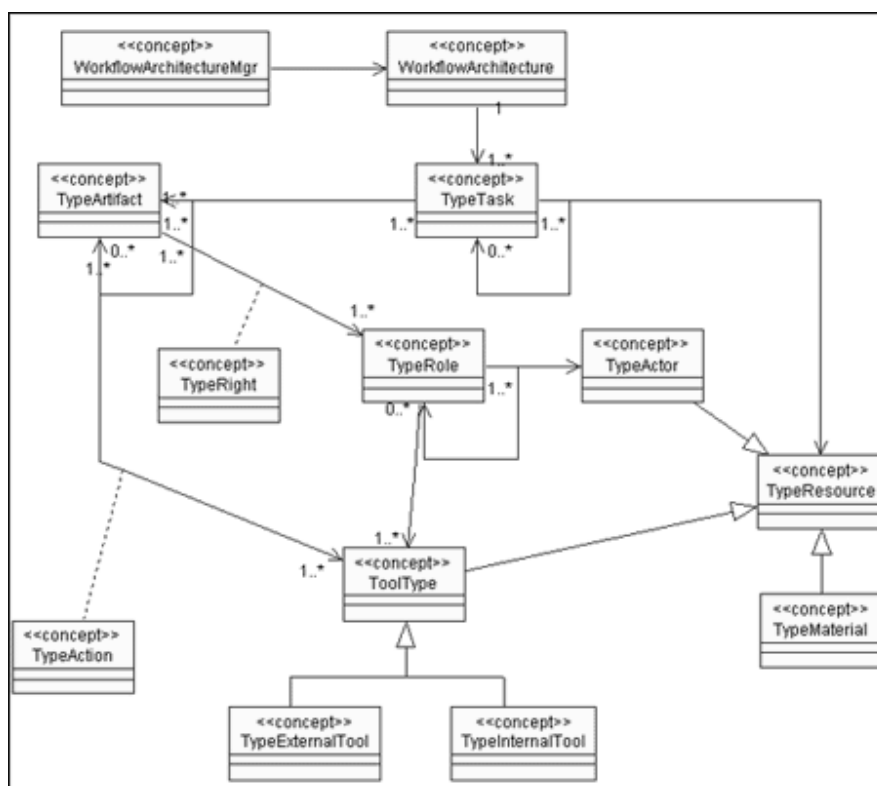


Figura 4.5 - Modelo Conceitual de Negócio do Componente WorkflowArchitectureMgr

Os requisitos foram encontrados através da análise isolada do componente e interagindo com os demais componentes da arquitetura. Também foi utilizado o trabalho de Fantinato (1999) como fonte de referência para coleta dos requisitos.

4.2.2. Workflow de Especificação (*Specification Workflow*)

O *workflow* de especificação é responsável por identificar os componentes e as suas interfaces, mostrar suas interações e especificá-los usando, para isso, diagramas da UML.

A seguir são apresentados os artefatos produzidos em cada etapa deste *workflow*.

4.2.2.1. Component Identification

A partir do Modelo Conceitual de Negócios chegamos ao Modelo de Tipos de Negócio, que representa a modelagem lógica precisa das informações relevantes ao contexto do sistema. A Figura 4.6 apresenta o Modelo de Tipos desenvolvido.

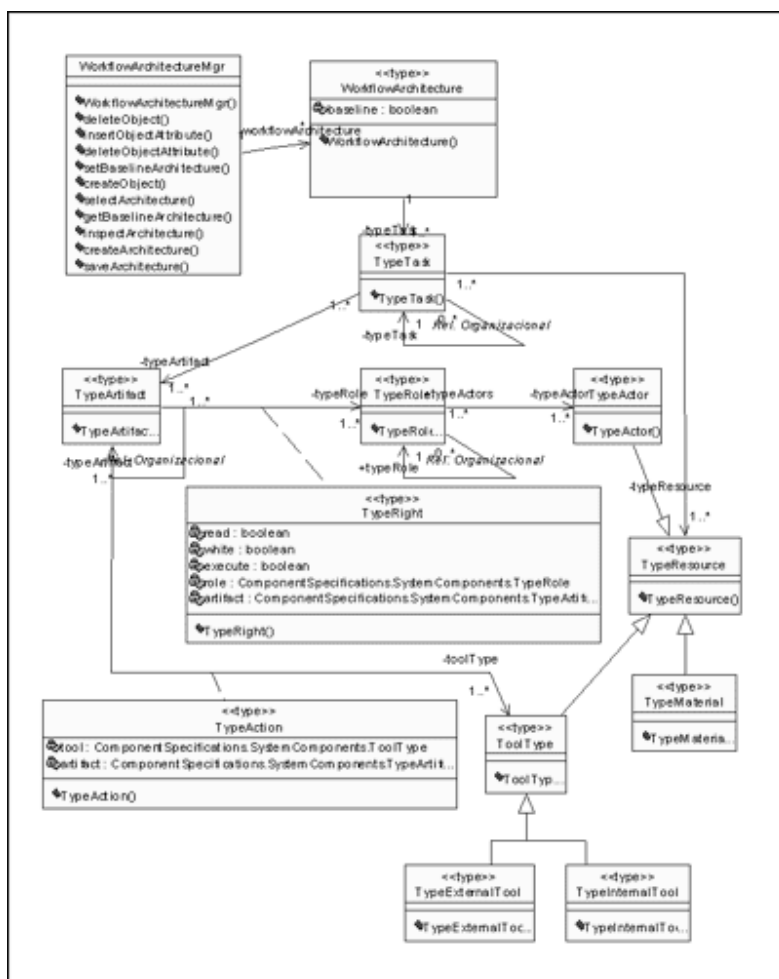


Figura 4.6 - Modelo de Tipos do Componente *WorkflowArchitectureMgr*

Neste modelo, os atributos foram especificados e os seus tipos foram definidos.

Através da análise do modelo de casos de uso, chegamos até a definição de uma única interface de sistema para o componente *WorkflowArchitectureMgr*, pois, segundo o processo *UML Components*, para cada caso de uso é definida uma interface de sistema.

Os casos de uso são divididos em passos e estes revelam as operações da interface, porém as operações ainda não possuem suas assinaturas definidas.

Foram identificados os seguintes passos para o caso de uso *DefineWorkflowArchitecture* (Figura 4.4):

- 1- Criar uma arquitetura;
- 2- Selecionar uma arquitetura de *workflow* finalizada;
- 3- Adicionar tipos de objetos a arquitetura;
- 4- Excluir tipos de objetos de uma arquitetura;
- 5- Adicionar um novo atributo de definição na arquitetura;
- 6- Excluir um novo atributo de definição na arquitetura;
- 7- Verificar se uma arquitetura está finalizada;
- 8- Definir uma arquitetura como finalizada (*baseline*);
- 9- Validar uma arquitetura;
- 10- Salvar uma arquitetura.

A partir dos passos identificados, a interface de sistema denominada *ICreateArchitectureMgt* teve suas operações identificadas. A Figura 4.7 apresenta as operações dessa interface de sistema.

Operações	Descrição
createArchitecture()	Criar uma nova arquitetura
selectArchitecture ()	Selecionar uma arquitetura não finalizada
createObject ()	Adiciona um novo tipo de objeto na arquitetura
deleteObject ()	Exclui um tipo de objeto de uma arquitetura
insertObjectAttribute ()	Adiciona um novo atributo de definição na arquitetura
deleteObjectAttribute ()	Exclui um atributo de definição de uma arquitetura

InvokeExternalApplicationMgt (interface do componente *ExternalApplicationMgr*) e *IObjectStoreMgt* (interface do componente *ObjectMgr*) formam encontradas.

4.2.2.2. Interação dos Componentes (Component Interaction)

Seguindo o *UML Components*, é necessário representar a interação entre as interfaces de sistema identificadas, através de diagramas de colaboração da UML, com o propósito de identificar a assinatura de cada uma das operações destas interfaces.

Para cada operação da interface *ICreateArchitectureMgt*, foi desenvolvido um diagrama de colaboração. As Figuras 4.9 e 4.10 apresentam respectivamente dois dos diagramas de colaboração desenvolvidos: a interação com a operação *createArchitecture()* e a interação com a operação *setBaselineArchitecture()*.

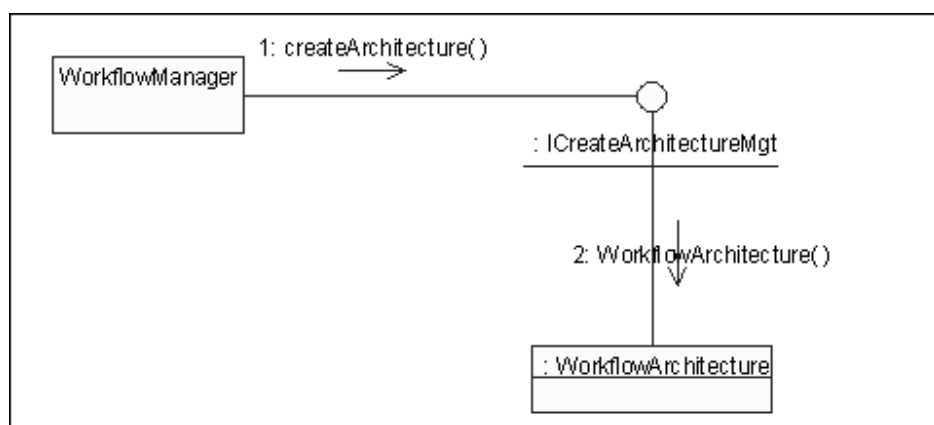


Figura 4.9 - Diagrama de Colaboração da Operação *createArchitecture()* da Interface *ICreateArchitectureMgt*

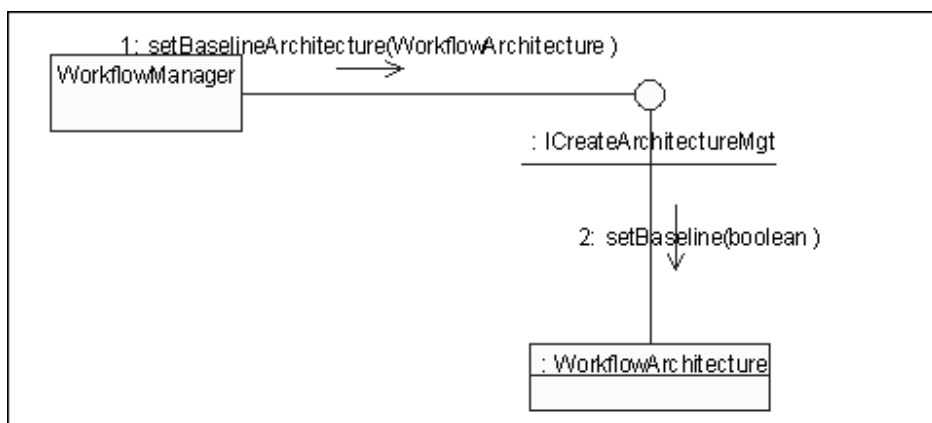


Figura 4.10 - Diagrama de Colaboração da Operação `setBaselineArchitecture()` da Interface `ICreateArchitectureMgt`

Também foi desenvolvido um diagrama de seqüência para mostrar mais claramente a interação interna do componente *WorkflowArchitectureMgr*, mostrando as trocas de informações entre as operações. A Figura 4.11 apresenta este diagrama.

A Figura 4.12 apresentam as assinaturas das operações da interface *ICreateArchitectureMgr* do componente *WorkflowArchitectureMgr*.

Assinatura das Operações
createArchitecture(): boolean
selectArchitecture(WorkflowArchitecture architecture): boolean
createObject(Object object , Object type): boolean
deleteObject(Object object , Object type): boolean
insertObjectAttribute(Object object, Object objectAttribute): boolean
deleteObjectAttribute(Object object, Object objectAttribute): boolean
getBaselineArchitecture(): boolean
setBaselineArchitecture(): boolean
inspectArchitecture(Object object): boolean
saveArchitecture(): boolean

Figura 4.12 - Assinatura das Operações da Interface *ICreateArchitectureMgr*

4.2.2.3. Especificação dos Componentes (Component Specification)

Nesta etapa os componentes da arquitetura do sistema são especificados, conforme mostra a Figura 4.13.

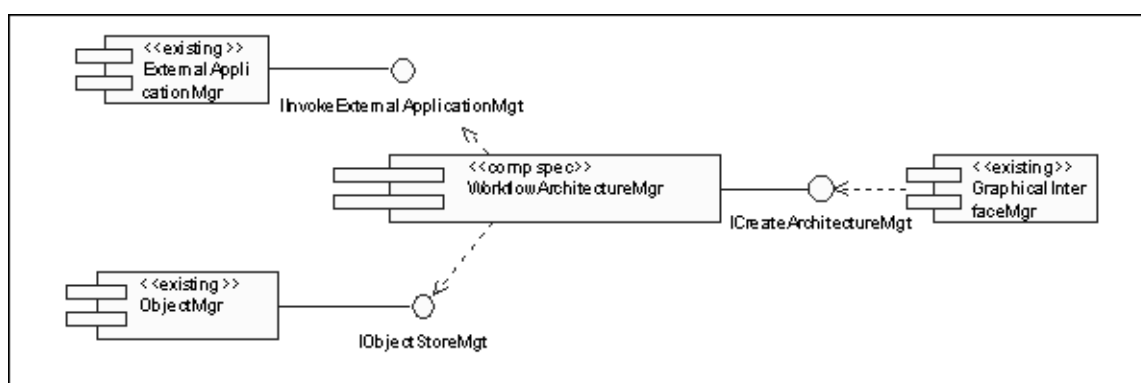


Figura 4.13 - Arquitetura de Componentes

O componente *WorkflowArchitectureMgr* possui o estereótipo `<<comp spec>>` indicando que este é o componente modelado. Os demais componentes da arquitetura possuem o estereótipo `<<existing>>` indicando que estes

componentes já foram especificados e podem ter os seus serviços acessados pelo componente *WorkflowArchitectureMgr*. O componente *GraphicalInterfaceMgr* não fornece serviços através de interfaces, mas usufrui dos serviços prestados pela interface *ICreateArchitectureMgt* do componente modelado.

4.3. Geração do Código

A ferramenta IBM/Rational Rose permite gerar código a partir de modelos bem especificados e sintaticamente corretos para várias linguagens como: Java, C++, Visual Basic, entre outras (OLIVEIRA JUNIOR, 2004).

A especificação resultante do componente *WorkflowArchitectureMgr* permitiu a geração do seu código em Java. Assim, a Figura 4.14 apresenta parte do código gerado do componente *WorkflowArchitectureMgr*.

```
package ComponentSpecifications.SystemComponents;

import ComponentSpecifications.BusinessComponents.ObjectMgr;
import ComponentSpecifications.BusinessComponents.ExternalApplicationMgr;

public class WorkflowArchitectureMgr implements IArchitectureMgt {
    private WorkflowArchitecture workflowArchitecture;

    public WorkflowArchitectureMgr(){
    }

    public boolean createObject(Object object){
        return true;
    }

    public boolean deleteObject(Object object, Object type){
        return true;
    }

    public boolean insertObjectAttribute(Object object, Object objectAttribute){
        return true;
    }

    public boolean deleteObjectAttribute(Object object, Object objectAttribute){
        return true;
    }

    public boolean getBaselineArchitecture(){
        return true;
    }

    public void setBaselineArchitecture(){
        return true;
    }
}
```

```

public boolean selectArchitecture(WorkflowArchitecture architecture){
    return null;
}

public boolean inspectArchitecture(Object object){
    return true;
}

public boolean createArchitecture(){
    return true;
}

public boolean saveArchitecture(){
    return true;
}
}

interface IArchitectureMgt {
    public boolean createArchitecture();
    public boolean selectArchitecture(WorkflowArchitecture architecture);
    public boolean createObject(Object object, Object type);
    public boolean deleteObject(Object object, Object type);
    public boolean insertObjectAttribute(Object object, Object objectAttribute);
    public boolean deleteObjectAttribute(Object object, Object objectAttribute);
    public boolean setBaselineArchitecture();
    public boolean getBaselineArchitecture();
    public boolean inspectArchitecture(Object object);
    public boolean saveArchitecture();
}
}

```

Figura 4.14 - Parte do Código Java do Componente *WorkflowArchitectureMgr* Gerado pela Rational Rose.

4.4. Avaliação do Componente *WorkflowArchitectureMgr*

Para a avaliação do componente *WorkflowArchitectureMgr* foi implementada uma classe *driver* em Java para fazer a simulação da execução do componente. A Figura 4.15 apresenta esta classe.

```

package ComponentSpecifications.SystemComponents;

public class Test {

    public static void main(String[] arg){
        try{
            WorkflowArchitectureMgr test=new WorkflowArchitectureMgr();
            // Criar arquitetura
            System.out.println("Criar arquitetura =" +test.createArchitecture());

            // Criar Tarefas
            System.out.println("Criar tarefa 1="+test.createObject(test.workflowArchitecture,new TypeTask());
            System.out.println("Criar tarefa 2="+test.createObject(test.workflowArchitecture,new TypeTask());

            // Criar subtarefas, recursos e artefatos
            Object[] task1=test.workflowArchitecture.listTypeTask();
            System.out.println("Criar Artefato="+test.createObject((TypeTask) task1[0] ,new TypeArtifact());
            System.out.println("Criar Resource="+test.createObject((TypeTask) task1[0] ,new TypeMaterial());

            System.out.println("Criar Artefato="+test.createObject((TypeTask) task1[1] ,new TypeArtifact());

```

```

System.out.println("Criar Resource="+test.createObject((TypeTask) task1[1],new TypeMaterial()));

// Criar Artefatos, ferramentas e cargos
Object[] artefato1=((TypeTask) task1[0]).listTypeArtifact();
//System.out.println("Criar no Artefato/SubArtefato="+test.createObject((TypeArtifact)
artefato1[0],new TypeArtifact());
System.out.println("Criar no Artefato/Cargos="+test.createObject((TypeArtifact) artefato1[0],new
TypeRole());
System.out.println("Criar no Artefato/Ferramenta="+test.createObject((TypeArtifact)
artefato1[0],new TypeInternalTool());
System.out.println("Criar no Artefato/Ferramenta="+test.createObject((TypeArtifact)
artefato1[0],new TypeExternalTool());
Object[] artefato2=((TypeTask) task1[1]).listTypeArtifact();
System.out.println("Criar no Artefato/Cargos="+test.createObject((TypeArtifact) artefato2[0],new
TypeRole());
System.out.println("Criar no Artefato/Ferramenta="+test.createObject((TypeArtifact)
artefato2[0],new TypeExternalTool());

// Criar atores
Object[] direito1=((TypeArtifact) artefato1[0]).listTypeRight();
System.out.println("Criar no Cargos/ Ator="+test.createObject(((TypeRight)
direito1[0]).getTypeRole(),new TypeActor());
System.out.println("Criar no Cargos/ Ator="+test.createObject(((TypeRight)
direito1[0]).getTypeRole(),new TypeActor());
Object[] direito2=((TypeArtifact) artefato2[0]).listTypeRight();
System.out.println("Criar no Cargos/ Ator="+test.createObject(((TypeRight)
direito2[0]).getTypeRole(),new TypeActor());

// Inspeccionar arquitetura
System.out.println("Baseline Atual="+test.getBaselineArchitecture());
System.out.println("Inspeccionar="+test.inspectArchitecture(test.workflowArchitecture));
System.out.println("Baseline Ativar="+test.setBaselineArchitecture());
System.out.println("Baseline Atual="+test.getBaselineArchitecture());

System.out.println("Ok");
} catch (Exception e){
    System.out.println("Erro");
}
}
}

```

Figura 4.15 - Classe *Driver* para Testes.

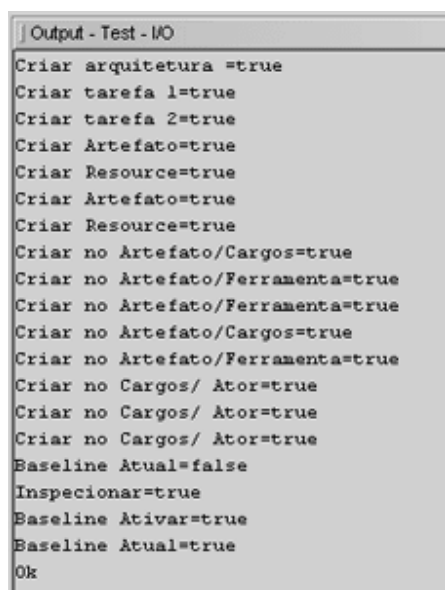
A classe *driver* de teste aciona o componente através da instanciação da classe principal do componente e, em seguida, chama as operações da interface nela implementadas na mesma seqüência que um gerente de projetos o faria, caso a interface gráfica estivesse disponível.

As primeiras operações chamadas são para criar e preencher a arquitetura com seus respectivos tipos. Em seguida são chamadas as operações para validar e finalizar a arquitetura construída. Todas as operações são chamadas juntamente com funções que permitem exibir o resultado em forma de texto na tela.

4.4.1. Análise dos Resultados

Para fazer a simulação, foi aberto o projeto do componente no IDE NetBeans 3.6 e incluída a classe *driver* Java nesse projeto. Em seguida, foi feita a execução da classe.

A simulação mostrou os resultados da execução. Esses resultados podem ser vistos na Figura 4.16. Os resultados apontaram que o componente desenvolvido está de acordo suas especificações. A classe *driver* permitiu que fosse criada uma arquitetura de *workflow* de forma bem próxima do feito em produção, o que demonstra que as funcionalidades requeridas para o componente foram satisfeitas, mas a ausência de outros componentes implementados, principalmente os que interagem com o componente *WorkflowArchitectureMgr*, prejudicou muito a fase de testes e a demonstração mais clara do seu funcionamento.



```
Output - Test - IAO
Criar arquitetura =true
Criar tarefa 1=true
Criar tarefa 2=true
Criar Artefato=true
Criar Resource=true
Criar Artefato=true
Criar Resource=true
Criar no Artefato/Cargos=true
Criar no Artefato/Ferramenta=true
Criar no Artefato/Ferramenta=true
Criar no Artefato/Cargos=true
Criar no Artefato/Ferramenta=true
Criar no Cargos/ Ator=true
Criar no Cargos/ Ator=true
Criar no Cargos/ Ator=true
Baseline Atual=false
Inspeccionar=true
Baseline Ativar=true
Baseline Atual=true
Ok
```

Figura 4.16 - Resultados da Simulação

4.5 Considerações Finais

Este capítulo apresentou as características e as formas de utilização do componente *WorkflowArchitectureMgr*, bem como a especificação dos seus requisitos.

Também foi gerado o código do componente pela ferramenta Rational Rose e realizada a implementação a partir da especificação do componente.

A avaliação do componente *WorkflowArchitectureMgr* foi feita por uma classe *driver* que simulou a chamada de suas operações. Essa avaliação ficou prejudicada devido à falta do componente *GraphicalInterfaceMgr*, que é responsável pelo gerenciamento da interface gráfica com o usuário do sistema. Dessa forma, não foi possível mostrar claramente o funcionamento do componente.

Não foram analisados nesse trabalho questões sobre as variabilidades do componente *WorkflowArchitectureMgr*, deixando isso para um próximo trabalho.

No próximo capítulo serão apresentadas as conclusões e propostas para trabalhos futuros.

CAPÍTULO 5 – CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou o projeto do componente *WorkflowArchitectureMgr*, que faz parte da arquitetura proposta por Lazilha (2002) para WfMS e revisada por Halmeman (2003) e Oliveira Junior (2004). Esse componente é responsável pelo controle, gerenciamento e manutenção de arquiteturas de *workflow*.

Para o desenvolvimento desse trabalho foram estudados os conceitos de linha de produto de software, sistemas de gerenciamento de *workflow* e desenvolvimento baseado em componentes.

O desenvolvimento do componente seguiu o processo *UML Components* para o DBC de arquiteturas de LP.

A principal contribuição do projeto é a especificação e implementação do componente *WorkflowArchitectureMgr*, que incrementa o núcleo de artefatos da LP para WfMS, contribuindo para o seu desenvolvimento.

A avaliação do componente foi feita através de uma classe *driver* que simulou a execução do componente. Esse artifício foi utilizado devido à falta de outros componentes implementados na arquitetura. Os resultados obtidos demonstraram que o componente está de acordo com suas especificações. O componente, a classe *driver* de teste e o projeto podem ser encontrados no endereço <http://www.embusca.com.br/fabiozaupa>.

Dentre os trabalhos futuros possíveis, pode-se destacar a identificação de variabilidades associadas ao componente *WorkflowArchitectureMgr* e também o projeto e implementação dos demais componentes da arquitetura, de modo que no futuro possam ser gerados produtos a partir do núcleo de artefatos da LP para WfMS.

REFERÊNCIAS BIBLIOGRÁFICAS

BACHMANN, F.; BASS, L. **“Managing Variability in Software Architectures”**, Sysmposium on Software Reusability, Toronto, Canada. 18 – 20 may 2001.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language Users Guide**. Addison-Wesley Publishing Company, 1999.

CLEMENTS, P. **Software Product Line Fundamentals**. Addison- Wesley, 2001.

CLEMENTS, P.; NORTHROP, L. M. **Software Product Lines: Practices and Patterns**. SEI Series in Software Engineering, Ed. Addison Wesley. 563 p, 2001.

CHEESMAN, J., DANIELS, J., **UML Components: A Simple Process for Specifying Component-Based Software**. *Addison-Wesley*, Upper Saddle River-NJ, 2000.

D’SOUZA, D., WILLS, A. **Objects, Components and Frameworks with UML – The Catalysis Approach**, Addison-Wesley Publishing Company, 1999. 816 p.

FANTINATO, M. **Um Escalonador de Tarefas para o ExPSee**. Monografia. Universidade Estadual de Maringá, 1999.

GIMENES, I. M. S.; HUZITA, E. H. M.; BARROCA, L. CARNIELLO, A. **O Processo de Desenvolvimento Baseado em Componentes Através de Exemplos**. ERI 2000 VIII ESCOLA DE INFORMÁTICA DA SBC-SUL, Foz do Iguaçu, Ed. Raul Ceretta, 2000. 252 p.

GIMENES, I. M. S., TRAVASSOS, G. H. **O Enfoque de Linha de Produto para Desenvolvimento de Software**. Florianópolis, SC: SBC, 2002. 34 p. Trabalho apresentado na Jornada de Atualização em Informática, 2002.

GIMENES, I. M. S., OLIVEIRA JUNIOR, E. A., LAZILHA, R. F., BARROCA, L., **A Product Line Architecture for Workflow Management Systems with**

Component-based Development. In *IEEE International Conference on Information Reuse and Integration*, Las Vegas, United States, p. 112-119, October 2003.

GRISS, M. **Integrating Feature Modeling with RSEB.** Proc. of the and International Conference on Software Reuse (ICSR-5), ACM/IEEE, Victoria, Canadá, Junho de 1998.

HALMEMAN, R. J., GIMENES, I. M. S., **Projeto do Componente Gerenciador de Execução de Workflow Segundo a Abordagem de Linha de Produto de Software.** Dissertação de Mestrado, Universidade Federal do Paraná, Curitiba-PR, julho de 2003.

JACOBSON, I.; GRISS, M.; JONSSON, P. **Software Reuse – Architecture Process and Organization for Business Success.** New York: Ed. Addison-Wesley, 1997. 497 p.

LAZILHA, F. R. **Uma Proposta de arquitetura de Linha de Produto para Workflow Management Systems.** Dissertação de Mestrado. Universidade Federal do Rio Grande do Sul. Instituto de Informática. Porto Alegre. Janeiro de 2002.

OLIVEIRA JUNIOR, E. A. GIMENES, I.M.S. **Portando o ExpSEE para a Plataforma Linux, Relatório Semestral de Iniciação Científica,** PPG-UEM, Maringá, 2002.

OLIVEIRA JUNIOR, E. A., GIMENES, Itana Maria de Souza, LAZILHA, Fabrício Ricardo, NISHIMURA, Ruy Tsutomu. **Implementação de uma Arquitetura de Linha de Produto para Sistemas de Gerenciamento de Workflow em ArchJava,** IV Congresso Brasileiro de Computação, 2004, Itajaí-SC.

Anais do IV Congresso Brasileiro de Computação. Itajaí-SC: Editora UNIVALI, 2004. v.1. p.88 - 94

OLIVEIRA JUNIOR, E. A., NISHIMURA, R. T., **Especificação de um Componente de Gerenciamento de *Workflow* de Acordo com o Processo *UML Components***, Programa de Pós-Graduação em Ciência da Computação, Universidade Estadual de Maringá, abril de 2004.

RATIONAL. **Rational Rose**, Disponível em: <<http://www.rational.com>>. Acesso em: 20 out. 2004.

TANAKA, S. **Um Framework de Agenda de Tarefas para Gerenciadores de Processos**. 2000. 124p. Dissertação de Mestrado. Instituto de Informática. Universidade Federal do Rio Grande do Sul, Porto Alegre.

WEISS, D. M.; CHI TAU, R. L. **Software Product-Line Engineering: A Family-Based Software Development Approach**. Addison-Wesley, 1999.

WERNER, C. M. L., BRAGA, R. M. M. **Desenvolvimento Baseado em Componentes**. Proc. of the nd XIV SBES, 2000, João Pessoa. João Pessoa: SBC, 2000. p. 297-329.

WfMC - WORKFLOW MANAGEMENT COALITION. **Workflow: An Introduction**. Disponível em: http://www.wfmc.org/standards/docs/Workflow_An_Introduction.pdf. Acesso em: 25 ago. 2001.

WfMC - WORKFLOW MANAGEMENT COALITION. **Workflow Reference Model**. Document number TC00-1003, January 19, 1995. 55 p.

WFMC - WORKFLOW MANAGEMENT COALITION. **Terminology & Glossary**. Bruxelas, Jun. 1999. 52p.