UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Uma Proposta de Arquitetura de Linha de Produto para Sistemas de Gerenciamento de Workflow

por

FABRÍCIO RICARDO LAZILHA

Dissertação de mestrado submetida à avaliação
como requisito parcial para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Roberto Tom Price
Orientador

Profa. Dra. Itana Maria de Souza Gimenes
Co-Orientadora

Porto Alegre, setembro de 2002

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Lazilha, Fabrício Ricardo

Uma Proposta de Arquitetura de Linha de Produto para Sistemas de Gerenciamento de *Workflow* / por Fabrício Ricardo Lazilha. – Porto Alegre: PPGC da UFRGS, 2002.

119 f.:il.

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2002. Orientador: Price, Roberto Tom; Co-Orientadora: Gimenes, Itana Maria de Souza.

1. Arquitetura de linha de produto. 2. *Workflow*. 3. Arquitetura de software. 4. Reutilização. I. Price, Roberto Tom. II. Gimenes, Itana Maria de Souza. III. Título.

Agradecimentos

Primeiramente a Deus, que sempre esteve presente em minha vida, ajudando-me em todos os momentos, auxiliando minhas decisões, sempre me direcionando ao caminho certo a ser seguido.

A toda minha família, principalmente minha esposa e minha filha, pela paciência, amor, carinho e por entenderem (às vezes) minhas ausências. Aos meus pais por sempre me incentivarem a buscar o conhecimento.

Aos meus professores orientadores Roberto Tom Price e Itana Maria de Souza Gimenes, pela paciência, amizade e dedicação. Agradecimento especial à Prof. Itana que, mesmo em período de licença maternidade me atendeu em sua residência e sempre compreendeu e respeitou minhas limitações.

A todos os professores do Instituto de Informática da Universidade Federal do Rio Grande do Sul, especialmente os professores Carlos Alberto Heuser e José Palazzo Moreira de Oliveira pelas contribuições fornecidas ao trabalho.

Aos professores Guilherme Horta Travassos e Cláudia Werner, da Universidade Federal do Rio de Janeiro, pelas contribuições.

A professora Leonor Barroca, da Open University de Londres, pelas contribuições.

Ao Centro Universitário de Maringá (CESUMAR), em especial ao Magnífico Reitor Prof. Wilson de Matos Silva pelo apoio e compreensão. Aos meus colegas de sala, Márcia Pascutti e Robinson Patroni por toda a ajuda dispensada e amizade. Ao Prof. Valdecir Bertoncello por acreditar em mim e também pela amizade. Ao amigo Evandro Campitelli que se prontificou em resolver os problemas da instalação das ferramentas de validação.

Ao Grupo de Engenharia de Software da Universidade Estadual de Maringá (UEM), pela ajuda e apoio no período em que convivemos juntos. Agradecimento mais do que especial à Edna Takano, pela ajuda, paciência e disposição.

A Humberto Gessinger pela inspiração e conforto através de suas canções.

A todos aqueles que de forma direta ou indireta, influenciaram na realização deste trabalho.

Sumário

Lista de Abreviaturas **6**
Lista de Figuras **7**
Lista de Tabelas **8**
Resumo **9**
Abstract **10**
1 Introdução **11**
1.1 Motivação e objetivos do trabalho **12**
1.2 Organização do texto **13**
2 Sistemas Gerenciadores de *Workflow* **14**
2.1 Conceitos Básicos **14**
2.2 Caracterização de Workflows **18**
2.3 Modelagem de Sistemas Gerenciadores de Workflow **19**
2.4 Arquitetura Genérica para WfMSs **20**
2.5 O Modelo de Referência para WfMSs **22**
2.5.1 O Serviço de Execução do Workflow (Workflow Enactment Service) **22**
2.5.2 Ferramentas de Definição de Processo **23**
2.5.3 Aplicações de Workflow para Clientes **24**
2.5.4 Aplicações Invocadas **25**
2.5.5 Interoperabilidade entre Serviços de Execução de Workflow **25**
2.5.6 Ferramentas de Administração e Supervisão **25**
2.6 Considerações Finais **26**
3 Linha de Produto de Software **27**
3.1 Conceitos Básicos **27**
3.2 Elementos de uma Linha de Produto de Software **28**
3.2.1 Variabilidade **30**
3.2.2 Desenvolvimento do Núcleo de Artefatos **33**
3.2.3 Desenvolvimento do Produto **34**
3.2.4 Gerenciamento do Produto **35**
3.3 Arquitetura de Software **36**
3.4 Desenvolvimento Baseado em Componentes **37**
3.4.1 Conceitos Básicos **38**
3.4.2 O método Catalysis [D'SO 99] **41**
3.5 Descrição de Arquitetura de Software **42**

3.5.1 Introdução e Conceitos Básicos **42**
3.5.2 Linguagens para Descrição de Arquiteturas (ADLs) **43**
3.5.3 A Linguagem Rapide **44**
3.5.4 Outras ADLs Disponíveis **52**
3.6 Abordagens Existentes de Linha de produto 53
3.6.1 A Abordagem Synthesis **54**
3.6.2 FAST: Family-Oriented Abstraction, Specification and Translation **55**
3.6.3 PulSE: Product Line Software Engineering **57**
3.6.4 FODA: Feature Oriented Domain Analysis **58**
3.6.5 A Iniciativa PLP (Product Line Practice) do SEI/CMU **59**
3.6.6 A Abordagem Bosch **60**
3.6.7 Outras Abordagens **63**
3.7 Considerações Finais 64
4 Uma Arquitetura de Linha de Produto para Sistemas de Gerenciamento de *Workflow* 65
4.1 Extensões Necessárias para Representar Variabilidade 65
4.2 Análise de Requisitos 67
4.3 Especificação do Sistema 69
4.4 Projeto da Arquitetura 71
4.4.1 Arquitetura de Componentes **76**
4.4.2 Arquitetura de Componentes com CORBA **82**
4.5 Validação da Arquitetura Proposta 83
4.5.1 Dificuldades Encontradas **90**
4.6 Considerações Finais 91
5 Conclusões e Trabalhos Futuros 92
Anexo 1 Código-Fonte da Simulação 95
Anexo 2 Artigo aceito para publicação no XVI Simpósio Brasileiro de Engenharia de Software (SBES 2002), 16-18 outubro, 2002, Gramado, RS 100
Bibliografia 115

Lista de Abreviaturas

ADL	Architecture Description Language
CMM	Capability Maturity Model
CORBA	Common Object Request Broker
COTS	Commercial Of The Shelf
DBC	Desenvolvimento Baseado e Componentes
DCOM	Distributed Component Object Model
ExPSEE	Experimental Process-Centered Software Engineering Environment
FAST	Family-Oriented Abstraction, Specification and Translation
FODA	Feature-Oriented Domain Analysis
HAD	Heterogêneo, Assíncrono, Distribuído
IDL	Interface Definion Languagem
KLOC	Lines of Code
OCL	Object Constraint Language
OMG	Object Management Group
OO	Orientação a Objetos
ORB	Object Request Broker
Pov	Parcial Order Viewer
PSEE	Process Software Engineering Environment
PuLSE	Product Line Software Engineering
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RTS	Runtime System
SEI	Software Engineering Institute
STARS	Software Technology for Adaptables, Reliable Systems
UML	Unified Modeling Language
WAPI	Workflow Application Programming Interface
WfMC	Workflow Management Coalition
WfMS	Workflow Management System
SQL	Structured Query Language

Lista de Figuras

FIGURA 2.1 – Relação da terminologia básica associada com *workflow*. 15
FIGURA 2.2 – Fases do desenvolvimento de sistemas de *workflow*. 20
FIGURA 2.3 – Arquitetura genérica de um sistema de *workflow*. 21
FIGURA 2.4 – Modelo de referência (componentes e interfaces) 22
FIGURA 3.1 – O processo de desenvolvimento de uma linha de produto. 29
FIGURA 3.2 – Uma arquitetura Rapide e seu sistema. 45
FIGURA 3.3 – Análise da Ferramenta 48
FIGURA 3.4 – Exemplo de uma simples arquitetura de sistema 49
FIGURA 3.5 – Animação da arquitetura de sistema Application Program/Resource ... 50
FIGURA 3.6 – Poset Browser 51
FIGURA 3.7 – Código-fonte gerado a partir de uma arquitetura. 52
FIGURA 3.8 – Estrutura do Processo de Análise de Características comuns de FAST 56
FIGURA 3.9 – Artefatos de uma linha de produto de Software 62
FIGURA 4.1 – Exemplo de representação de variabilidade em casos de uso 66
FIGURA 4.2 – Exemplo de representação de variabilidade, estendendo a notação UML 66
FIGURA 4.3 – Diagrama de Caso de Uso do Gerente de Arquitetura de *Workflow*. 68
FIGURA 4.4 – Diagrama de Caso de Uso do Gerente de *Workflow* 68
FIGURA 4.5 – Diagrama de Caso de Uso do Usuário do *Workflow*. 69
FIGURA 4.6 – Diagrama Estático de Tipos para Sistemas de Gerenciamento de *Workflow* 70
FIGURA 4.7 – Diagrama de Colaboração de Alto Nível para WfMS 72
FIGURA 4.8 – Diagrama de Camadas Verticais de Alto Nível. 73
FIGURA 4.9 – Diagrama de Camadas Verticais 75
FIGURA 4.10 – Arquitetura de Componentes para WfMS. 76
FIGURA 4.11 – Diagrama de Seqüência/Gerente de Arquitetura de *Workflow*. 77
FIGURA 4.12 – Diagrama de Seqüência/Gerente de *Workflow*. 78
FIGURA 4.13 – Diagrama de Seqüência/Usuário do *Workflow* 79
FIGURA 4.14 – Arquitetura de Linha de produto para WfMS proposta. 83
FIGURA 4.15 – Arquitetura de linha de produto proposta modelada na ferramenta *Raparch*. 85
FIGURA 4.16 – Animação da arquitetura de linha de produto proposta. 86
FIGURA 4.17 – Código Rapide do Componente Gerenciador de Tarefas 89

Lista de Tabelas

TABELA 3.1 – Mecanismos de Variabilidade..... 32
TABELA 3.2 – Visão Acadêmica versus a Visão Industrial de Componentes de Software..... 40
TABELA 4.1 – Seqüência de eventos do componente GERENCIADOR_TAREFAS. 89

Resumo

A tecnologia de *workflow* vem apresentando um grande crescimento nos últimos anos. Os *Workflow Management Systems* (WfMS) ou Sistemas de Gerenciamento de *Workflow* oferecem uma abordagem sistemática para uniformizar, automatizar e gerenciar os processos de negócios. Esta tecnologia requer técnicas de engenharia de software que facilitem a construção desse tipo de sistema.

Há muito vem se formando uma consciência em engenharia de software de que para a obtenção de produtos com alta qualidade e que sejam economicamente viáveis torna-se necessário um conjunto sistemático 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. Muitas técnicas que favorecem a reutilização têm sido propostas ao longo dos últimos anos. Entre estas técnicas estão: engenharia de domínio, *frameworks*, padrões, arquitetura de software e desenvolvimento baseado em componentes. Porém, o que falta nesse contexto é uma maneira sistemática e previsível de realizar a reutilização. Assim, o enfoque de linha de produto de software surge como uma proposta sistemática de desenvolvimento de software, baseada em uma família de produtos que compartilham um conjunto gerenciado de características entre seus principais artefatos. Estes artefatos incluem uma arquitetura base e um conjunto de componentes comuns para preencher esta arquitetura. O projeto de uma arquitetura para uma família de produtos deve considerar as semelhanças e variabilidades entre os produtos desta família.

Esta dissertação apresenta uma proposta de arquitetura de linha de produto para sistemas de gerenciamento de *workflow*. Esta arquitetura pode ser usada para facilitar o processo de produção de diferentes sistemas de gerenciamento de *workflow* que possuem características comuns, mas que também possuam aspectos diferentes de acordo com as necessidades da indústria.

O desenvolvimento da arquitetura proposta tomou como base a arquitetura genérica e o modelo de referência da *Workflow Management Coalition* (WfMC) e o padrão de arquitetura *Process Manager* desenvolvido no contexto do projeto ExPSEE[1]. O processo de desenvolvimento da arquitetura seguiu o processo sugerido pelo Catalysis com algumas modificações para representar variabilidade. A arquitetura proposta foi descrita e simulada através da ADL (*Architecture Description Language*) Rapide.

A principal contribuição deste trabalho é uma arquitetura de linha de produto para sistemas de gerenciamento de *workflow*. Pode-se destacar também contribuições para uma proposta de sistematização de um processo de desenvolvimento de arquitetura de linha de produto e também um melhor entendimento dos conceitos e abordagens relacionados à prática de linha de produto, uma vez que esta tecnologia é recente e vem sendo largamente aplicada nas empresas.

Palavras Chaves: *workflow*, sistemas de gerenciamento de *workflow*, arquitetura de linha de produto, reutilização, arquitetura de software.

[1] ExPSEE – Experimental Process-Centred Software Engineering Environment, é um projeto desenvolvido no Laboratório de Engenharia de Software do Departamento de Informática da Universidade Estadual de Maringá, financiado pelo CNPq de 1994 a 2002.

TITLE: "A PROPOSAL PRODUCT LINE ARCHITECTURE FOR WORKFLOW MANAGEMENT SYSTEMS"

Abstract

The workflow technology is rapidly evolving. Workflow Management Systems (WfMS) provide a systematic approach to normalize, automate and manage business processes. This technology requires software engineering techniques that make the production of WfMS easier.

The software engineering community is already aware that in order to produce high quality and economically feasible software it is mandatory to follow a set of systematic processes, techniques and tools. Reuse is amongst the most relevant of these techniques. It is considered that by reusing well specified and tested software components it is possible to build systems faster as well as increasing reliability. Several techniques have been proposed which contributes to this approach such as domain engineering, frameworks, patterns, software architecture and component based development. However, there is still a lack of a systematic way of applying software reuse. Thus, the product line approach proposes a systematic method to develop software systems based on a product family of which its artifacts share a set of manageable features. These artifacts include a base architecture and a set of commons components. The design of an architecture to a family of products must consider its commonalities and variabilities.

This work proposes a product line architecture for workflow management systems. This architecture is used to facilitate the process of production of different WfMS that has several common features but also have different aspects according to industry needs.

The development of the proposed architecture was based on the generic architecture and reference model of the Workflow Management Coaliation. Is has also considered the Process Manager pattern produced within the ExPSEE[2] project. The architecture development followed the Catalysis process extended with variability representation. The proposed architecture was specified simulated using the ADL (Architecture Description Language) Rapide.

The main contribution of this work is a product line architecture for WfMS. The work also contributes by offering a systematic process to develop product line architecture based on a conventional component based method. For the product line area of research this work also contributes to the understanding and dissemination of concepts and techniques in this area.

Keywords: workflow, workflow management system, product line architecture, reuse, software architecture.

[2] ExPSEE – Experimental Process-Centred Software Engineering Environment, is a project developed in the Laboratory of Software Engineering of the Computing Department of Maringá State University, founded by CNPq, 1994 -2002.

1 Introdução

Uma arquitetura representa um investimento significativo para uma organização. Em função disto, é natural que essas organizações desejem maximizar este investimento pela reutilização da arquitetura em vários sistemas. Quando uma organização está produzindo vários sistemas similares e reutilizando a mesma arquitetura e os componentes associados a esta, pode se beneficiar de algumas vantagens substanciais como a redução dos custos de produção e do tempo para disponibilizar este produto no mercado. Segundo Bass [BAS 98], esta é a idéia principal da abordagem denominada de linha de produto de software: uma coleção de sistemas que compartilham um conjunto gerenciado de características construído de um conjunto dos principais artefatos[3] de software já desenvolvidos. Esses artefatos incluem a arquitetura base e um conjunto comum de componentes para preencher essa arquitetura.

A reutilização de software tem sido comentada há muito tempo pela comunidade de engenharia de software. Já no final dos anos 60, a noção de construção de sistemas pela composição de componentes foi discutida como a mais promissora abordagem para tratar a crise do software. Durante os anos 70, várias abordagens baseadas em módulos para alcançar reutilização foram propostas, e durante os anos 80, a programação orientada a objetos propunha o uso de classes como unidades reutilizáveis. Porém, todos esses esforços fornecem reutilização somente em nível individual, freqüentemente em pequena escala, em que componentes são usados como blocos na construção de novas aplicações [BAS 98].

Uma linha de produto de software envolve um conjunto de aplicações similares dentro de um domínio que podem ser desenvolvidas a partir de uma arquitetura genérica comum, a arquitetura de linha de produto, e um conjunto de componentes que povoam a arquitetura. Esta abordagem tem por objetivo identificar os aspectos comuns e as diferenças entre os artefatos de software ao longo do processo de desenvolvimento da linha de produto, de modo a explicitar os pontos de decisão em que a adaptação dos componentes para a geração de produtos específicos pode ser realizada. Para tal, durante o processo de desenvolvimento, deve-se identificar os pontos de variabilidade que são pontos em que as características dos produtos podem se diferenciar. Esses pontos de variabilidade aparecem inicialmente na definição dos requisitos da linha de produto e devem ser representados ao longo de todo o processo de desenvolvimento. Dessa forma, pode-se decidir por uma característica ou outra para um produto específico tanto em nível de projeto quanto em nível de implementação.

Dentre os artefatos de uma linha de produto, a arquitetura de software tem um papel importante. Ela representa a infra-estrutura central da linha de produto. A arquitetura de um sistema de software engloba a definição de suas estruturas gerais, descrevendo os elementos que compõem os sistemas e as interações entre estes [BAS 98].

Uma razão para a definição explícita da arquitetura de software, é que ela especifica um conjunto de componentes reutilizáveis na linha de produto de software. A arquitetura de software para um produto específico pode ser derivada da arquitetura de

[3] A expressão *artefato de software* é usada neste trabalho de forma genérica, não se referindo necessariamente a código, podendo abranger qualquer produto intermediário gerado ou utilizado ao longo do processo de software.

linha de produto. A arquitetura de linha de produto e a arquitetura de um produto específico são equivalentes. Dessa forma, componentes compartilhados com a linha de produto podem ser usados para satisfazer os requisitos dos produtos. Esses componentes compartilhados são instanciados e configurados para alcançar os requisitos de um produto específico [BOS 2000].

Diferente da visão popular da programação orientada a componentes, os componentes de uma arquitetura de linha de produto são grandes pedaços de software com mais de cem mil linhas de código. Esses componentes não são tipicamente modelados como componentes caixa-preta, e sim como *frameworks* orientados a objetos, que possuem funcionalidades comuns para os produtos na linha de produto e suportam variabilidade requerida para os vários produtos. Este tipo de arquitetura apresenta uma importante abordagem para aumentar a reutilização de software e reduzir o custo do desenvolvimento de software e a necessidade de manutenção desses produtos por parte das companhias de software [BOS 98].

Um *workflow* é definido como uma coleção de tarefas organizadas para realizar um processo, quase sempre de negócio. Essas tarefas podem ser executadas por um ou mais sistemas de computador, por um ou mais agentes humanos, ou então por uma combinação destes. A ordem de execução e as condições pelas quais cada tarefa é iniciada também estão definidas no *workflow*, sendo que o mesmo é capaz ainda de representar a sincronização das tarefas e o fluxo de informações.

Nos últimos anos, muitas organizações têm se preocupado com a melhoria da qualidade de seus processos, com o aumento da produtividade e também com a redução dos custos de produção. Assim, o interesse pela tecnologia de *workflow* por parte dessas organizações tem crescido nos últimos anos. A tecnologia de *Workflow* permite a modelagem do processo de negócios, a reengenharia de processos especificados e a automação dos processos de negócios, que são formas de aumentar a produtividade, a qualidade e a criatividade dessas organizações.

Um Sistema de Gerenciamento de *Workflow* é um sistema que permite a definição, o gerenciamento e a execução de *workflows* [WMC 95]. Estes sistemas executam uma ou mais máquinas de *workflow* que são aptas para interpretar a definição dos processos, interagir com os participantes do *workflow* e invocar ferramentas ou aplicações quando necessário. Neste sentido, arquiteturas de software e arquiteturas de linha de produto podem facilitar a construção de sistemas de *workflow*.

1.1 Motivação e objetivos do trabalho

A abordagem de linha de produto é aplicável aos domínios em que existe uma demanda por produtos específicos, no entanto existe um conjunto de características comuns e pontos de variabilidade bem definidos. O domínio dos Sistemas de Gerenciamento de *Workflow* (WfMS) é altamente favorável à aplicação da abordagem de linha de produto, devido aos esforços da WfMC [WMC 95], que viabilizou a construção de uma arquitetura genérica para WfMS que pode ser ajustada à maioria dos produtos do mercado. A partir disso a WfMC estabeleceu um modelo de referência para WfMS. Cada implementação de um WfMS pode adaptar componentes ou interfaces de acordo com as necessidades da aplicação. Produtos de *workflow* com características similares, porém com diferentes especificidades são necessários nas mais diversas empresas que utilizam esta tecnologia.

Esta dissertação apresenta uma proposta de arquitetura de linha de produto para sistemas de gerenciamento de *workflow*. Esta arquitetura pode ser usada para facilitar o processo de produção de diferentes sistemas de gerenciamento de *workflow* que possuem características comuns, mas que também possuam aspectos diferentes de acordo com as necessidades da indústria.

O desenvolvimento da arquitetura tomou como base a arquitetura genérica e o modelo de referência da WfMC [WMC 95] [WMC 99] e o padrão de arquitetura *Process Manager* [GIM 99a] desenvolvido no contexto do projeto ExPSEE [GIM 99b]. O processo de desenvolvimento da arquitetura seguiu o processo sugerido pelo Catalysis [D'SO 99] com algumas modificações para representar variabilidade. Para fins de validação da arquitetura, a ADL Rapide [CSL 97] foi utilizada para simulação.

A principal contribuição deste trabalho é uma arquitetura de linha de produto para sistemas de gerenciamento de *workflow*. Pode-se destacar também contribuições para uma proposta de sistematização de um processo de desenvolvimento de arquitetura de linha de produto e também um melhor entendimento dos conceitos e abordagens relacionados à prática de linha de produto, uma vez que esta tecnologia é recente e vem sendo largamente aplicada nas empresas.

1.2 Organização do texto

O texto está organizado como segue. O capítulo 2 apresenta os principais conceitos relacionados a *workflow* e sistemas de gerenciamento de *workflow*.

O capítulo 3 apresenta conceitos de linha de produto de software, arquitetura de software, desenvolvimento baseado em componentes e descrição de arquiteturas de software. Apresenta ainda uma descrição de algumas abordagens de linha de produtos existentes.

O capítulo 4 apresenta a arquitetura de linha de produto para sistemas de gerenciamento de *workflow*, bem como o processo seguido para o projeto dessa arquitetura. Apresenta também as extensões utilizadas para representar variabilidade no processo de desenvolvimento e a validação da arquitetura proposta em uma ADL.

No capítulo 5 são apresentados as conclusões e os trabalhos futuros que podem ser realizados a partir deste esforço.

O anexo 1 apresenta o código Rapide da simulação da arquitetura proposta.

O anexo 2 apresenta o artigo aceito para publicação do XVI Simpósio Brasileiro de Engenharia de Software (SBES) que acontecerá entre os dias 15 e 18 de outubro de 2002 em Gramado, RS.

2 Sistemas Gerenciadores de *Workflow*

Este capítulo apresenta uma introdução à tecnologia de *workflow,* visando destacar os principais conceitos. Serão definidos os conceitos de *workflow* baseados nos padrões propostos pela WfMC. A WfMC é uma organização internacional formada em 1993 que tem como objetivo promover a área de *workflow* através da divulgação da tecnologia e do desenvolvimento de padrões para a interoperabilidade de sistemas de *workflow*. Serão definidos também os principais conceitos relacionados a Sistemas Gerenciadores de *Workflow* (WfMSs), bem como sua arquitetura genérica e o modelo de referência propostos pela WfMC.

2.1 Conceitos Básicos

As tecnologias de *workflow* e gerenciamento de *workflow* estão em contínuo crescimento e vêm sendo exploradas cada vez mais em diversos domínios de negócios diferentes. Muitos produtos de gerenciamento de *workflow* estão disponíveis no mercado, cada um deles dando ênfase em suas características particulares. Dentro destas características aparecem: produtos desenvolvidos para diferentes plataformas de hardware, produtos com funcionamento centralizado ou distribuído e ainda funcionalidades específicas de cada implementação [TAN 2000a].

Existem várias definições de *workflow* e WfMS na literatura [GEO 95] [CAS 95] [ULT 97], porém existe uma enorme semelhança entre esses conceitos. Com base nas definições da WfMC [WMC 95], podemos definir *workflow* como sendo uma coleção de atividades organizadas para realizar um processo de negócios, por exemplo para processar um pedido de compra ou tomar uma decisão. 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 do grupo.

Um sistema gerenciador de *workflow* fornece os mecanismos para a automação dos processos de negócios através do gerenciamento das atividades de trabalho, além da invocação dos recursos humanos ou de máquinas associados com os vários passos de cada atividade. Assim, um WfMS é um sistema computacional que permite a definição, a gerência e a execução completa de *workflows* [WMC 95]. Esses sistemas executam em uma ou mais máquinas de *workflow* que são aptas a interpretar a definição dos processos, interagir com os participantes do *workflow* e quando necessário, invocar ferramentas ou aplicações. Existem várias técnicas de implementação e vários ambientes operacionais para um WfMS.

Em um nível mais alto de abstração, pode-se observar que um WfMS fornece apoio em três áreas funcionais que são: funções em tempo de construção, funções de controle em tempo de execução e funções de interação em tempo de execução. As funções em tempo de construção tratam da definição do processo de *workflow* e suas atividades constituintes.

As funções de controle em tempo de execução tratam do gerenciamento do *workflow* em um ambiente operacional e do seqüenciamento das várias atividades a serem tratadas como parte de cada processo. Essas funções controlam instâncias de

processo, agendamento de atividades, invocação de aplicações externas, entre outras. Por fim, as funções de interação em tempo de execução lidam com a interação com usuários humanos e aplicações de tecnologia da informação para o processamento das várias etapas de uma atividade.

A principal característica de uma infra-estrutura de execução de *workflow* é a habilidade de distribuir tarefas e informações entre participantes. Essa distribuição pode ocorrer dentro de grupos de trabalho ou até mesmo entre organizações inteiras, podendo utilizar para isso mecanismo de suporte à comunicação como correio eletrônico, passagem de mensagens e até tecnologias de objetos distribuídos. É necessário, então, a presença de uma estrutura central que possua interfaces para usuários, aplicações e outros serviços de execução [KAS 2000].

Para padronizar as especificações de produtos de *workflow*, a WfMC propôs um modelo de referência para sistemas gerenciadores de *workflow* [WMC 95]. Este modelo contém a identificação das principais características, a terminologia, principais componentes e interfaces dos WfMSs. O modelo de referência procura identificar o fluxo de informação entre os diversos componentes, além de apresentar diversos cenários de interoperabilidade.

A tecnologia de *workflow* envolve muitos conceitos inter-relacionados. A Figura 2.1 mostra como esses conceitos se relacionam.

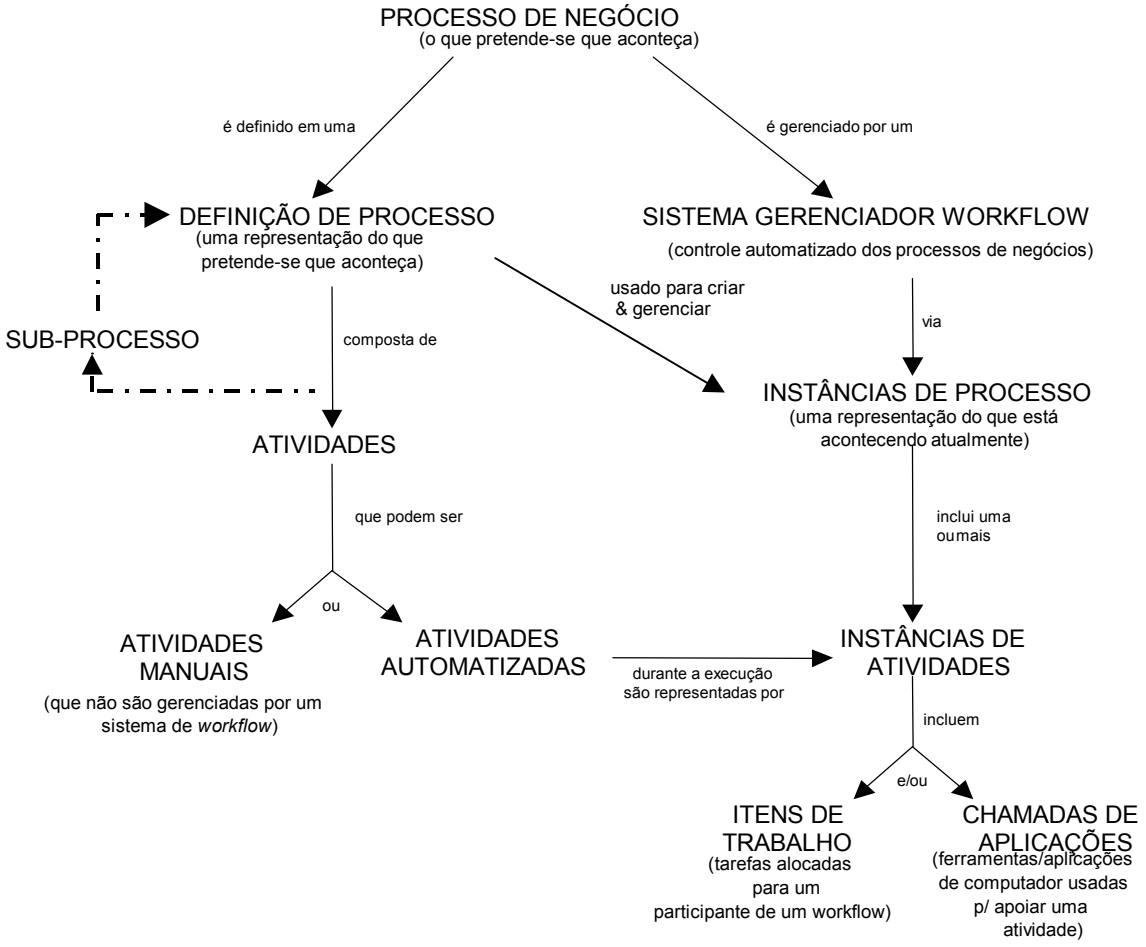

FIGURA 2.1 – Relação da terminologia básica associada com *workflow* [WMC 95].

A seguir são descritos os conceitos básicos relacionados à terminologia utilizada na Figura 2.1, associados com a tecnologia de *workflow* [WMC 95].

- **Processo de Negócios**

Processo de negócios, ou simplesmente processo, é um conjunto de um ou mais procedimentos ou atividades relacionadas, que coletivamente atinge um objetivo de negócios, normalmente dentro do contexto de uma estrutura organizacional, que define cargos/funções e relações. Os processos podem variar significativamente em aspectos como tempo de duração (desde meses até minutos) e abrangência dentro das organizações (desde um único setor até diferentes setores em uma organização). As atividades englobadas nos processos podem variar de programas de computador até atividades humanas, por exemplo, reuniões e tomadas de decisão. Um processo é definido em uma definição de processo.

- **Definição de Processo**

A definição de um processo é a representação de um processo de modo a permitir manipulações automatizadas, como simulação, ou execução por um sistema de gerenciamento de *workflow*. Uma definição de processo consiste de especificações de uma rede de atividades, seus relacionamentos, critérios para indicar o início e o término dos processos e informações sobre as atividades em si, como por exemplo, participantes, aplicações utilizadas e dados de sistemas de informações relacionados.

- **Sub-Processo**

Um processo que é executado ou chamado de outro processo hierarquicamente superior (ou sub-processo), e que faz parte do processo global. Múltiplos níveis de sub-processos podem ser definidos. Um sub-processo é útil para definir unidades reutilizáveis dentro de outros processos. Cada sub-processo tem sua própria definição de processo.

- **Atividade**

Atividade é a descrição de um conjunto de itens de trabalho que realiza um passo lógico dentro de um processo. Uma atividade pode ser manual, ou automatizada. Uma atividade de *workflow* pode requerer recursos humanos e/ou computacionais para ser executada.

- **Instância de Processo**

É a representação de uma única execução de um processo. Uma instância de processo é criada, gerenciada, e (eventualmente) terminada por um sistema gerenciador de *workflow*, conforme sua definição de processo. Cada instância de processo exibe um estado interno que representa a evolução do processo e de suas atividades.

- **Instância de Atividade**

É a representação de uma atividade em uma (única) execução de um processo. Uma instância de atividade é criada e gerenciada por um sistema gerenciador de *workflow*, conforme a definição de processo.

- **Item de Trabalho**

Do ponto de vista do participante do *workflow*, as atividades são representadas como uma coleção de itens de trabalho. Cada item de trabalho (tarefa) representa uma partição de uma atividade, como a revisão de um documento ou o preenchimento de um formulário.

- **Chamada de Aplicação**

É a invocação de uma aplicação de *workflow* pelo sistema gerenciador de *workflow* para automatizar, completamente ou parcialmente, uma tarefa, ou ajudar um participante de *workflow* no processamento de um item de trabalho. A aplicação pode ser chamada diretamente pelo sistema gerenciador de *workflow* ou pode ser chamada indiretamente por um ator da aplicação.

- **Ator**

Um ator é responsável pela execução parcial ou total de uma determinada instância de atividade. Um ator pode tanto ser humano (por exemplo, um funcionário), quanto um software (por exemplo, um software para envio de fax).

Segundo Casati [CAS 96], na execução de um *workflow,* tem-se os seguintes tipos de atores:

  - **executor do *workflow***: é o ator que inicia (dispara) o *workflow*. Em geral, o *workflow* possui uma lista de possíveis executores;
  - **responsável**: é o ator que tem responsabilidade sobre o *workflow*. Pode ser tanto o executor, como qualquer outra pessoa pertencente à organização;
  - **executor da atividade**: é o responsável pela execução das atividades.

- **Cargo/Função**

Cargo/Função é um conjunto de atributos associados aos participantes de um *workflow*. Por exemplo, um processo de desenvolvimento de software poderia ter cargos como, ‘analista de sistemas’ e ‘programador’.

Dependendo da organização na qual o *workflow* está sendo executado, para cada atividade pode existir um ou mais atores aptos para sua execução. Nem sempre é possível ou desejável definir previamente o responsável pela execução de uma atividade. Sendo assim, utiliza-se o conceito de cargo/função, que nada mais é do que a representação de um conjunto de atores que possuem um mesmo leque de características (habilidades) que os tornam aptos a executarem a atividade relacionada ao cargo/função

[TAN 2000a]. Desta forma, ao definir-se um *workflow*, ao invés de associar um ator (pessoa) à atividade, associa-se um cargo/função. Outro ponto a ser destacado, é o fato de um mesmo ator (pessoa) poder executar mais de um cargo/função.

A seção seguinte apresenta uma caracterização de *workflows* sobre a ótica de negócios.

2.2 Caracterização de *Workflows*

Ainda não existe uma caracterização ou categorização padrão para *workflow* [GEO 95]. Sob a ótica de negócios, pode-se caracterizar um *workflows* conforme descrito abaixo.

- ***Workflow* Ad-hoc**

Neste tipo de *workflow* não existe uma estrutura pré-definida para o processo, ou esta estrutura pode ser modificada em tempo de execução. As tarefas envolvem tipicamente coordenação, colaboração e co-decisão humana. A coordenação e a ordenação das tarefas são feitas manualmente. Este tipo de *workflow* é indicado para representar aplicações *groupware*, nas quais vários usuários compartilham uma mesma base de dados e colaboram para a realização de tarefas específicas.

- ***Workflow* de Produção**

Envolve processos repetitivos e previsíveis. Tipicamente envolve processos complexos que possuem uma estrutura fixa e um conjunto de regras de roteamento entre as atividades. Normalmente acessam múltiplos sistemas de informação para realizar um trabalho e recuperar dados utilizados em tomadas de decisão. Este tipo de *workflow* é usado para monitorar o estado e a localização dos processos e documentos manipulados pelo sistema a cada instante de tempo. A ordenação e a coordenação das tarefas podem ser automatizadas.

- ***Workflow* Administrativo**

Possuem diversas características dos *workflows* de produção (estruturados), porém são direcionados para atividades administrativas internas da organização. *Workflows* administrativos geralmente não são de missão crítica, visto que são menos exigentes em relação a confiabilidade, correção e integração com sistemas externos do que os *workflows* de produção. Como exemplo pode-se citar: avaliação de trabalhos submetidos a um congresso onde já se conhece previamente os revisores.

- **Workflows Orientado a Pessoas e Orientado a Sistemas**

Georgakopoulos [GEO 95] apresenta uma outra classificação para w*orkflow*s, caracterizando-os dentro de dois aspectos: orientado a pessoas e orientado a sistemas. Em W*orkflows* orientados a pessoas, os atores cooperam com o WfMS coordenando e executando tarefas, assegurando assim a consistência dos resultados do *workflow*. Neste tipo de *workflow* não se tem o conhecimento real da informação que é processada, então

o WfMS não pode ser responsável pela consistência da informação. Em um *workflow* orientado a pessoas, as principais questões a serem analisadas são:

- a interação homem-máquina;
- existência de habilidades humanas para atender as exigências das tarefas;
- a cultura do escritório, o que determina como as pessoas preferem ou necessitam trabalhar.

Já os *workflow*s orientados a sistemas são altamente automatizados e executam operações intensivas de computação e software especializado em tarefas que normalmente tem acesso a sistemas de informação HAD (Heterogêneo, Autônomo, Distribuído). WfMSs devem ter a capacidade de coordenar tarefas de software e fornecer controle de concorrência e mecanismos de recuperação para garantir a consistência e a confiabilidade. *Workflow* orientados a sistemas têm conhecimento da semântica de informação, e podem ser responsável por manter consistências de informação. Em um *workflow* orientado a sistemas, as principais questões a serem analisadas são:

- acesso a sistemas HAD;
- determinar novas necessidades de software de forma a permitir automação dos processos de negócios;
- assegurar a execução correta e segura dos sistemas.

A seção seguinte apresenta uma breve caracterização sobre as etapas que envolvem o desenvolvimento de sistemas de gerenciamento de *workflow*.

2.3 Modelagem de Sistemas Gerenciadores de *Workflow*

Georgakopoulos [GEO 95] identifica as etapas que envolvem o desenvolvimento de sistemas gerenciadoresde *workflow,* conforme mostra a Figura 2.2. Estas etapas são descritas a seguir.

1. **modelagem de processos e definição de *workflow***: requer modelos de *workflow* e métodos para capturar o processo como uma definição de *workflow*. Assim, em um passo inicial, é necessário compreender o processo que se deseja modelar, o que tipicamente é feito através de entrevistas com os participantes do processo. Nesta etapa, pode-se utilizar técnicas de entrevistas similares às utilizadas para o projeto de sistemas de informações, como forma de se obter um conhecimento mais aprofundado da natureza dos processos. Quando conhecimento suficiente for adquirido, pode-se capturar o processo em uma definição de *workflow*. Para realizar a definição de um processo, é necessário a utilização de um modelo de *workflow.* Um modelo de *workflow* tipicamente inclui uma série de conceitos para descrever os processos, como os já vistos: atividades, dependências e cargos. O modelo é então escrito em uma linguagem de especificação.

2. **implementação e automação de *workflow***: requer métodos e técnicas para utilizar sistemas de computação e participantes humanos para implementar, escalonar, executar e controlar as atividades definidas no *workflow*.

3. **aplicação de *workflow*:** implementações automatizadas de *workflow* são tipicamente aplicações distribuídas que acessam sistemas HAD. Como qualquer outra aplicação distribuída HAD, um WfMS tem que tratar da integração da aplicação, interoperabilidade, correção e a confiabilidade da execução. As limitações de um WfMS para tratar essas questões, e as tecnologias de infraestrutura que podem ser usadas para tratar estas limitações são melhor discutidas em [GEO 95].

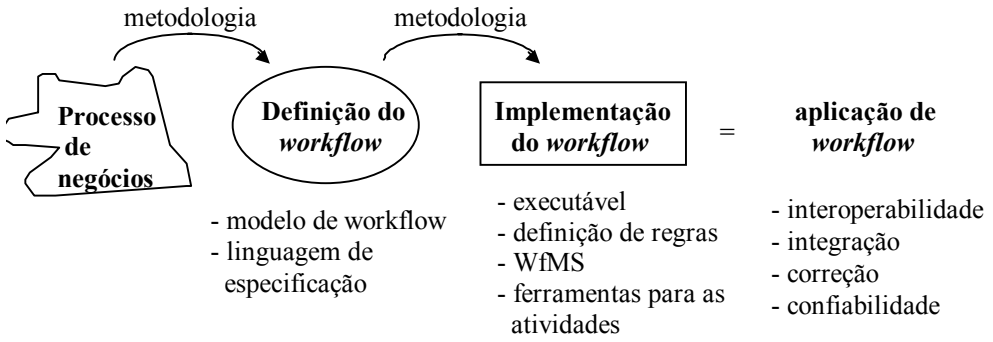

FIGURA 2.2 – Fases do desenvolvimento de sistemas de *workflow* [GEO 95].

Historicamente, muitos produtos já possuem algumas das funcionalidades de WfMSs. Entre esses produtos pode-se citar: processadores de imagens, gerenciadores de documentos, correio eletrônico, aplicações de *groupware*, aplicações baseadas em transação, *software* de suporte à projetos e ferramentas de projeto de sistemas.

A seção seguinte apresenta a descrição da arquitetura genérica para WfMS proposta pela WfMC.

2.4 Arquitetura Genérica para WfMSs

Mesmo com a variedade de produtos de *workflow* existentes, a WfMC, após uma série de análises e comparações entre os produtos disponíveis, identificou a viabilidade da construção de um modelo de implementação genérico de um sistema de *workflow* que pode ser ajustado à maioria dos produtos do mercado. Esse modelo identifica os principais componentes funcionais e suas respectivas interfaces dentro de um sistema de gerenciamento de *workflow*. A partir deste modelo cada implementação de um sistema gerenciador de *workflow* pode optar por adaptar componentes ou interfaces de acordo com as necessidades da aplicação[LAZ 2001]. A arquitetura genérica de um sistema de *workflow* é apresentada na Figura 2.3. Esta arquitetura é composta por três tipos de componentes:

- **componentes de software**: são os componentes que oferecem suporte as funcionalidades do WfMS, tais como a ferramenta de definição, a máquina de

workflow, o gerenciador da lista de trabalho e a interface do usuário;

- **dados de controle do sistema**: são dados utilizados pelos componentes de software, tais como os dados de controle de *workflow,* os dados relevantes do *workflow* e o modelo organizacional;

- **aplicativos e suas bases de dados**: são os aplicativos que não fazem parte dos WfMSs, mas podem ser invocados por ele como parte do sistema de *workflow*;

Cada um desses componentes que integram a arquitetura genérica dos WfMSs serão descritos na seção 2.5, que trata do modelo de referência para sistemas gerenciadores de *workflow* derivado desta arquitetura genérica.

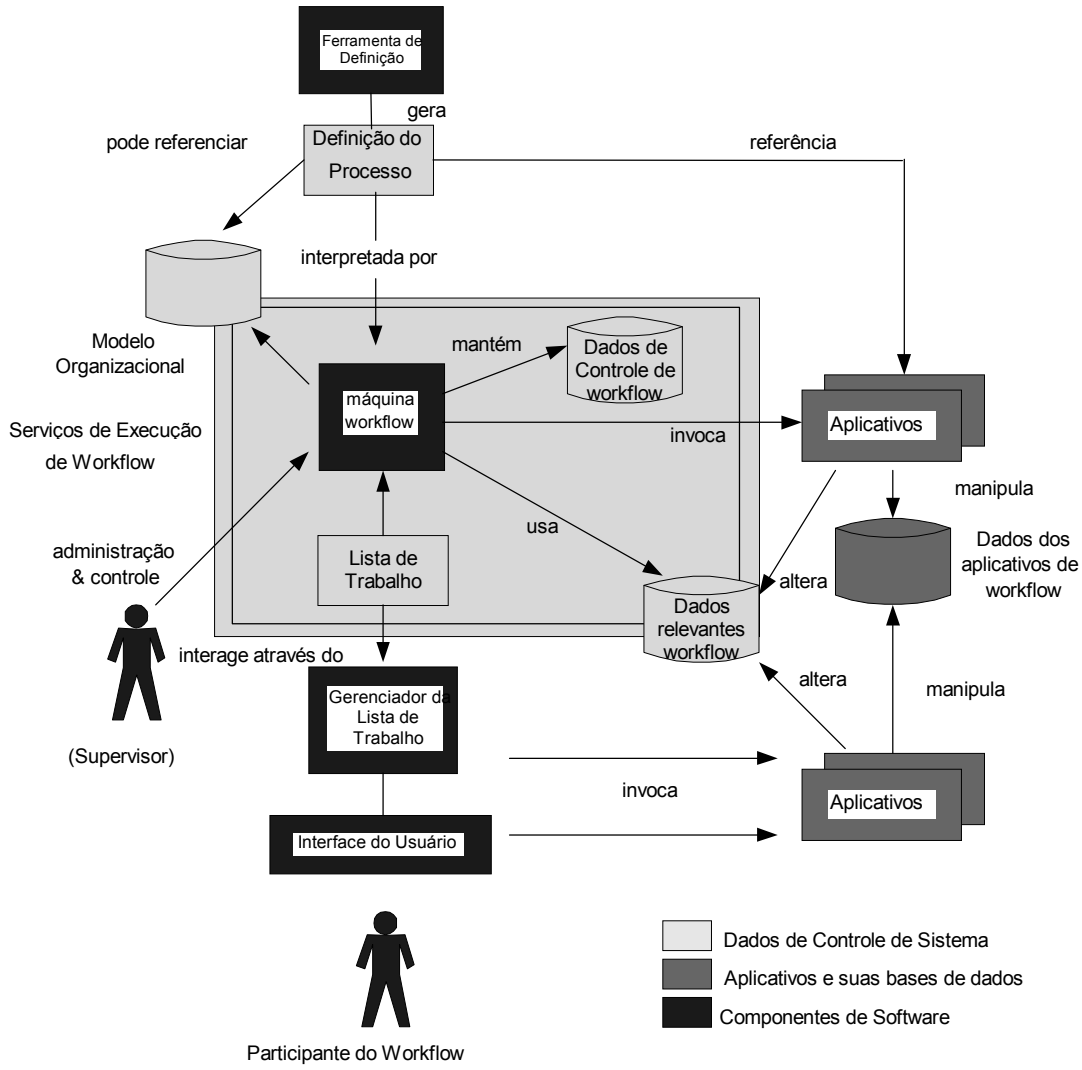

FIGURA 2.3 – Arquitetura genérica de um sistema de *workflow* [WMC 95].

A arquitetura genérica para WfMSs serviu como base para a definição do modelo de referência apresentado na seção seguinte.

2.5 O Modelo de Referência para WfMSs

A arquitetura genérica de aplicações de *workflow* descrita na seção anterior, serviu de suporte para o desenvolvimento e construção do modelo de referência para sistemas de gerenciamento de *workflow* [WMC 95]. Isso foi feito através da análise dos elementos que compõem a estrutura genérica dos WfMSs e da análise das interfaces existentes entre eles. Além da identificação de cada interface, identificou-se também o comportamento de cada uma delas.

Assim, um conjunto de interfaces padronizadas e formatos de intercâmbio de dados surgiram como resultado deste trabalho. A partir desse modelo, diferentes produtos de *workflow* podem alcançar variados níveis de interoperabilidade. Portanto, o modelo de referência inclui cinco padrões de interoperabilidade e comunicação que permitem a coexistência e interação de múltiplos produtos de *workflow* com o ambiente do usuário, como é mostrado na Figura 2.4.

As interfaces ao redor do serviço de execução do *workflow* são chamadas de *Workflow Application Programming Interface* (WAPI) e os formatos de intercâmbio de dados regulam e definem como o serviço de execução do *workflow* vai se comunicar com os outros componentes do sistema. As seções seguintes trazem uma discussão mais completa sobre cada uma das interfaces e componentes do modelo de referência.

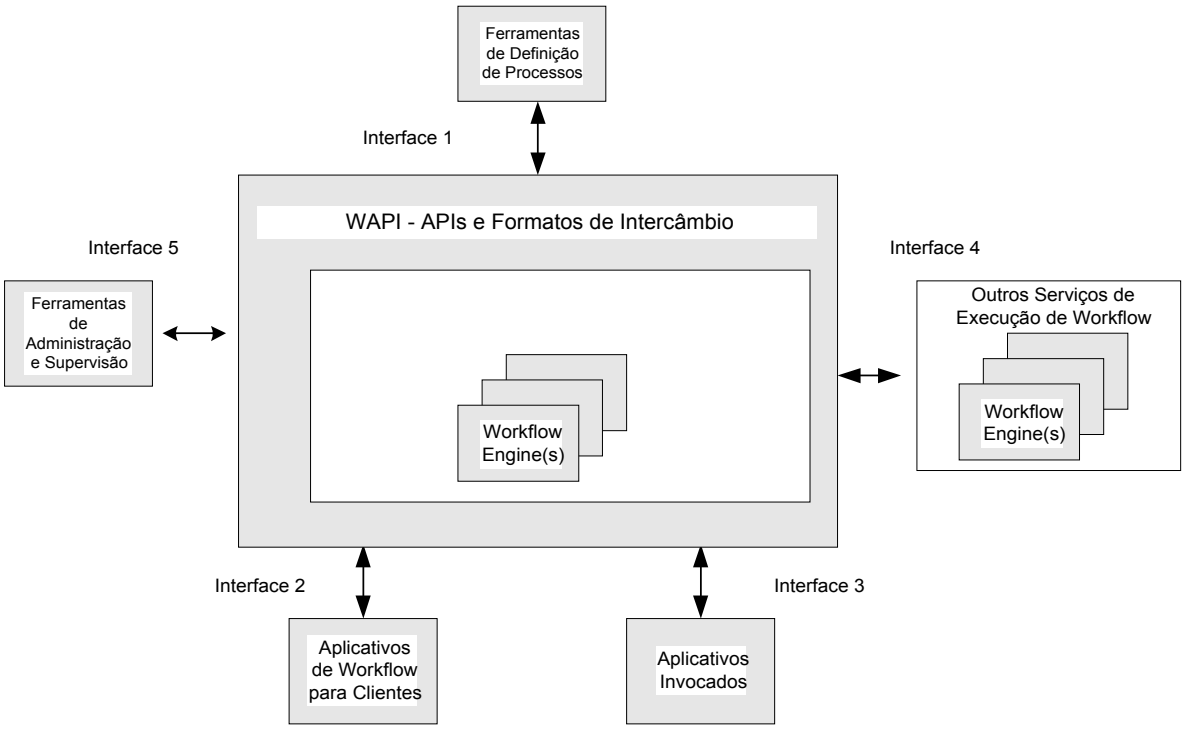

FIGURA 2.4 – Modelo de referência (componentes e interfaces) [WMC 99].

2.5.1 O Serviço de Execução do *Workflow* (*Workflow Enactment Service*)

O serviço de execução do *workflow* é quem fornece o ambiente e os mecanismos de execução nos quais instanciações e ativações de processo ocorrem, usando para isso uma ou mais máquinas de *workflow*, as quais são responsáveis pela interpretação e

ativação de parte, ou de toda a definição de processo, além de interagir com recursos externos necessários para a execução das atividades.

A função desse serviço é criar, gerenciar e executar instâncias de *workflow*. Aplicações externas devem acessar o serviço de execução via WAPI, que pode ser considerada o limite de interação do sistema de *workflow* com recursos e aplicações externas. Assim, a interação com o serviço de execução do *workflow* pode ocorrer via interface de aplicações de *workflow* para clientes (interface 2) ou via interface de aplicações invocadas (interface 3).

Ao invés de ser considerada apenas uma entidade centralizada, o serviço de execução de *workflow* pode ser também funcionalmente distribuído. Neste caso, as diversas máquinas de *workflow* (*workflow engines*) envolvidas devem fazer uso de protocolos específicos e formatos de intercâmbio para sincronizar suas operações e possibilitar o intercâmbio de processos, informações de controle de atividades e dados relevantes do *workflow*.

Uma máquina de *workflow* é responsável por partes, ou por todo, o controle do ambiente de execução dentro de um serviço de execução. É na verdade, um serviço de software ou uma espécie de "máquina" que fornece o ambiente de tempo de execução para uma instância de *workflow*. Neste caso, o processo deve ser particionado dentro do conjunto de máquinas existentes. As máquinas de *workflow* devem ter a capacidade de: interpretar uma definição de processo, controlar instâncias de processos, manipular atividades dentro de um processo, controlar participantes, identificar itens de trabalho associados a cada participante, tratar dados de controle e dados relevantes do *workflow*, invocar aplicações externas e permitir funções de administração do *workflow* [TAN 2000a] [LAZ 2001].

Alguns tipos principais de dados manipulados pelo serviço de execução do *workflow*:

- **dados de controle:** dados internos que são gerenciados pelo WfMS ou por uma máquina de *workflow*;

- **dados relevantes:** dados que são usados por um WfMS para determinar a transição de estado de uma instância de processo de *workflow*;

- **dados de aplicação:** dados específicos de aplicações que não são acessíveis ao WfMS.

2.5.2 Ferramentas de Definição de Processo

Uma variedade de diferentes ferramentas podem ser usadas para analisar e descrever o processo de negócio. Essas ferramentas podem ser manuais, como, por exemplo, papel e caneta ou podem ser dispositivos mais sofisticados e formais. Da mesma forma, elas podem ou não fazer parte do produto de *workflow*. Elas devem fornecer como resultado final a descrição do modelo de processo de negócios de uma determinada organização.

A definição de processo é interpretada, em tempo de execução, por uma ou mais máquinas de *workflow* dentro do serviço de execução de *workflow*. A transferência da

definição de processo feita em uma ferramenta de definição para um serviço de execução de *workflow* é feita através da interface 1. Esse intercâmbio é feito por meio de funções de importação e exportação e pode usar para isso, tanto recursos de transferência eletrônicos como físicos.

É importante neste contexto de definição de processos, o uso de padronizações, como é o caso das linguagens de definição de processos. Uma série de benefícios podem ser obtidos utilizando-se formas padronizadas de definição. Esse recurso vem sendo muito explorado pela WfMC.

Uma definição de processo deve conter no mínimo os seguintes atributos: nome do processo, número da versão, condições de início e fim, dados de controle sobre processos, nome de atividades, tipos de atividades, pré e pós-condições de atividades, condições de fluxo, condições de execução, informações organizacionais e informações sobre aplicações externas relacionadas [KAS 2000].

Além disso, uma ferramenta de definição deve fornecer uma série de comandos (APIs – *Application Programming Interface*) para permitir o acesso aos dados de definição de um processo. Assim, deve permitir que participantes do sistema criem suas sessões de trabalho, que seja possível a utilização de operações de definição de um processo específico e ainda que seja possível o manuseio de objetos específicos de definição (como, por exemplo, o ajuste de um parâmetro de uma ferramenta externa a ser invocada).

2.5.3 Aplicações de *Workflow* para Clientes

O gerenciador de lista de trabalhos é uma entidade de software que interage com o usuário final e com aquelas atividades que envolvem recursos humanos em um WfMS. Igualmente à ferramenta de definição de processo, o gerenciador de lista de trabalho pode ser fornecido como parte do WfMS ou pode aparecer como um produto de terceiros.

A interação do gerenciador de lista de trabalhos com o serviço de execução do *workflow* é baseada no conceito de lista de trabalhos. Uma lista de trabalhos é uma fila de itens de trabalho atribuídos por uma determinada máquina de *workflow* a um determinado usuário ou grupo de usuários. A interação mais simples ocorre quando uma máquina de *workflow* atribui um item de trabalho na lista de trabalhos para que o gerenciador de lista de trabalho posteriormente recupere este item e encaminhe para o seu destinatário.

É possível também, que seja necessário após a recuperação de um item de trabalho pelo gerenciador de lista de trabalhos, a invocação de uma determinada aplicação externa relacionada a este item. Os dados referentes à aplicação externa invocada podem ser de responsabilidade do próprio gerenciador de lista de trabalhos ou da máquina de *workflow*, dependendo da implementação do WfMS.

As interfaces de um gerenciador de lista de trabalho devem garantir que seja possível o estabelecimento de sessões de trabalho para usuários e que seja possível também a manipulação da lista de trabalhos [LAZ 2001]. Em alguns casos, as aplicações de clientes podem ter funcionalidades maiores do que a manipulação da lista de trabalhos. Elas podem ter funções que permitam a definição de operações dentro do

workflow, funções de controle de processos, funções de estados de processos, de administração, além das de invocação de aplicações externas.

2.5.4 Aplicações Invocadas

Um sistema gerenciador de *workflow* deve ter a capacidade de invocar certas aplicações para automatizar uma atividade (ou parte dela) que faz parte de um item de trabalho de um determinado participante [KAS 2000].

Os tipos de invocação que podem ser feitas por um WfMS para uma aplicação podem variar de uma simples chamada local de função até a execução de procedimento remoto. Algum tipo de mecanismo de integração pode ser utilizado na implementação de um sistema gerenciador de *workflow* para garantir a comunicação entre a máquina de *workflow* e a aplicação invocada.

As informações que permitem à máquina de *workflow* ter condições de localizar e, posteriormente invocar uma determinada aplicação, são fornecidas pela definição de processo. Para que uma aplicação possa ser localizada ela deve estar acessível localmente ou através de uma rede.

O interfaceamento entre a máquina de *workflow* e a aplicação invocada é composta por uma série de funções que permitem o manuseio e gerência de atividades e as sinalizações entre os elementos envolvidos em relação ao estado das atividades, além de fornecer dados da máquina de *workflow* para a aplicação.

2.5.5 Interoperabilidade entre Serviços de Execução de *Workflow*

Através da interface de interoperabilidade, é possível que várias implementações diferentes de WfMSs, troquem itens de trabalho entre si, possibilitando dessa forma, que duas ou mais máquinas de *workflow* se comuniquem e trabalhem juntas.

O objetivo aqui não é padronizar a implementação dos sistemas e sim a forma de cooperação entre eles. A idéia é fornecer um ambiente composto por máquinas de *workflow* provenientes de diferentes implementações, mas que se comporte como um único serviço de execução do *workflow*. Dentro deste contexto, vários cenários de interoperabilidade existem, descrevendo caminhos alternativos através dos quais uma instância de processo é compartilhada entre as máquinas de *workflow*.

As funções fornecidas para permitir essa interoperabilidade entre as máquinas de *workflow* envolvidas devem suportar a passagem de atividades, funcionalidades de controle de atividades e processos, sincronização, trocas de informações relevantes e acessibilidade à definição de processo [LAZ 2001].

2.5.6 Ferramentas de Administração e Supervisão

Por fim, é necessário também a existência de ferramentas e interfaces que possibilitem atividades de administração e monitoramento dentro de um WfMS. Dessa maneira, torna-se possível a avaliação do estado geral e a extração de métricas do sistema, elementos que são indispensáveis em algumas organizações.

Uma ferramenta de gerenciamento e supervisão deve ter a capacidade de operar sobre diferentes sistemas gerenciadores de *workflow* e, além disso, de alguma forma os serviços de execução de *workflow* envolvidos devem fornecer acesso aos dados referentes aos estados dos sistemas. Funções relacionadas a aspectos de segurança, controle e autorização dentro do sistema devem ser suportadas por uma ferramenta deste tipo [KAS 2000]. Além disso, outras funcionalidades como: a administração de usuários, gerenciamento de papéis, operações de auditoria, controle de recursos, supervisão de processos e análise do estado dos processos também devem ser funções suportadas.

2.6 Considerações Finais

Este capítulo apresentou uma introdução à tecnologia de *workflow* visando destacar conceitos importantes tomados como base para o desenvolvimento de uma proposta de arquitetura de linha de produto para sistemas de gerenciamento de *workflow*.

Com o objetivo de criar padrões para esta área, surgiu em 1993 a *Workflow Management Coalition* (WfMC) [WMC 95] [WMC 99]. A partir da arquitetura genérica de sistemas gerenciadores de *workflow*, a WfMC desenvolveu o modelo de referência para WfMSs, o qual é utilizado como aplicação de estudo de caso neste trabalho.

As principais características do modelo de referência foram exploradas e para cada um dos componentes presentes neste modelo, foram feitas descrições apresentando suas principais interfaces e formatos de intercâmbio de dados.

A abordagem de linha de produto é aplicável aos domínios em que existe uma demanda por produtos específicos, no entanto existe um conjunto de características comuns e pontos de variabilidade bem definidos. O domínio dos Sistemas de Gerenciamento de *Workflow* (WfMS) é altamente favorável à aplicação da abordagem de linha de produto, devido aos esforços da *Workflow Management Coalition* (WfMC) apresentados neste capítulo. Cada implementação de um WfMS pode adaptar componentes ou interfaces de acordo com as necessidades da aplicação. Produtos de *workflow* com características similares, porém com diferentes especificidades são necessários nas mais diversas empresas que utilizam esta tecnologia.

O capítulo seguinte apresenta conceitos importantes para o desenvolvimento de uma arquitetura de linha de produto para WfMS.

3 Linha de Produto de Software

Este capítulo apresenta uma introdução aos conceitos relacionados à linha de produto de software. Na seção 3.2 são apresentados os elementos de uma linha de produto, explorando as atividades essenciais para sua construção. O projeto da arquitetura de software tem sido objeto de muitas técnicas recentes, que devem ser consideradas no projeto de uma arquitetura de linha de produto. Dessa forma, na seção 3.3 são explorados os principais conceitos relacionados à arquitetura de software. Outra técnica importante que deve ser considerada são os métodos de desenvolvimento baseado em componentes [ALL 98] [D'SO 99] [JAC 99] [STE 2000]. Assim, a seção 3.3 apresenta os principais conceitos relacionados ao Desenvolvimento Baseado em Componentes (DBC) e uma caracterização sucinta da abordagem Catalysis [D'SO 99] que guiou o processo de formalização da arquitetura de linha de produto para WfMS que está sendo proposta neste trabalho.

A seção 3.5 oferece uma caracterização dos principais conceitos relacionados à descrição de arquiteturas de software apresentando os principais conceitos. Esta seção inclui também uma breve caracterização de algumas ADLs disponíveis atualmente, especialmente sobre a ADL Rapide, adotada para descrever e validar a arquitetura proposta.

3.1 Conceitos Básicos

Os conceitos apresentados neste capítulo estão fortemente baseados em Gimenes e Travassos [GIM 2002]. Uma técnica importante para melhorar a qualidade e a produtividade do processo de software é a reutilização de soluções. Conceitos como arquitetura de software, desenvolvimento baseado em componentes, engenharia de domínio, padrões e *frameworks* estão sendo estudados e aprimorados para que possam ser aplicados na produção de uma família de produtos. Porém, o que falta nesse contexto é uma maneira sistemática e previsível de realizar a reutilização. Assim, o enfoque de linha de produto de software surge como uma proposta de construção sistemática de software baseada em uma família de produtos [GIM 2002].

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.

Uma linha de produto de software envolve um conjunto de aplicações similares dentro de um domínio que podem ser desenvolvidas a partir de uma arquitetura genérica comum, a arquitetura de linha de produto, e um conjunto de componentes que povoam a arquitetura. Esta abordagem tem por objetivo identificar os aspectos comuns e as diferenças entre os artefatos de software ao longo do processo de desenvolvimento da linha de produto, de modo a explicitar os pontos de decisão em que a adaptação dos componentes para geração de produtos específicos pode ser realizada. Para tal, durante o processo de desenvolvimento deve-se identificar os pontos de variabilidade que são pontos em que as características dos produtos podem se diferenciar. Esses pontos de variabilidade aparecem inicialmente na definição dos requisitos da linha de produto e devem ser representados ao longo de todo o processo de desenvolvimento. Dessa forma,

pode-se decidir por uma característica ou outra para um produto específico tanto em nível de projeto quanto em nível de implementação.

Ainda existe uma grande controvérsia sobre o enfoque de linha de produto, considerando técnicas anteriores com propósitos similares, principalmente: engenharia de domínio e *frameworks*. O que parece mais produtivo é considerar o enfoque de linha de produto e essas técnicas como complementares [GIM 2002]. O enfoque de linha de produto pode ser visto como uma forma de organização dessas técnicas que nasceu da necessidade evidente de empresas como a Celsius Tech [CLE 2001] que se encontrou em uma situação de sucesso por ter uma alta demanda por seus produtos, mas que não conseguiria atender sua demanda se não fosse encontrada uma forma de gerenciar sua família de produtos. Poulin [POU 97] afirma que linha de produto e domínios representam o mesmo nível de abstração e que o termo linha de produto foi criado para facilitar a comunicação entre o pessoal técnico e os gerentes que pensam em termos de produtos.

Frameworks e padrões também são recursos importantes para o enfoque de linha de produto. Para se conceber uma linha de produto é necessário partir de conceitos que levem a generalização de aplicações e que permitam instanciações para aplicações específicas. Os *frameworks* fornecem este tipo de recurso. Um *framework* pode ser descrito como uma arquitetura de software semidefinida que consiste de um conjunto de unidades individuais e de interconexões entre elas, de forma a criar uma infra-estrutura pré-fabricada para desenvolvimento de aplicações de um ou mais domínios específicos [GIM 2002]. Portanto, *frameworks* são utilizados como mecanismos de apoio a linha de produto. Da mesma forma, os padrões permitem que a experiência de desenvolvedores de software seja documentada e reutilizada, registrando-se soluções de projeto para um determinado problema ou contexto particular [GAM 95]. Portanto, o enfoque de linha de produto encontra nos padrões indicações de como construir tanto a arquitetura quanto os componentes desta.

A abordagem de linha de produto de software é um assunto recente na comunidade de engenharia de software. O primeiro *workshop* sobre a prática de linha de produto de software [BAS 97] foi promovido pelo *Software Engineering Institute* (SEI) em dezembro de 1996. O objetivo era compartilhar experiências da indústria de software e do governo com a prática da abordagem de linhas de produto visando explorar aspectos técnicos e não técnicos envolvidos.

Os resultados desse *workshop* estão apresentados em [BAS 97] e sintetiza as apresentações dos seminários e discussões que identificaram alguns fatores envolvidos na prática de linha de produto. Neste *workshop* também foram analisados alguns assuntos nas áreas de arquitetura de software, pessoal, organização, administração e modelos empresariais. Esses assuntos estão relacionados direta ou indiretamente com a abordagem de linha de produto.

3.2 Elementos de uma Linha de Produto de Software

Segundo Clements e Northrop [CLE 2001], as três atividades essenciais da construção de uma linha de produtos são:

- desenvolvimento do núcleo de artefatos;

- desenvolvimento do produto, e
- gerenciamento da linha de produtos;

Essas três atividades estão intrinsecamente relacionadas de tal forma que a alteração em uma delas implica em analisar o impacto nas demais [GIM 2002].

É importante notar que o desenvolvimento do núcleo de artefatos baseado em uma família de produtos faz a reutilização de artefatos para a linha de produto previsível. Clements e Northrop [CLE 2001] reconhecem explicitamente que a atividade de desenvolvimento do núcleo de artefatos tem sido chamada de engenharia de domínio e que a atividade de desenvolvimento do produto tem sido chamada de engenharia de aplicação. Weiss [WEI 99] também incorpora engenharia de domínio e engenharia de aplicação em sua abordagem denominada FAST (*Family-Oriented Abstraction, Specification and Translation*). Weiss reconhece que a família de produtos também é conhecida pelo termo domínio.

Segundo Gimenes e Travassos [GIM 2002], o processo de desenvolvimento de uma linha de produto de software pode ser visto como dois modelos de ciclo de vida, conforme mostra a Figura 3.1.

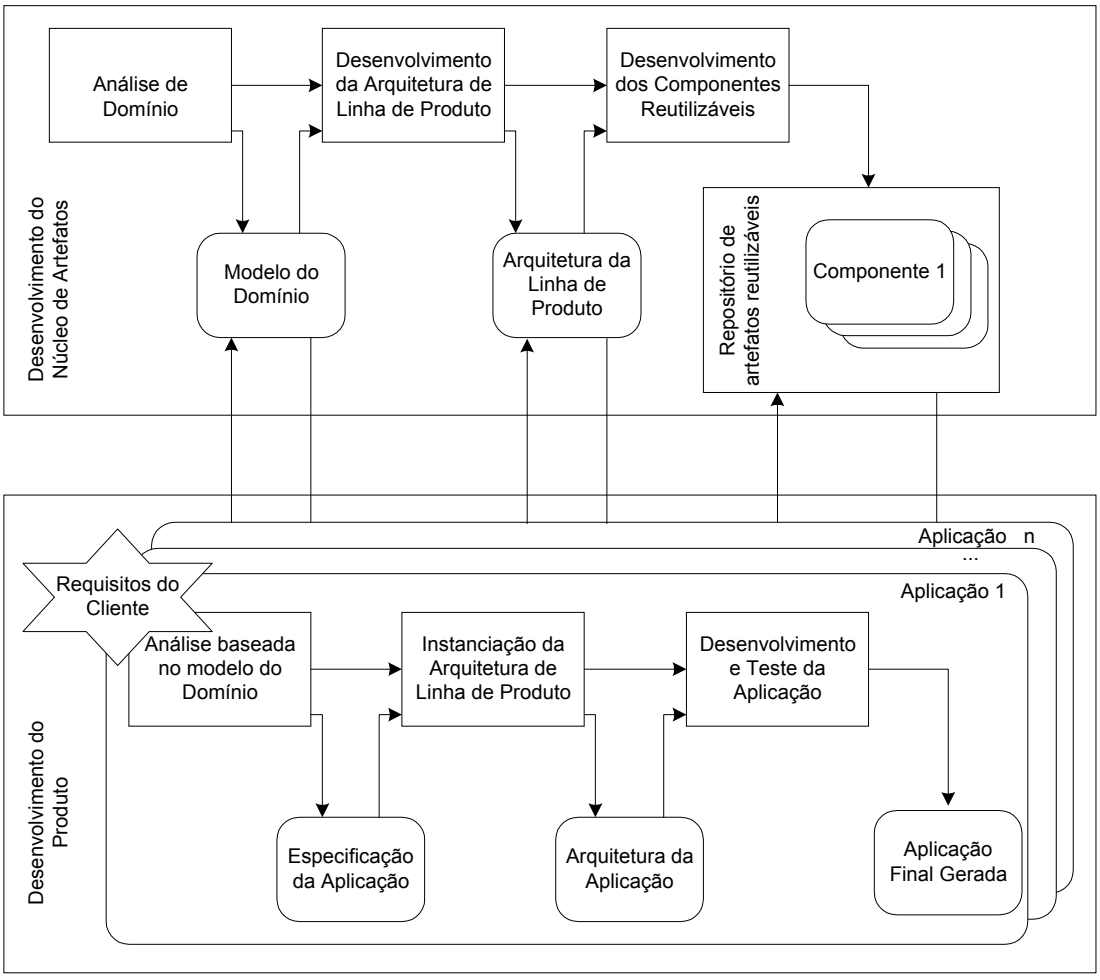

FIGURA 3.1 – O processo de desenvolvimento de uma linha de produto [GIM 2002].

O primeiro modelo do ciclo de vida representa o Desenvolvimento do Núcleo de Artefatos (Engenharia de Domínio), o qual engloba análise de domínio,

desenvolvimento da arquitetura e desenvolvimento de componentes reutilizáveis. Este ciclo produz um modelo de domínio, uma arquitetura, um conjunto de componentes reutilizáveis e geradores de software para a linha de produtos. Esses conceitos serão explorados na seção 3.2.2. O segundo modelo representa o Desenvolvimento do Produto (Engenharia de Aplicações) e será sucintamente caracterizado na seção 3.2.3, porém não faz parte do escopo deste trabalho.

Um dos conceitos mais importantes no enfoque de linha de produto de software é o de variabilidade. Assim, antes de explorar os modelos de ciclo de vida, este conceito será explorado e definido na seção seguinte.

3.2.1 Variabilidade

Um dos pontos principais no desenvolvimento de uma linha de produto é abstrair e representar as variabilidades associadas à arquitetura e aos seus componentes. Produtos em uma linha de produto existem simultaneamente e podem se diferenciar em termos de comportamento (conjunto de ações), atributos de qualidade, plataforma, configuração física, fatores de escala, entre muitos outros [CLE 2001].

Variações são diferenças tangíveis entre produtos que podem ser reveladas e distribuídas entre os artefatos da linha de produto, sejam eles a arquitetura, os componentes, as interfaces entre componentes ou as conexões entre componentes. As variações podem ser reveladas em qualquer fase do ciclo de desenvolvimento de uma linha de produto, a começar da análise de requisitos [GIM 2002]. A variação mais conhecida em construção de sistemas é a existência de várias implementações para uma mesma especificação. A expressão de uma variação pode ser obtida pela introdução de parâmetros instanciáveis em tempo de construção associados a componentes, subsistemas ou coleção de subsistemas a partir dos quais um produto pode ser configurado, atribuindo-se um conjunto de valores a esses parâmetros [CLE 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.

Em orientação a objetos variações podem ser representadas, em nível de projeto, por meio de especializações de determinadas classes. Existem vários mecanismos para o tratamento de variabilidade [GIM 2002] [JAC 97]. 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.

Um conceito importante na representação de pontos de variação é o conceito de *feature*. Este conceito tem origem na engenharia de domínio [KAN 90]. Uma *feature* é uma característica de um produto que usuários e clientes consideram importantes na descrição e distinção de membros de uma família de produto [GRI 98]. 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. Um modelo de features é, geralmente, representado por um grafo em que os nós contém as features e

seus atributos. Este grafo contém decorações para indicar features opcionais, obrigatórias e composição de features.

Em [GIM 2002], os autores apresentam a relação existente entre o modelo de casos de uso e o modelo de features segundo [GRI 98], destacando as diferenças existente entre esses dois modelos, que incluem:

- o modelo de casos de uso é orientado para o usuário e o modelo de features orientado para o reutilizador;

- o modelo de casos de uso descreve os requisitos do usuário em termos de funcionalidades do sistema. O modelo de features organiza o resultado da análise dos aspectos comuns e variáveis, preparando uma base para a reutilização;

- o modelo de casos de uso deve cobrir todos os requisitos de uma sistema individual do domínio, enquanto que o modelo de features deve incluir apenas aquelas características que o analista do domínio considera importante, pois este resume apenas os itens essenciais relativos aos objetivos do domínio;

- a notação utilizada para representar features é diferente daquela utilizada nos modelos de caso de uso, pois nem todas as features podem ser automaticamente relacionadas com casos de uso;

- nem todas essas features aparecem nos casos de uso, apesar de existir um grupo de features básicas que o usuário vê como capacidades do sistema. 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.

Segundo a abordagem FeatuRSEB [GRI 98], a construção do modelo de features é um processo concorrente que tem como entrada sistemas exemplo, requisitos, especialistas do domínio e modelos anteriores do domínio. Essas entradas são alimentadas em um processo concorrente e contínuo de definição do contexto, modelagem de features e modelagem de casos de uso. O processo de construção do modelo de features pode ser resumido como segue:

- mescle os modelos de caso de uso individuais para formar o modelo de casos de uso do domínio, mostrando os pontos de variação;

- crie um modelo de features funcionais derivadas do modelo de casos de uso do domínio;

- crie o modelo de objetos de análise, aumentando o modelo de *features* com *features* de arquitetura. Essas *features* estão relacionadas com a estrutura e configuração do sistema ao invés de funções específicas;

- crie o modelo de projeto, aumentando o modelo de *features* com as *features* de implementação.

Jacobson et al. [JAC 97], apresentam um conjunto de mecanismos de variabilidade. Componentes abstratos podem ser especializados utilizando uma série de técnicas de generalização e especialização, o qual os autores tratam como mecanismos de variabilidade. Esses mecanismos incluem:

- Herança: é usada para criar subtipos ou subclasses que especializam tipos abstratos ou classes em seus pontos de variação. Uma operação virtual pode ser pensada como um ponto de variação que pode ser especializado através de herança;

- *Uses*: este é um mecanismo de herança em casos de uso;

- Extensões e pontos de extensões: são pequenos anexos particulares que podem ser usados para expressar variações (extensões) anexadas a pontos de variação (pontos de extensão) em casos de uso e componentes;

- Parametrização: utilizado em tipos e classes através de *templates*, *frames* e macros;

- Configuração e Linguagens de Interconexão de módulos: são utilizados para conectar componentes opcionais ou alternativos em configurações completas;

- Geração de componentes derivados e vários relacionamentos de linguagens ou *templates*.

Esses mecanismos não são totalmente distintos, podendo às vezes ser necessária uma combinação de dois ou mais mecanismos. O reutilizador necessita escolher o mecanismo que fornece suporte ao tipo correto de variabilidade e a possível faixa de variação. A escolha correta dos mecanismos auxilia na construção de um conjunto de componentes genéricos. Esses componentes podem então ser combinados de várias formas para a produção de sistemas de forma rápida e eficiente. A Tabela 3.1 extraída de Jacobson [JAC 97] apresenta uma descrição sumária de como os pontos de variação são expressos em cada um dos mecanismos sugeridos. Apresenta ainda situações em que o mecanismo melhor se adapta.

TABELA 3.1 – Mecanismos de Variabilidade [JAC 97].

Mecanismo	Tipo de ponto de variação	Tipo de variante	Melhor utilizar quando
Herança	Operação virtual	Subclasse ou subtipo	Especializar e adicionar operações, mas mantendo outras
Extensões	Ponto de extensão	Extensão	Pode ser possível anexar várias variantes a cada ponto de variação ao mesmo tempo
Uses	*Use point*	Caso de uso	Reutilizar um caso de uso abstrato para criar um caso de uso especializado
Configuração	Item de configuração	Item de configuração	Escolher funções alternativas e implementações

Parâmetros	Parâmetro	Parâmetro de ligação	Exitir várias pontos variação pequenos para cada *feature* variável
Geração	Parâmetro ou linguagem *script*	Parâmetro de ligação ou expressão	Fazer criação em larga-escala de um ou mais tipos ou classes de uma linguagem específica do problema

Jacobson [JAC 97] discute *features* relacionadas com casos de uso. Uma *feature* é definida como um caso de uso ou parte deste. Dessa forma, no projeto de uma família de produto deve existir alguma forma de representar os aspectos que diferenciam os produtos membros da linha, como a abordagem de *features*. Esses aspectos serão considerados na próxima fase, desenvolvimento do produto, para instanciar o produto específico adequado.

3.2.2 Desenvolvimento do Núcleo de Artefatos

No desenvolvimento do núcleo de artefatos três aspectos devem ser considerados: a definição do contexto da linha de produto, o núcleo de artefatos e o plano de produção. Um artefato é um item reutilizável de software utilizado como bloco de construção de uma linha de produto. Pode ser uma arquitetura de software, um componente, um *framework*, ou alguma documentação relativa à arquitetura de software ou a algum componente.

Arquitetura da linha de produto

Dentre os artefatos da linha de produto, a arquitetura tem um papel importante. Ela representa a infa-estrutura central da linha de produto. A arquitetura de um sistema de software engloba a definição de suas estruturas gerais, descrevendo os elementos que compõem os sistemas e as interações entre estes [BAS 98]. Além de descrever esses elementos e suas interações, uma arquitetura de software apóia também questões importantes de projeto, tais como: a organização do sistema como composição de componentes, as estruturas de controle globais, os protocolos de comunicação, a composição dos elementos do projeto e a designação da funcionalidade dos componentes do projeto. Assim, é fundamental que hajam técnicas de engenharia de software que tornem sua especificação precisa e que possibilitem a flexibilização necessária para produção de diferentes produtos de uma mesma família.

Uma arquitetura representa um investimento significativo para uma organização. Em função disto, é natural que essas organizações desejem maximizar este investimento pela reutilização da arquitetura em vários sistemas. Quando uma organização está produzindo vários sistemas similares e reusando a mesma arquitetura e os componentes associados a esta, pode se beneficiar de algumas vantagens substanciais como a redução dos custos de produção e do tempo para disponibilizar este produto no mercado.

A arquitetura genérica ou *framework* de arquitetura da linha de produto representa um meta-projeto das aplicações que constituem a família de produtos. Alguns requisitos importantes da arquitetura de linha de produtos são [CLE 2001]:

- ser estável, persistindo durante a vida da linha de produtos, sofrendo poucas alterações;
- permitir integração flexível de novas funcionalidades durante o ciclo de vida da arquitetura;
- permitir a representação explícita de variações de modo a maximizar a reutilização;
- fornecer mecanismos para implementar variações;
- permitir as variações inerentes da família de aplicação, e
- permitir as variações impostas pelo mercado.

A seção 3.3 explorará melhor alguns conceitos relacionados à arquitetura de software.

3.2.3 Desenvolvimento do Produto

O desenvolvimento do produto, também conhecido como engenharia de aplicação, em geral, inclui os seguintes artefatos:

- o modelo do domínio, base para identificar os requisitos do cliente;
- um *framework* de arquitetura de linha de produto, base para especificar uma arquitetura para um membro da família;
- um conjunto de componentes reutilizáveis a partir do qual um subconjunto de componentes será integrado à arquitetura para gerar um produto.

De acordo com o ciclo de vida mostrado na Figura 3.1, a primeira atividade é a análise baseada no modelo do domínio. Nesta atividade, o modelo de domínio é utilizado para elicitar informações sobre a aplicação específica a ser desenvolvida e definir uma lista de características para a aplicação. Assim, o modelo do domínio representa as necessidades do cliente em termos da arquitetura de linha de produto e dos componentes disponíveis no repositório. Associado ao modelo do domínio, tem-se o modelo de decisão que define os principais aspectos a serem considerados no desenvolvimento do produto [GIM 2002]. Este modelo permite ao desenvolvedor da aplicação, por exemplo, identificar os ajustes necessários nos valores iniciais das características descritas pelos componentes, tendo em vista as restrições arquiteturais impostas pelas variabilidades. As necessidades não cobertas pelo modelo são denominadas novos requisitos. Os novos requisitos podem implicar na necessidade de estender a arquitetura de linha de produto e adquirir novos componentes. Para os requisitos não satisfeitos pelo modelo existente, deve ser investigado o custo de desenvolver o software por encomenda, modificar a arquitetura ou os componentes existentes. A lista de características da aplicação recuperada do modelo de domínio mais aquelas definidas nos novos requisitos são integradas para formar a especificação

do produto. Informações sobre os requisitos que não foram antecipados no modelo de domínio devem ser realimentadas para estender o modelo de domínio e/ou repositório de componentes existentes.

A segunda atividade é a instanciação da arquitetura do produto. O *framework* de arquitetura é percorrido de cima para baixo, tomando-se decisões recursivamente até que a arquitetura genérica seja transformada na arquitetura específica do produto. O processo de construção da arquitetura é iterativo. A instância inicial provavelmente contém pendências como pontos de variação e decisões ainda não resolvidas. Essas pendências, geralmente, ocorrem devido aos requisitos específicos da aplicação. A implementação desses requisitos pode provocar modificações no *framework* de arquitetura da linha de produto ou apenas afetar a instância da arquitetura da aplicação. A decisão depende da perspectiva de utilização do requisito por outros clientes.

A terceira atividade refere-se ao povoamento da arquitetura com os componentes adequados. As técnicas utilizadas neste processo são similares àquelas utilizadas para o desenvolvimento baseado em componentes. Como nas atividades anteriores, atenção especial deve ser dada a implementação dos requisitos específicos da aplicação. Os componentes podem ter várias origens, tais como:

- o repositório da linha de produtos pode conter o componente desejado;
- um novo componente é implementado, encomendado ou adquirido de terceiros, pois os requisitos do usuário não foram satisfeitos pelos componentes existentes no repositório;
- o requisito do cliente é implementado na própria aplicação;
- um componente de prateleira (COTS) é adquirido, e
- um componente é obtido de aplicações legadas.

Os testes da aplicação são feitos também de forma similar às técnicas de desenvolvimento baseado em componentes que incluem: teste de unidade e verificação de componentes, teste de integração e verificação geral das funcionalidades da aplicação.

3.2.4 Gerenciamento do Produto

Grande parte do sucesso de uma linha de produto depende do compromisso da organização em construí-la e mantê-la. Assim, é importante que a organização estabeleça um plano de adoção da linha de produtos que descreva o estado desejado e as estratégias para atingir este estado [CLE 2001]. Para tal é necessário prover a equipe responsável pela linha de produto com recursos e garantir que as atividades da linha de produto sejam coordenadas e supervisionadas. Deve haver um sincronismo entre o grupo que desenvolve os artefatos e o grupo que gera os produtos. Deve também ser garantido o apoio ao desenvolvimento e à evolução dos artefatos da linha.

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

A aplicação da técnica de linha de produto está atualmente favorecida pelo amadurecimento de técnicas da engenharia de software. Dentre essas técnicas, pode-se afirmar que a consciência da importância da arquitetura de software tem aumentado consideravelmente nos últimos anos. A arquitetura de software representa o núcleo de uma linha de produto. Técnicas que tornem a sua especificação precisa e que permitam a flexibilização necessária para a produção de diferentes produtos de uma mesma família são fundamentais para o enfoque de linha de produto. Dessa forma, a seção seguinte apresenta os principais conceitos relacionados à arquitetura de software.

3.3 Arquitetura de Software

Um dos fatores que servem de motivação para o estudo da arquitetura de um produto de software é o crescente aumento do tamanho e da complexidade desses produtos nos últimos anos. Neste contexto, os projetistas de software começaram a dar mais importância à estrutura dos produtos e à reutilização de seus componentes. Adotar uma estrutura correta para esses produtos pode trazer benefícios como a redução do seu tempo de manutenção. A manutenção é facilitada pela organização do produto de software, o que facilita sua compreensão e também possibilita a substituição de módulos constituintes, na medida em que seus limites estejam bem definidos [SIL 99].

Em função dos fatores descritos acima, o projeto e a especificação global da estrutura de um produto de software chega a ser uma questão tão significante quanto à escolha de algoritmos e estruturas de dados durante o projeto do produto. Questões estruturais incluem a organização dos produtos de *software* como uma composição de componentes, os protocolos de comunicação, a sincronização e o acesso aos dados.

A visão de um produto de software pode se ater a diferentes níveis de abstração. De fato, diferentes técnicas de modelagem produzem descrições de um software em diferentes níveis de sua escala de abstração. A arquitetura representa o mais elevado grau de abstração de um software, que o define em termos de um conjunto de artefatos de software e conexões entre esses artefatos.

Segundo Shaw [SHA 96], a arquitetura de software surgiu como uma evolução natural das abstrações de projeto, na busca de novas formas de construir produtos de software maiores e mais complexos. A arquitetura de software busca formas de descrever organizações de sistemas de software existentes, com vista a promover a reutilização de experiência de projeto, possibilitando a escolha da forma de organização mais adequada para um produto de software a ser desenvolvido.

Apesar da descrição da arquitetura de software ser mais abstrata no ciclo de vida de um produto de software suprimindo detalhes da implementação, esta define aspectos estruturais importantes e fornece uma base para as outras fases de desenvolvimento do produto de software [BOS 2000].

A definição da arquitetura de um produto de software viabiliza a reutilização de artefatos de projeto permitindo maior rapidez no desenvolvimento. Pode-se dizer ainda, que a definição da arquitetura de um produto de software facilita a comunicação entre os desenvolvedores de software e permite um entendimento maior da evolução desse produto.

A origem da arquitetura de software está na verificação de que determinadas estruturas de software são adequadas para determinados tipos de problema de projeto. Por exemplo, o estilo arquitetural *pipeline*, em que fluxos de dados são transportados linearmente através de um conjunto de componentes de software, tem sido usado para a construção de compiladores. A experiência de construção de software produziu várias estruturas organizacionais de software que são aqui tratadas como estilos arquiteturais.

Um estilo arquitetural define uma família de sistemas em termos de um modelo de organização estrutural ou mais especificamente, um estilo arquitetural define um vocabulário de componentes e tipos de conectores, um conjunto de restrições e formas de como esses componentes podem ser combinados. Para alguns estilos, podem existir um ou mais modelos semânticos que especificam como determinar as propriedades globais de um sistema ou parte dessas propriedades [BAS 98] [SHA 96].

Um estilo pode ser pensado como um conjunto de restrições sobre uma arquitetura de *software* – restrições dos tipos de componentes e seus padrões de interação. Essas restrições definem um conjunto ou família de estilos arquiteturais que as satisfaçam [BAS 98] [JAC 97]. Um estilo arquitetural não é uma arquitetura, mas traz consigo uma visão geral do sistema – impondo restrições úteis para a especificação do produto de *software* a ser desenvolvido.

O estilo mais adequado para um determinado sistema, depende primeiramente de seus requisitos de qualidade. Transformar uma arquitetura de software pela aplicação de um estilo arquitetural, resulta em uma completa reorganização da arquitetura original [BOS 2000].

Embora estilos arquiteturais possam ser combinados, deve-se ter certa atenção, pois, se durante a especificação de um projeto, um segundo estilo arquitetural for selecionado para fazer parte do sistema, é necessário ter certeza que as restrições dos dois estilos não conflitem entre si [BOS 2000].

Alguns estilos arquiteturais que podem ser encontrados na literatura técnica [SHA 96] [BOS 2000] [JAC 97] são: camadas, tubos e filtros (*pipes and filters*), organização orientada a objetos, sistemas baseados em invocação implícita (*Implict Invocation Systems*) e sistemas baseados em repositórios. Um detalhamento desses e estilos arquiteturais são apresentados em Lazilha [LAZ 2001].

A evolução das técnicas de desenvolvimento baseado em componentes também contribui com o enfoque de linha de produto. Mecanismos que permitem a especificação e a construção de componentes oferecendo a representação de variações entre estes é um ponto importante para o enfoque da linha de produto. Assim, a seção seguinte oferece uma introdução sobre DBC e uma caracterização sumária das principais abordagens existente, especialmente a abordagem Catalysis [D'SO 99] que guiou o processo de formalização da arquitetura de linha de produto para WfMS que é proposta neste trabalho.

3.4 Desenvolvimento Baseado em Componentes

A seção 3.3 tratou a arquitetura de software como um conjunto de componentes juntamente com uma descrição de suas propriedades, regras de como estes componentes podem interagir uns com os outros e os estilos de interação destes componentes na

arquitetura. Quando se fala de arquitetura de software, está implícita a idéia de componente de software.

O crescente amadurecimento dos métodos de desenvolvimento baseado em componentes tem contribuído com a abordagem de linha de produto. Os componentes são os elementos que preenchem a arquitetura de linha de produto. Dessa forma, o oferecimento 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 também um ponto essencial para o enfoque de linha de produto. Esses métodos consideram, em seu contexto, conceitos importantes como padrões e *frameworks*.

Esta seção apresenta os conceitos básicos relacionados ao desenvolvimento baseado em componentes de software. Na seção 3.4.1 é apresentada uma caracterização de componentes de software. A seção 3.4.2 oferece uma breve caracterização da abordagem Catalysis [D'SO 99], escolhida para guiar o processo de obtenção da arquitetura de linha de produtos proposta neste trabalho.

3.4.1 Conceitos Básicos

Segundo Werner [WER 2000], até bem pouco 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 implícito. O Desenvolvimento Baseado em Componentes (DBC) surgiu como uma nova perspectiva para o desenvolvimento de software, cujos objetivos incluem: quebrar esses blocos monolíticos em componentes interoperáveis, reduzir a complexidade no desenvolvimento, reduzir os custos através da utilização de componentes que podem ser adequados a outras aplicações.

A caracterização do que seria um componente de software ainda não é um tópico fechado. Cada grupo de pesquisa caracteriza de maneira mais adequada ao seu contexto o que seria um componente. Em [WER 2000], são identificadas algumas definições apresentadas em workshops e conferências a partir de 1998 sobre DBC.

Segundo [D'SO 99], 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 [GIM 2000].

Segundo Bosch [BOS 2000], componentes podem ser classificados em 3 tipos: (1) componentes desenvolvidos pelo próprio cliente, (2) componentes adquiridos para um determinado domínio e (3) os chamados componentes *Commercial-Of-The-Shelf* (COTS) ou componentes de prateleira, que são componentes genéricos encontrados no

mercado. Durante o desenvolvimento de um produto a partir da arquitetura da linha de produto, esses três tipos de componentes podem ser selecionados para preencher a arquitetura do produto específico.

A técnica de desenvolvimento baseado em componentes visa fornecer um conjunto de ferramentas, técnicas e notações que possibilitem que, ao longo do processo de software, ocorra tanto a produção de novos componentes quanto a reutilização de componentes existentes. O amadurecimento dessas técnicas também contribui muito com o enfoque de linha de produto, 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 com menor esforço de desenvolvimento.

Para Werner [WER 2000], para que uma abordagem consistente de DBC tenha sucesso, é primordial o entendimento dos conceitos que estão por trás das funcionalidades disponibilizadas pelos componentes e seus relacionamentos. A concretização do conhecimento em componentes reutilizáveis e o uso de um modelo que apresente o interfaceamento entre os componentes (estilo arquitetural) também são questões importantes. Além disso, para que se possa desenvolver produtos de software baseados em componentes, é necessário o auxílio de métodos e ferramentas que facilitem a especificação dos produtos de software e garantam a qualidade dos mesmos.

Para que a reutilização de componentes seja possível é preciso que estes sejam projetados para serem reutilizáveis. O projeto de um componente deve ser conduzido de tal forma que, além de tornar a sua execução correta e eficiente, deve ser genérico para que se torne adaptável a vários propósitos (domínios) e não somente ao propósito (domínio) primário.

O processo de desenvolvimento baseado em componentes pode ser visto ainda como uma evolução natural da abordagem de orientação a objetos. Devido principalmente ao advento da Internet e da mudança da arquitetura cliente/servidor para a idéia de pequenas aplicações que se comunicam pela WEB, formando uma aplicação distribuída, a importância da distribuição e interoperabilidade de componentes heterogêneos se disseminou pela comunidade de desenvolvedores de software [WER 2000].

Diversos padrões para interoperabilidade de componentes surgiram no mercado nos últimos anos como CORBA (*Common Object Request Broker*) [OMG 2000], DCOM (*Distributed Component Object Model*) [MIC 2000] e RMI (*Remote Method Invocation*) [JDC 2001]. Como motivação para seu uso, estes padrões propõem um alto nível de abstração para viabilizar a interoperação dos componentes através da utilização de linguagens intermediárias, geradores de código e linguagens para definição de interfaces (IDLs – *Interface Definition Language*). A utilização desses padrões facilita a interoperação entre componentes [WER 2000].

Os mecanismos de interconexão, apesar de solucionarem a questão da heterogeneidade entre elementos que precisam interagir, acarreta em queda de desempenho, exigindo que este aspecto seja avaliado [SIL 99].

Ainda com relação a componentes de software, Bosch [BOS 2000] indica que existe uma diferenciação entre a compreensão acadêmica e a industrial deste conceito. A Tabela 3.2 apresenta uma caracterização geral destas duas visões. A visão acadêmica trata os componentes (artefatos) como entidades caixa-preta com uma interface restrita. A visão industrial trata os componentes como grandes pedaços de software, como *frameworks* orientados a objetos, com uma estrutura interna complexa e nenhum limite

de encapsulamento explícito. Devido a esta falta de limite de encapsulamento, os engenheiros de software são capazes de acessar qualquer entidade interna do componente, incluindo as entidades privadas.

Mesmo ao usar apenas entidades de interface, a utilização de artefatos é muito complexa devido ao tamanho do código. Variações, do ponto de vista acadêmico, estão limitadas em número e são configuradas durante a instanciação através de outros componentes do tipo caixa-preta. Na prática, a variabilidade de um componente é implementada através de configuração, mas também através de implementação ou substituição de entidades internas do componente. Além disso, múltiplas implementações de um artefato podem estar disponíveis para tratar a variabilidade requerida. Os componentes do ponto de vista acadêmico são providos de interfaces padronizadas e são usados em componentes de mercado. Na tentativa de se fazer isto, há um enfoque na funcionalidade do componente e sua apresentação formal. Na prática, quase todos os artefatos são desenvolvidos internamente. Em casos excepcionais, um componente pode ser adquirido externamente, requerendo, neste caso, uma adaptação dos componentes internos. Adicionalmente, a qualidade dos atributos dos artefatos têm no mínimo a mesma prioridade quando comparado à sua funcionalidade.

TABELA 3.2 – Visão Acadêmica versus a Visão Industrial de Componentes de Software [BOS 2000].

Visão Acadêmica	Visão na Indústria
Os artefatos reutilizáveis são componentes caixas-pretas.	Os artefatos são grandes pedaços de *software* (às vezes com mais de 80 KLOC[4]) com uma complexa estrutura interna e nenhum limite de encapsulamento. Ex: *frameworks* orientados a objetos.
Os artefatos possuem uma interface restrita através de um ponto de acesso.	A interface do artefato é provida através de entidades. Estas entidades de interface não têm nenhuma diferença explícita das entidades sem interface.
Os artefatos possuem poucos pontos de variação explicitamente definidos que são configurados durante a instanciação.	Variação é implementada através de configuração, especialização ou substituição de entidades do artefato. Às vezes podem ser requeridas múltiplas implementações (versões) de um artefato para se cobrir os requisitos de variação.
Os artefatos implementam interfaces padronizadas e podem ser comercializados no mercado.	Os artefatos são desenvolvidos principalmente nas organizações. Artefatos adquiridos externamente devem ser adaptados para alcançar os requisitos da arquitetura de linha de produto.
Foco na funcionalidade do artefato e na verificação formal da funcionalidade.	Atributos de funcionalidade e qualidade (por exemplo: performance, confiança, tamanho do código, reusabilidade e manutenibilidade) têm igual importância.

[4] KLOC – *Lines of Code* – Linhas de Código.

Neste trabalho, quando falamos em componentes de uma arquitetura de linha de produto, trata-se da visão industrial, onde componentes são grandes pedaços de software, podendo ser comparados a *frameworks* orientados a objetos.

Outros conceitos importantes relacionados a componentes e a DBC estão relacionados ao armazenamento e recuperação de componentes (repositórios de componentes), a arquitetura de software para conexão de componentes, certificação da qualidade de componentes e ainda evolução e adaptação.

Atualmente, tem crescido muito o interesse em métodos de desenvolvimento baseados em componentes. Este amadurecimento contribui muito com o enfoque de linha de produto, pois tornam-se necessários mecanismos para especificar e construir componentes de forma a permitir variações entre estes e que possam facilitar a geração de aplicações com características diferentes. Esses métodos focalizam os seguintes elementos-chave: componentes, interfaces e construção de componentes, e utilizam *Unified Modeling Language* (UML) [BOO 99] [FOW 97] [RUM 99] como notação. Exemplos desses métodos são o Processo Unificado da Rational [JAC 99], o *Select Perspective Method* [ALL 98], a abordagem *Sterling Software's Enterprise-CBD* [BRO 99], e o método Catalysis [D'SO 99] que foi escolhido para guiar o processo de desenvolvimento da arquitetura de linha de produto proposta neste trabalho. Este método será descrito a seguir.

3.4.2 O método Catalysis [D'SO 99]

O método Catalysis [D'SO 99] incorpora conceitos relevantes que apoiam o desenvolvimento de software para sistemas baseados em objetos e componentes. Este método baseia-se na linguagem de modelagem UML [BOO 99] [FOW 97] [RUM 99] e alinha-se aos padrões de sistemas de objetos distribuídos abertos, construídos a partir de componentes e *frameworks*.

Catalysis oferece modelos precisos e um processo de desenvolvimento completo e sistemático, permitindo aos desenvolvedores partirem da análise e especificação do domínio da aplicação e chegarem ao código, dividindo o sistema e identificando ao longo do processo os elementos de reutilização.

O método tem como base os princípios de abstração, precisão e compatibilização de partes. Esses princípios podem ser aplicados em todos os níveis de desenvolvimento de software a partir do domínio de um problema. A abstração propõe a visualização clara dos aspectos essenciais do problema, tais como o que o sistema deve fazer, a definição da arquitetura e da concorrência entre funções. A precisão é importante para evitar a ambigüidade de compreensão e para a análise de correspondência entre níveis de abstrações diferentes. A compatibilização de partes está intimamente ligada à reusabilidade, com o objetivo de reaproveitar os componentes, *frameworks*, padrões e especificações que constituem o software.

Em termos arquiteturais, o Catalysis descreve dois níveis, o nível da macro-arquitetura, que seria o nível de colaboração entre os componentes e a micro-arquitetura, que é a descrição de como cada componente é projetado internamente.

Existe uma grande preocupação em relação à interface dos componentes. Esta abordagem sugere que a colaboração entre os componentes para a formação da arquitetura seja feita através das interfaces. Por isso, existe todo um formalismo para a

definição das interfaces, com especificação de restrições, detalhamento da interface, entre outros, de forma que esta consistência possa ser verificada de maneira formal.

Um aspecto interessante do Catalysis é que, apesar de não ser um método de engenharia de domínio como o FODA, apresenta um conceito de tipo, que seria a descrição de alguma funcionalidade da aplicação em termos apenas de seu comportamento externo. Esta definição vem muito ao encontro da definição de *features* que descrevem funcionalidades. No entanto, o Catalysis não especifica nenhum tipo de modelo de relacionamento destas funcionalidades, capturando as similaridades e diferenças entre as mesmas, como é feito no modelo de features.

Neste trabalho, foi adotado o método Catalysis devido a sua forte enfâse na produção de *frameworks* e componentes. O método Catalysis permite também a reutilização de padrões e a extensibilidade de diagramas de pacotes, além de permitir a interação entre diagramas. Outra característica do Catalysis é sua abordagem direcionada ao comportamento, que possibilita um detalhamento maior das ações do domínio em questão. No capítulo 4 são mostradas as aplicações práticas do método Catalysis realizadas neste trabalho.

3.5 Descrição de Arquitetura de Software

Esta seção apresenta os principais conceitos relacionados às formas de descrição de arquiteturas de software. O foco desta seção está em caracterizar as ADL's, bem como a modelagem de componentes, conectores e configurações. Dentre as ADLs disponíveis foi selecionada a linguagem Rapide, por ser mais adequada para os interesses deste trabalho, por ser caracterizada como uma linguagem de descrição arquitetural com finalidade geral, que permite modelar interfaces de componentes e seus comportamentos externamente visíveis, e ainda por ser uma linguagem que permite simulação e modelagem do comportamento dinâmico descrito por uma arquitetura. Por fim, a seção 3.5.4 apresenta uma breve caracterização de outras ADLs disponíveis.

3.5.1 Introdução e Conceitos Básicos

A forma mais comum de descrever arquiteturas de software está apresentada na seção 3.3 em que foram mencionados alguns estilos arquiteturais que fazem uso de uma descrição textual e diagramas com seqüência imprecisa. Porém, esta forma de descrição possui os inconvenientes da informalidade, na qual se pode ter descrições incompletas e dúbias. Shaw [SHA 96] identifica um conjunto de requisitos para os mecanismos de descrição de arquiteturas de software que são apresentados a seguir:

- **composição:** deve ser possível descrever um produto de software como uma composição de módulos e conexões independentes;

- **abstração:** deve ser possível descrever módulos e suas interações na arquitetura de software, de modo a definir com precisão, de forma clara e explícita, seus papéis abstratos em um sistema;

- **reusabilidade:** deve ser possível reusar concepções de módulos, conectores e padrões arquiteturais em diferentes descrições arquiteturais, para diferentes tipos de sistemas;

- **configuração:** descrições arquiteturais devem permitir o entendimento e possibilitar a alteração da estrutura arquitetural de um produto de software sem a necessidade de se examinar individualmente cada um dos elementos do sistema. Devem suportar também reconfiguração dinâmica;

- **heterogeneidade:** deve ser possível combinar descrições arquiteturais distintas e heterogêneas;

- **análise:** deve ser possível proceder análise de descrições arquiteturais.

A seção seguinte apresenta os conceitos básicos relacionados a linguagens para descrição de arquiteturas de software.

3.5.2 Linguagens para Descrição de Arquiteturas (ADLs)

Uma ADL enfatiza estruturas de alto nível, omitindo detalhes de implementação e engloba algumas propriedades que se tornam desejáveis e importantes para a descrição da arquitetura de um produto de software. Segundo Medvidovic [MED 2000], uma ADL deve ser simples, entendível e possibilitar uma sintaxe gráfica bem compreendida. É desejável também que os tipos de ferramentas que auxiliam na visualização sejam entendíveis e que forneçam análises simples de descrições arquiteturais.

As ADLs são linguagens formais que podem descrever e representar produtos de software, e possuem características que permitem especificar a natureza dos componentes, suas propriedades, a semântica das conexões e o comportamento do sistema especificado. Uma ADL suporta a descrição de um sistema em termos de componentes e conectores [BAS 98].

Atualmente existe uma variedade de ADLs e algumas ainda estão sendo pesquisadas e analisadas. Apesar dessa diversidade, segundo Bass [BAS 98], as ADLs possuem algumas características comuns:

- a maioria das linguagens fornece uma sintaxe gráfica, possue uma forma textual e relaciona todas as características sintáticas e semânticas formalmente definidas;

- possuem características para a modelagem de sistemas distribuídos;

- fornecem um pequeno suporte para formalizar um histórico sobre projetos arquiteturais;

- manipulam fluxo de dados e fluxo de controle como mecanismos de interação, outros tipos de conexão são menos suportados;

- possuem a habilidade para representar níveis hierárquicos de detalhes e manipulam múltiplas instâncias de modelos como um caminho rápido para copiar subestruturas durante a criação da linguagem.

Uma ADL deve explicitamente modelar componentes, conectores e suas configurações, e ainda, para que seja verdadeiramente utilizável deve possuir uma ferramenta para o desenvolvimento e a evolução baseada na arquitetura. Além disso, deve fornecer os meios para a especificação explícita da ADL, o que nos permite determinar se uma notação particular é ou não uma ADL. A seção seguinte descreve a ADL Rapide.

3.5.3 A Linguagem Rapide

Nesta seção serão descritas as características mais relevantes da ADL Rapide. O objetivo desta seção é caracterizar a linguagem Rapide para que se entenda o seu funcionamento, bem como as formas de conexão entre componentes e principalmente como essa linguagem pode facilitar a construção de sistemas baseados em arquitetura.

A linguagem Rapide tem o propósito de fornecer facilidades para permitir que uma arquitetura seja usada para direcionar a estrutura de um sistema. Além disso, permite a prototipação do comportamento do sistema antes que seja completamente construído [LUC 95].

De acordo com o propósito deste trabalho, a linguagem Rapide foi escolhida, em virtude de ser uma linguagem que permite simulação e modelagem do comportamento dinâmico descrito por uma arquitetura.

Rapide é uma linguagem orientada a objeto desenvolvida para prototipar sistemas distribuídos, na qual suporta especificação, análise e verificação de arquitetura de sistemas compostas de componentes que processam eventos. A linguagem Rapide tem o propósito de fornecer facilidades para permitir que um plano de arquitetura seja usado para direcionar a estrutura de um sistema.

Uma arquitetura Rapide consiste de uma coleção de interfaces, uma coleção de conexões entre as interfaces e um conjunto de restrições formais que definem o comportamento legal e ilegal. A Figura 3.2 mostra um sistema de módulos conectados em uma arquitetura. Os paralelogramos sombreados representam as interfaces, as linhas grossas representam as conexões e as caixas representam os módulos. A arquitetura é mostrada como o conjunto de interfaces e conexões; o sistema é mostrado como o conjunto de módulos ligados pela arquitetura.

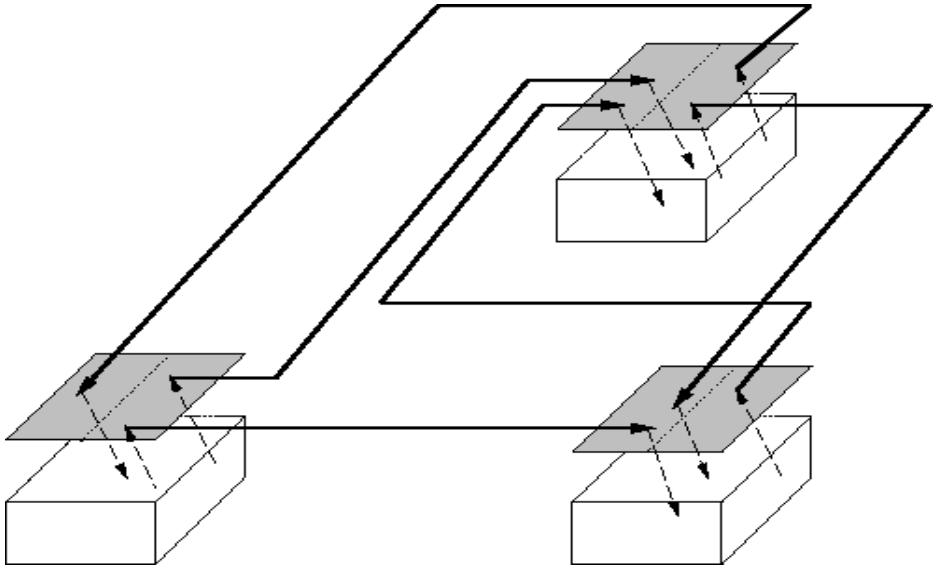

FIGURA 3.2 – Uma arquitetura Rapide e seu sistema.

Pode-se imaginar uma arquitetura sendo executada através do fluxo de comunicações dos módulos para suas interfaces, ao longo das conexões da arquitetura para outras interfaces, e indo para os módulos receptores, e assim por diante. Cada módulo e conexão pode executar de modo independente e concorrente. A comunicação em cada interface deve satisfazer as restrições da interface, e a comunicação nas conexões deve satisfazer as restrições da arquitetura.

O sistema da Figura 3.2 pode ser escrito em Rapide para suportar diferentes métodos de análise. Pode ser executado para permitir a simulação e animação do comportamento da arquitetura. Pode ser também puramente uma descrição baseada em restrições do comportamento permitido do sistema. Ademais, a definição baseada em restrições pode ser utilizada para verificar o comportamento da arquitetura por concordância às restrições, assim como o comportamento real de uma implementação.

Rapide é estruturada como um conjunto de cinco sublinguagens. Este conjunto de linguagens é chamado *framework.* O propósito deste *framework* é definir componentes que podem ser aplicados ou migrados para outras ferramentas de geração de eventos e não apenas para o simulador Rapide, além de ser um *framework* para a composição de módulos, e restrição de comunicação entre os módulos do sistema. As sublinguagens são:

- **linguagem de tipos:** fornece características básicas para a definição de tipos interfaces e tipos funções. Sua semântica é expressa por regras gerais que definem subtipos e supertipos, relacionando esses tipos para permitir substituição dinâmica de subtipos por supertipos;

- **linguagem padrão:** é uma parte fundamental de todas as estruturas executáveis (processos reativos, regras de comportamento e regras de conexão) do módulo executável, das linguagens de arquitetura e também das linguagens de restrição;

- **linguagem de arquitetura:** modela a arquitetura de conexão de interfaces;

- **linguagem de restrição:** proporciona características para definir restrições formais no comportamento de componentes e arquiteturas; e

- **linguagem executável:** contém estruturas de controle que são encontradas em muitas linguagens de programação e simulação.

3.5.3.1 Estrutura

Rapide suporta as abstrações de arquitetura básicas de componentes, conectores e configurações, proporcionando notações explícitas para cada um desses elementos. Estas notações de estruturas básicas são combinadas para descrever a arquitetura de software. A especificação de comportamento será descrita nas seções seguintes.

3.5.3.2 Modelando Componentes em Rapide

Para modelar um componente em Rapide, define-se um objeto chamado *Interface*. A linguagem Rapide faz a distinção entre relações síncronas e assíncronas. Um módulo pode comunicar-se sincronicamente com qualquer outro módulo, se ele chama uma de suas funções de interface requerida (*requires*), e existe uma conexão entre aquela função a uma função de interface fornecida (*provides*) do outro módulo. Por um outro lado, as ações *in* e *out* denotam eventos assíncronos. Assim, módulos do tipo interface podem comunicar-se assincronicamente com outros módulos. A comunicação é baseada em enviar e receber eventos que são gerados pelas ações, e então "transportados" pela conexão.

Rapide introduz um mecanismo único para expressar o comportamento de um componente e sua interação com outros componentes. São os conjuntos de eventos parcialmente ordenados chamados de *posets*. Por meio dos *posets* se pode visualizar como os eventos acontecem durante a execução. De modo conciso, quando uma arquitetura Rapide é executada, um *poset* é gerado. Quando um evento é gerado, ele pode produzir uma reação em conexões e módulos. Módulos e conexões tipicamente irão aguardar a geração de alguns eventos e algumas condições do estado do programa associado a estes eventos, então reagir aos eventos. Um módulo ou conexão reage através da execução de algum código que pode incluir mudanças ao estado do programa e a geração de eventos adicionais. Módulos e conexões geralmente ficam em ciclo: esperando, reagindo e gerando eventos múltiplas vezes. O *poset* gerado é então verificado à procura de violações da interface ou da arquitetura.

3.5.3.3 Modelando Conectores em Rapide

As conexões de Rapide são modeladas *in-line*, e como não são entidades de primeira classe, não podem ser nomeadas, subtipadas ou reutilizadas.

Existem dois tipos de conexões: básicas e complexas. Conexões básicas definem identidades entre pares de interfaces características, por exemplo, um par de funções ou um parte de ações. Identidade é a forma mais forte de conexão. Por exemplo, uma conexão básica entre uma função requerida de um componente e um função fornecida de outro, define a última função como um "*alias*" para a anterior. Desse modo, chamadas para a função requerida são devolvidas pelas chamadas para a função

fornecida. Conexões básicas entre funções e ações são essenciais para definir conexões entre componentes de uma arquitetura.

Conexões complexas definem conexões mais gerais entre componentes. Elas podem ativar os *posets* de eventos gerados por alguns componentes e então gerar *posets* de eventos que são recebidos através de outros componentes. Conexões complexas são essencialmente sintáticas para conectar componentes que podem ser definidos pelo tipo interface. Existem dois tipos de conexões complexas: conexões *pipe* que usa a sintaxe "=>", e conexões agente, que usa a sintaxe "||>". Pipes são conectores pipeline e agentes são conectores *multi-threaded*.

3.5.3.4 Modelando Configurações em Rapide

As configurações arquiteturais, ou topologias são grafos conectados de componentes e conectores que descrevem a estrutura de arquitetura. Esta informação é necessária para determinar se os componentes apropriados estão conectados, se suas relações combinam, se os conectores permitem uma comunicação apropriada, e se a semântica dessa configuração resulta no comportamento desejado [MED 2000]. Dessa maneira, Rapide oferece recursos para especificar detalhadamente os conectores partindo do princípio que uma configuração deve ser compreendida sem se conhecer os componentes e os conectores internamente.

Muitas ADLs proporcionam características explícitas para suportar composição hierárquica de componentes. Desse modo, Rapide permite descrever uma arquitetura em diferentes níveis de detalhes, o que é fundamental para uma ADL. Além disso, a linguagem Rapide suporta refinamento e rastreabilidade, proporcionando mapas para refinar a arquitetura em diferentes níveis [MED 2000]. Um mapa é uma estrutura que define regras de mapeamento que transformam a execução de uma ou mais arquiteturas em uma execução de uma outra arquitetura ou interface.

Enquanto a maioria das ADLs existentes visam configurações estáticas, Rapide suporta especificação e desenvolvimento de sistemas que podem ser alterados durante a execução. Segundo Medvidovic [MED 2000], todas as ADLs que permitem especificações de propriedades não funcionais de componentes e conectores, também permitem composição hierárquica, podendo assim, especificar tal propriedade da arquitetura tratando-a como composição de componentes. Rapide suporta modelagem direta de propriedades não funcionais de uma arquitetura por meio de sua linguagem de restrições. A especificação de restrições é fundamental para descrever dependências entre componentes e conectores.

3.5.3.5 Ferramentas de Apoio

A necessidade de ferramentas de apoio para uma ADL é bem conhecida. A maioria das ADLs fornecem ao menos duas formas de se modelar uma arquitetura – textual e gráfica – bem como suporte automatizado para elas. A Figura 3.3 apresenta uma análise das ferramentas de apoio à linguagem Rapide.

Existem diversas ferramentas para auxiliar os programadores a trabalharem com a linguagem Rapide. As próximas seções apresentam as ferramentas Rapide que incluem um editor de arquitetura, um compilador, um sistema de verificação de restrições em tempo de execução, um visualizador gráfico de *posets* e um conjunto de

ferramentas de animação para produzir, visualizar e animar *posets* gerados pelas computações do Rapide.

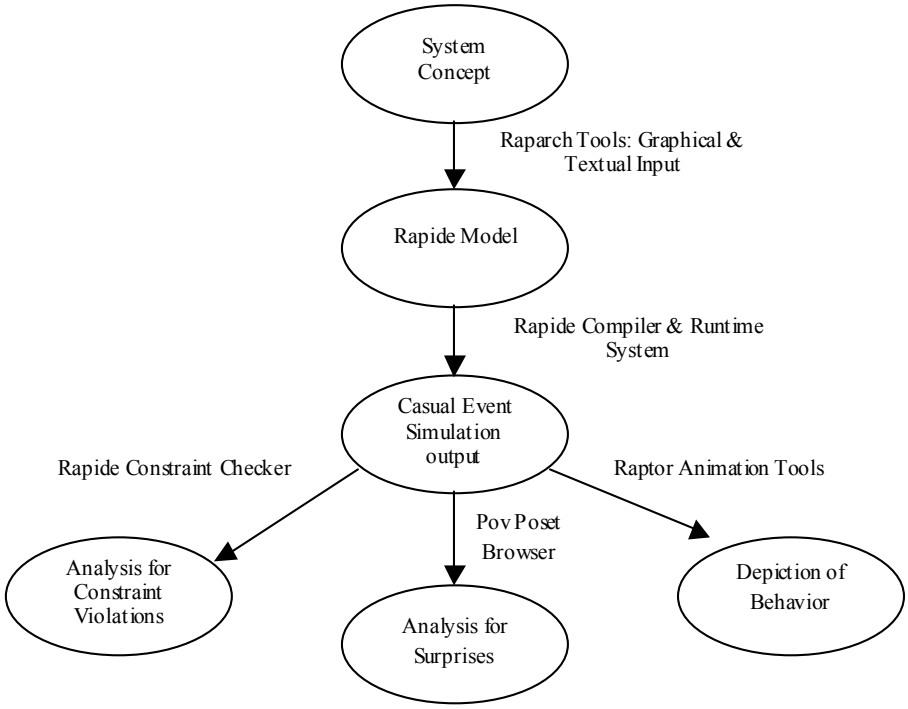

FIGURA 3.3 – Análise da Ferramenta [CSL 97]

Raparch – Editor de Arquitetura Rapide

Raparch é uma ferramenta para editar graficamente arquiteturas Rapide. Caixas e linhas podem ser desenhadas e editadas correspondendo aos módulos e conexões na arquitetura de um programa. Permite ainda associar o comportamento do protótipo ao sistema. Dessa maneira, podemos construir um modelo estrutural e comportamental da arquitetura do sistema.

A Figura 3.4 representa um simples exemplo de uma arquitetura de sistema, contendo um componente *Application Program* conectado a um componente *Resource*. De acordo com esta arquitetura, o componente *Application Program* faz uma solicitação ao componente *Resource* e recebe um resultado como resposta.

R.manager – Gerenciador de Biblioteca

Para compilar um programa Rapide, primeiramente uma biblioteca Rapide deve ser construída. Uma biblioteca Rapide armazena informações sobre os tipos e módulos definidos pelo usuário.

Compilador – Rpdc

O compilador Rapide analisa o código fonte, reporta os erros sintáticos e semânticos, e gera um programa executável. O programa executável gerará um arquivo log, contendo o histórico da execução, sendo este o evento log da computação.

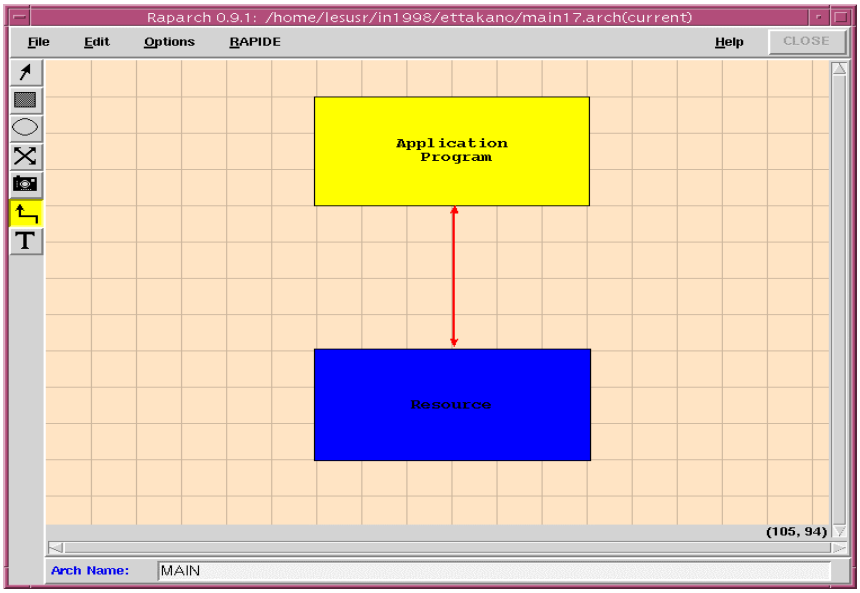

FIGURA 3.4 – Exemplo de uma simples arquitetura de sistema

Ferramentas de Análise

O resultado da simulação de uma arquitetura Rapide é um *poset* que mostra a história causal de eventos na execução, sincronismo e outras propriedades. Este recurso permite que se faça a simulação do comportamento de uma arquitetura antecipadamente ainda durante as fases de análise de requisitos e projeto do sistema.

Para maximizar o uso de simulações de eventos causais, são necessárias novas ferramentas de análise. Rapide é apoiada por três tipos de ferramentas para analisar simulações, caracterizadas a seguir.

Sistema de verificação de restrições em tempo de execução – Constraint Checking Runtime System (RTS)

Sistema de verificação de restrições semânticas que especificam características das conexões da arquitetura em Rapide, restringindo o comportamento visível dos componentes com suas relações. Os verificadores de restrições são usados para detectar automaticamente violações de restrições nas simulações.

Raptor – Animador de Simulação Rapide

Animação é a visualização das atividades de um sistema através de figuras. O movimento de mensagens em uma tela é um estilo de animação comum. Diferentes estilos de animação gráficos são apropriados para diferentes sistemas. A animação auxilia a compreensão humana. É uma poderosa ferramenta na prototipagem de arquiteturas. Normalmente, a animação de uma simulação no Rapide fornece o modo mais fácil para um usuário verificar o que o sistema está fazendo. Somente então é possível iniciar um processo mais formal de modificação do sistema, expressão de restrição no seu comportamento e assim por diante. Ainda, a animação de sistemas distribuídos é auxiliada por históricos causais, pois as dependências causais entre eventos implicam em simples regras sobre a ordem em que os eventos devem ser apresentados para proporcionar uma animação precisa.

O animador de simulações do Rapide chamada-se Raptor e consiste de um visualizador gráfico ativo de eventos da arquitetura que produz cenários da execução do

programa. O Raptor fornece uma ferramenta de demonstração para ilustrar e comunicar execuções da arquitetura. O visualizador fornece uma figura da arquitetura (atualmente de modo estático) e mostra o quê pode se comunicar com o quê. A ferramenta é ativa no sentido de que é capaz de reproduzir a execução de um programa. Foi desenvolvida com uma interface para históricos de eventos causais gerados por qualquer sistema, não somente o simulador Rapide.

A Figura 3.5 apresenta um instante da animação da arquitetura de sistema mostrada na Figura 3.4, onde o componente *Application Program* envia uma mensagem – Request() - ao componente *Resource* solicitando um recurso. O recurso sendo solicitado, o componente *Resource* envia uma mensagem – Result() – ao componente *Application Program*. Esta é uma especificação de uma arquitetura cliente-servidor, onde clientes fazem solicitações a servidores.

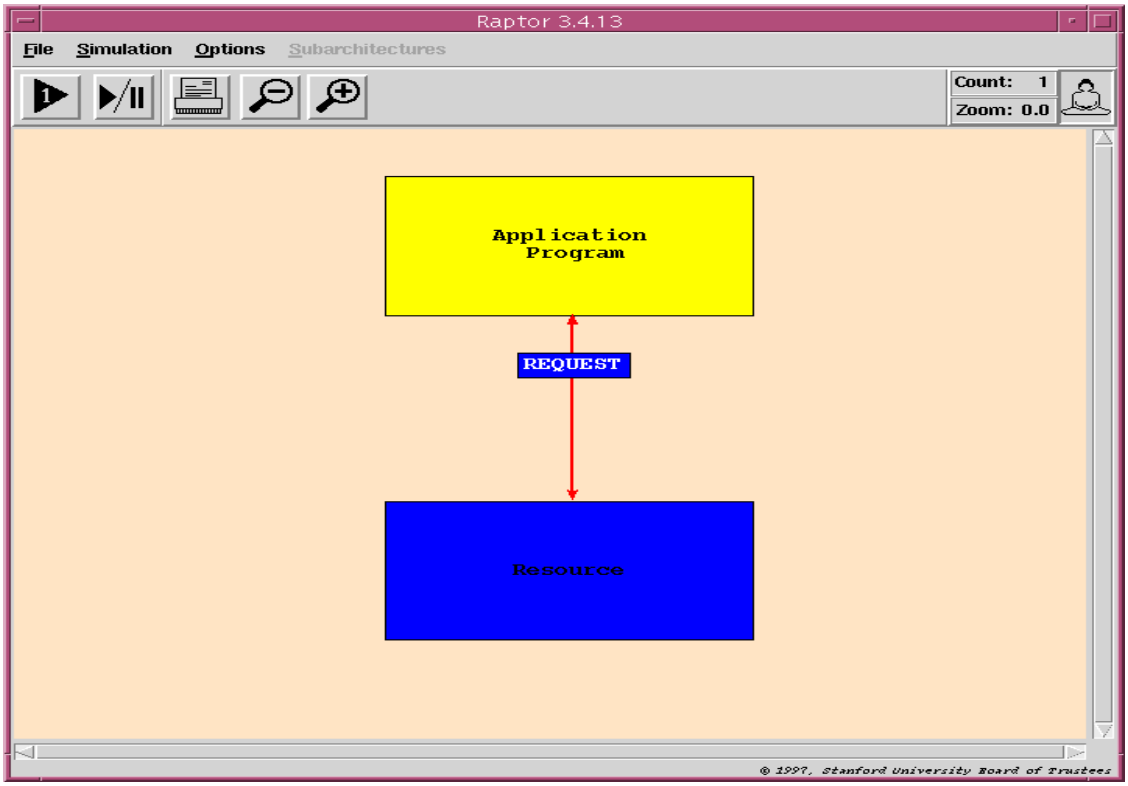

FIGURA 3.5 – Animação da arquitetura de sistema Application Program/Resource

Visualizador gráfico de posets – Partial Order (poset) Viewer (Pov)

Pov é uma ferramenta para visualizar graficamente os eventos parcialmente ordenados gerados pelos executáveis Rapide. O Pov pode criar diversas visões dos eventos que foram gerados durante a execução. Em particular, um grafo acíclico direcionado pode ser gerado cujas setas denotam o relacionamento causal (dependência) entre os eventos. Se uma seta vai de um evento A para um outro evento B, então o evento B depende do evento A. Esses eventos podem ser vistos e analisados através de diferentes visões pela ferramenta Poset Browser, chamada Pov (*The Rapide Order Viewer*). A Figura 3.6 representa a tela do *Poset Browser* da arquitetura apresentada na Figura 3.4.

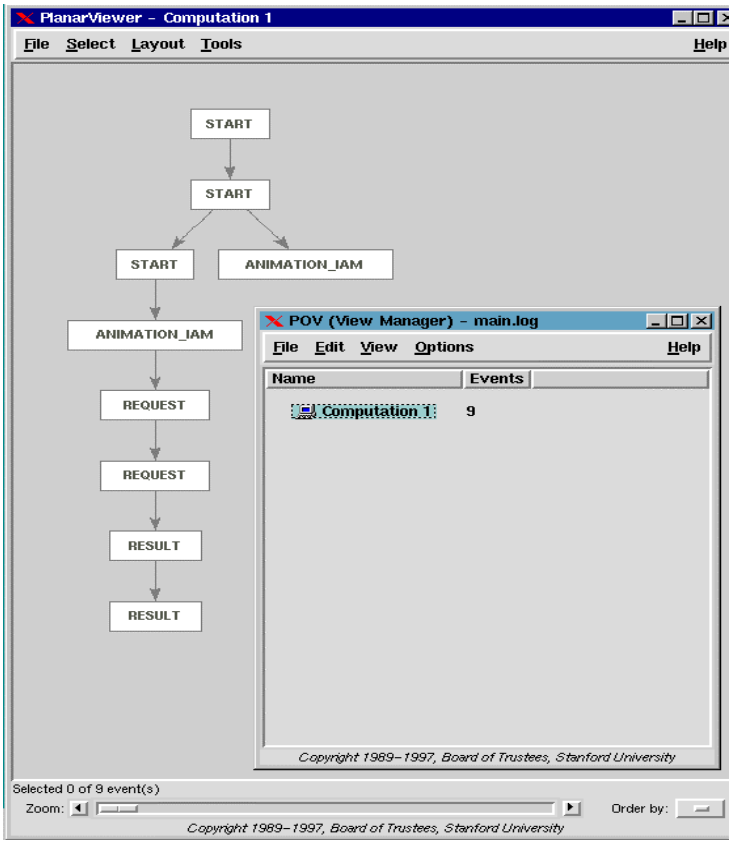

FIGURA 3.6 – Poset Browser

3.5.3.6 Exemplo

A seguir é apresentado o código Rapide da arquitetura de sistema da Figura 3.4. Neste exemplo, existem dois tipos de interfaces, APPLICATION_PROGRAM e RESOURCE. Eles definem as interfaces dos objetos application program e resource. Estas interfaces definem as características assíncronas de comunicação chamada *ações (actions)*. A interface APPLICATION_PROGRAM contém duas ações, uma ação **out** Request e uma ação **in** Result. Objetos deste tipo podem gerar eventos Request e podem receber eventos Result. A interface APPLICATION_PROGRAM também contém regras de transição de comportamento reativa. De acordo com a regra de comportamento, espera-se até que um evento Start seja observado e reage gerando dois eventos: um animation_Iam e um evento **out**, Request. O evento animation_Iam é parametrizado com a string "APPLICATION_PROGRAM". O tipo interface RESOURCE é similar. Ele pode receber um evento Request e seu comportamento é para reagir gerando o evento animation_Iam e um evento **out,** Result.

A arquitetura MAIN contém dois componentes, AP e R, cada um de um tipo de interface. Conexões entre estes dois componentes são definidas pelas regras de conexão reativa. A primeria regra inicia sempre que AP gera um Request e reage gerando um evento Request de R. Então, ele conecta a ação **out** Request do componente AP e a ação **in** Request do componente R. A segunda regra conecta a ação **out** Result do componente R com a ação **in** Result do componente AP. A Figura 3.7 apresenta o código-fonte gerado a partir da arquitetura apresentada na Figura 3.4.

```
TYPE APPLICATION_PROGRAM IS INTERFACE
action
  out Request();
  In Result();
            BEHAVIOR
  action animation_Iam(name : string);
            BEGIN
    start => animation_Iam("APPLICATION_PROGRAM") -> Request();;
END;

TYPE RESOURCE IS INTERFACE
action
  out Result();
  in Request();
            BEHAVIOR
  action animation_Iam(name : string);
            BEGIN
    start => animation_Iam("RESOURCE");;
    Request() => Result();;
END;
ARCHITECTURE MAIN() IS
       AP : APPLICATION_PROGRAM;
       R  : RESOURCE;
CONNECT
       AP.Request() => R.Request();
       R.Result() => AP.Result();
END.
```

FIGURA 3.7 – Código-fonte gerado a partir de uma arquitetura.

Dessa maneira, a arquitetura define a comunicação entre os dois componentes em termos das ações em suas interfaces.

3.5.4 Outras ADLs Disponíveis

Em [FLA 99], é apresentada uma breve caracterização sobre algumas ADLs disponíveis conforme segue:

- **SRI's SADL:** a linguagem de descrição arquitetural SADL [STA 2001] está planejada para definir hierarquias de arquitetura de software que serão analisadas formalmente. A linguagem de SADL pode ser usada para especificar a estrutura e a semântica de uma arquitetura, mas o foco principal está na estrutura. As características interessantes em SADL incluem o suporte ao mapeamento explícito entre arquiteturas, arquiteturas genéricas, estilos arquiteturais (incluindo restrições bem definidas) e padrões de refinamento de arquitetura que provêm soluções para problemas comuns de projeto. SADL é uma linguagem de programação independente, mas pode ser adaptada para modelar programas em linguagens de programação convencionais.

- **Carnegie Mellon UniCon:** UniCon [SHA 95] é uma linguagem de descrição arquitetural cujo foco está em apoiar a variedade de estilos arquiteturais disponíveis no mundo real e em construir sistemas de descrições de arquitetura desses estilos. Uma descrição de arquitetura em UniCon consiste em um conjunto de componentes e conectores. Um componente é um conjunto de dados

ou computação, enquanto um conector media a interação entre componentes. Cada componente tem uma interface que exporta um conjunto de atividades. Estas atividades definem os modos nos quais um componente pode interagir com o mundo externo. Da mesma forma, o protocolo de um conector exporta um conjunto de funções que definem as maneiras nas quais um conector pode mediar interação. A versão atual da linguagem UniCon suporta o estilo arquitetural de *pipes-and-filters* e módulos que interagem com chamadas de procedimento e dados compartilhados, sistemas distribuídos com chamadas de RPC (*Remote Procedure Call*), processos que compartilham processadores e acesso a banco de dados através de comandos SQL (*Structured Query Language*).

- **Carnegie Mellon ACME:** o projeto ACME [ACM 2001] teve início em 1995 com o objetivo de prover uma linguagem comum que poderia ser utilizada para apoiar o intercâmbio de descrições arquiteturais entre uma variedade de ferramentas de projeto. Embora ainda fosse útil como uma ferramenta de intercâmbio arquitetural, a linguagem ACME e seu *tollkit* ganharam espaço apoiando-se na idéia de que novas ferramentas de projeto e análise da arquitetura de software podem ser construídas sem a necessidade de se reconstruir a infraestrutura padrão. Atualmente, a linguagem de descrição arquitetural ACME e o *Acme Tool Developer's Library* (AcmeLib) fornecem uma infraestrutura genérica e extensível para descrever, representar, gerar e analisar descrições de arquiteturas de software.

- **SCG's Piccola [PIC 01]:** é uma linguagem experimental de composição. É apresentada como uma linguagem principal e uma camada de abstração sobre esta linguagem. Uma linguagem de composição suporta a construção de aplicações através de simples ligações entre componentes. Essas aplicações evoluem através de alterações nos componentes e/ou nos conectores.

Em análise recente [MED 2000], pesquisadores da área de arquitetura de software descreveram comparações sobre as diversas ADLs existentes. De acordo com os interesses deste trabalho, foi escolhida entre essas ADLs, a linguagem Rapide, por ser caracterizada como uma linguagem de descrição arquitetural com finalidade geral, que permite modelar interfaces de componentes e seus comportamentos externamente visíveis, e ainda por ser uma linguagem que permite simulação e modelagem do comportamento dinâmico descrito por uma arquitetura.

3.6 Abordagens Existentes de Linha de produto

Esta seção apresenta uma descrição de algumas abordagens que suportam o enfoque de linha de produto. A literatura técnica apresenta algumas dessas abordagens e estratégias. Algumas abordagens apresentam uma solução abrangente para as questões relacionadas à representação de aspectos ligados à engenharia do domínio e aplicação bem como aos conceitos inerentes a evolução, gerenciamento, análise da relação custo e benefício, tomadas de decisão orientada ao mercado e avaliação de riscos. Outras fornecem métodos concentrados em alguns dos aspectos inerentes a tecnologia de linha de produto, tais como a definição da família de produtos, construção da infra-estrutura

arquitetural básica ou mesmo a implementação de componentes reutilizáveis. Neste caso, elas estão usualmente integradas a algum outro método específico de engenharia de software como, por exemplo, análise de *features*, desenvolvimento baseado em componentes ou mesmo a utilização da notação UML ao longo de um processo de desenvolvimento de software.

Algumas abordagens existentes são a *Synthesis* que foi documentada pelo *Software Productivity Consortium* [SPC 93], *Family-Oriented Abstraction, Specification and Translation* (FAST) [WEI 99], *Product Line Software Engineering* (PuLSE) do Centro de Fraunhofer [BAY 99], *Feature-Oriented Domain Analysis* (FODA) documentado pelo SEI [KAN 90], a iniciativa PLP (Product Line Practice) do SEI [CLE 2001] e a abordagem proposta por Bosch [BOS 2000]. A seguir uma breve caracterização destas abordagens é apresentada conforme descrito em Gimenes e Travassos [GIM 2002]:

3.6.1 A Abordagem *Synthesis*

A abordagem *Synthesis* foi elaborada pelo *Software Productiviy Consortium* e descreve uma metodologia abrangente para a construção de sistemas de software representando instâncias de uma família de sistemas que possuem discrições similares. Suas características principais são:

- Formalização de domínios como famílias de sistemas que compartilham várias características em comum, e que também variam em formas e configurações bem definidas;
- Redução da construção de sistemas à resolução de requisitos com decisões de engenharia representando as características de variações existentes no domínio;
- Reutilização de artefatos de software através da adaptação mecânica dos componentes para satisfazer aos requisitos e as decisões de engenharia;
- Análise baseada na descrição do modelo do sistema de forma a auxiliar a compreensão das implicações de decisões de construção para com o sistema, permitindo assim avaliar as possíveis alternativas de construção.

Quatro princípios básicos norteiam a aplicação de *Synthesis*:

- Família de programas: especificando que desenvolvedores deveriam construir programas de software não como artefatos únicos, mas como instâncias de uma família de programas similares;
- Processo iterativo: considerando o desenvolvimento de produtos de software como uma repetição de atividades de produção e utilização;
- Especificações: descrevendo as propriedades verificáveis a serem consideradas para um produto membro ou um conjunto de produtos membro adaptáveis e também para derivar instâncias de cada produto membro e produzir um sistema produto em particular.

Essencialmente, *Synthesis* descreve os aspectos principais do desenvolvimento de uma linha de produto de software, comuns a todas as metodologias relacionadas a este tópico.

3.6.2 FAST: Family-Oriented Abstraction, Specification and Translation

A análise de características comuns é um aspecto marcante da utilização de FAST. Esta análise serve para identificar o contexto, descrever o domínio, fornecer um conjunto padrão de termos técnicos chave, identificar características comuns e variabilidades, quantificar as variabilidades através do fornecimento de parâmetros de variação, identificar e registrar várias informações úteis que aparecem durante a análise [GIM 2002].

A análise de características comuns é realizada por meio de uma série de reuniões com especialistas de domínio, facilitadas por um moderador. A equipe de análise produz o documento em grupo durante a reunião, decidindo sobre seu conteúdo por consenso e guiada pelo moderador. Tipicamente, cada especialista é especializado em um ou mais aspectos do domínio, enquanto o moderador deve possuir experiência no processo de análise de características comuns e possui habilidade para reconhecer quando as definições estão precisas, claras e bem formadas, identificar características comuns, variabilidades, parâmetros de variação e questões úteis que deverão compor o documento.

A análise de características comuns é organizada em cinco estágios: preparação, planejamento, análise, quantificação e revisão externa. A Figura 3.8 apresenta a estrutura do processo de análise de características comuns de FAST.

No estágio de preparação, o moderador assegura que todos os recursos necessários estão disponíveis para as seções seguintes. O moderador prepara um modelo de documento para registrar a definição e descrição dos termos, características comuns, variabilidades e outras informações de saída do processo de análise de características comuns. Além do esqueleto do documento, algumas definições iniciais de alguns termos e conceitos, incluindo características comuns à variabilidade, são preparadas e incluídas no documento permitindo iniciar a discussão.

O estágio de planejamento começa com uma breve revisão dos conceitos do processo de análise de características comuns. Então, a equipe revisa os documentos iniciais, os aprova e determina as fronteiras do domínio. Durante a execução dos estágios seguintes do processo de análise de características comuns estas informações são usualmente revisadas, visando definir a família de produto e tornar mais explícitas suas interfaces com outros domínios.

Durante o estágio de análise a equipe gera os termos técnicos e suas definições, as características comuns e variabilidade, identificando, ainda, as questões relacionadas ao domínio. Existe uma relação estreita entre a definição de termos e a descoberta de características comuns. Normalmente, os termos são identificados através da utilização da relação "*é um"*. As características comuns normalmente são identificadas sob a forma "*tem atributo"*. Além dos atributos comuns, alguns produtos membro usualmente possuem um conjunto de atributos especiais ou opcionais representando as variabilidades. Existem dois tipos de variabilidade: funções diferentes executadas por produtos membro e possíveis modificações de alguns produtos membro que devem ser

realizadas no futuro. Questões normalmente representam termos, características comuns ou variabilidades com informação insuficiente para sua perfeita identificação.

Para cada questão, um número de alternativas é estabelecido. Então a questão é associada a um dos membros da equipe para investigação adicional. O estágio termina quando nenhum novo termo, característica comum ou variabilidade possa ser descoberto e a descrição das características já identificadas não sofrem modificações durante as reuniões.

O estágio de quantificação consiste na geração do parâmetro de variação. Um parâmetro de variação é uma visão quantificada da variabilidade especificando a faixa de valores e o momento em que deve ser utilizada para tomada de decisão. Para cada variabilidade a equipe define o tipo de decisão que ela representa, quantifica a faixa de valores, especifica o tempo de ligação, e fornece um valor *default,* se existir algum, para a tomada de decisão do projeto.

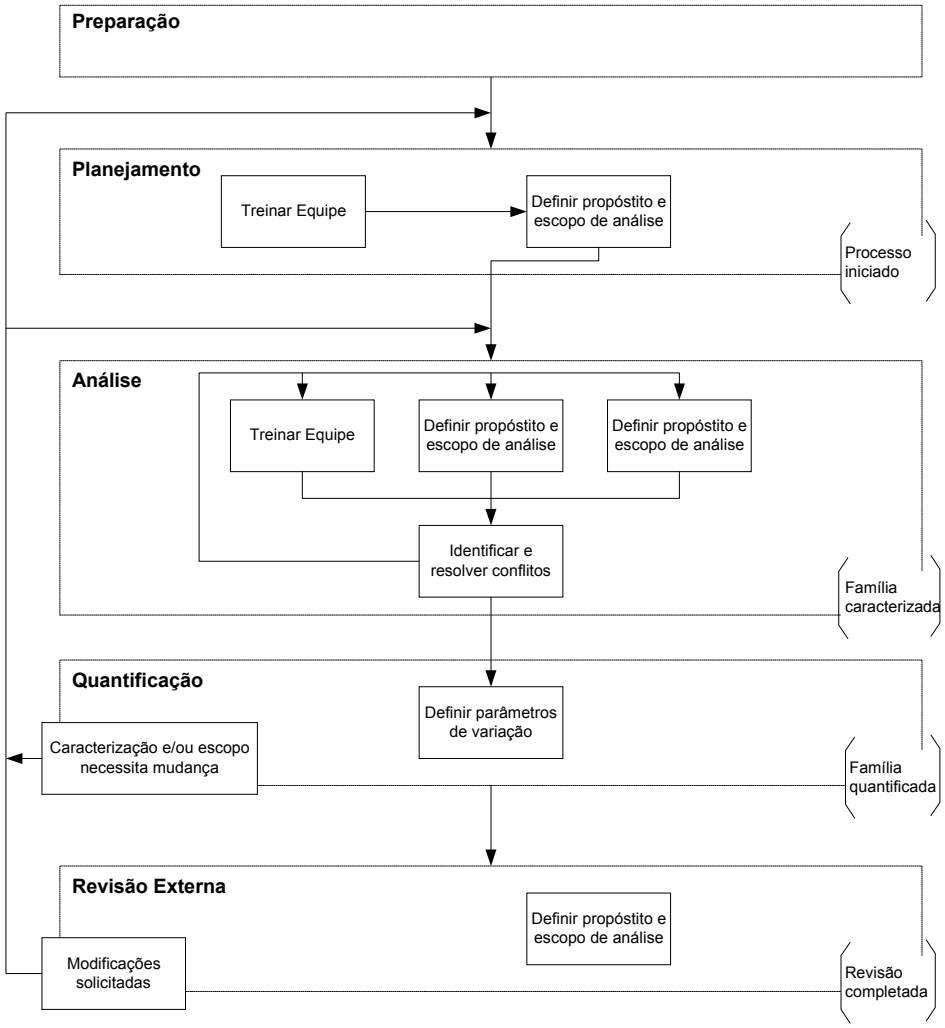

FIGURA 3.8 – Estrutura do Processo de Análise de Características comuns de FAST [TRA 2002]

A revisão externa tem por objetivo primariamente descobrir hipóteses e definições incorretas ou omitidas, e inconsistências na analise de características comuns. Além disso, ela revela a qualidade externa do documento, já que diferentes especialistas

do domínio ou usuários não envolvidos no processo de análise estarão utilizando o documento e avaliando sua clareza, legibilidade e compreensão.

O processo de análise de características comuns do FAST começou como um projeto experimental na Lucent em 1992, e ainda se encontra em desenvolvimento. Algumas questões relacionadas à alocação de recursos, atributos que caracterizam um bom moderador, definição de trabalhos para os membros da equipe que não foram discutidos aqui podem ser encontrados na literatura [WEI 99].

3.6.3 PulSE: Product Line Software Engineering

PulSE [BAY 99] é uma metodologia de construção e utilização de linhas do produto. A estrutura geral de PuLSE inclui fases de desenvolvimento (iniciação, construção de infra-estrutura, utilização da infra-estrutura e evolução e gerenciamento), componentes técnicos (PuLSE-Eco utilizado para identificação de escopo para um membro da linha de produto; PuLSE-CDA contém análise de domínio e elaboração do modelo de decisão que será utilizado para a criação da arquitetura da linha de produto; PuLSE-DSSA suporta a definição de arquitetura específica de domínio o que implica no desenvolvimento incremental da estrutura de linha de produto guiado por cenários; PuLSE-I executa a instanciação do produto na fase que o utiliza, e PuLSE-EM que lida com gerenciamento de configuração e questões, relacionadas à evolução do produto), e componentes de suporte (Pontos de entrada de projetos que customizam PuLSE para tipos maiores de projeto; Escala de maturidade que fornece caminhos para integração e evolução; e, questões organizacionais que fornecem diretrizes para configuração e manutenção da estrutura de organização). A metodologia PuLSE serve como base para vários projetos industriais e científicos, como por exemplo, KobrA descrito nesta seção.

PuLSE-Eco (*Economic Scoping*) é uma forma de identificação e representação do contexto econômico de uma linha de produto que compreende as seguintes atividades:

- Determinação antecipada de produtos;
- Mapeamento e avaliação das características dos produtos membro;
- Determinação das melhores características e melhores candidatos para uma linha de produto, e
- Caracterização dos candidatos escolhidos e fornecimento da análise de beneficio para cada candidato.

O resultado apresenta o contexto econômico da linha de produto que consiste de uma lista de características significantes para o domínio em questão e um mapa do produto. Este mapa é uma tabela que fornece uma classificação de produtos membro existentes, futuros e potenciais de acordo com as características que eles possuem.

PuLSE-CDA (*Customizable Domain Analysis*) inicialmente refina o contexto econômico para especificar os limites da linha de produto. Para tal, ele utiliza uma técnica de modelagem chamada *storyboards.* Os *storyboards* capturam tipos de

seqüência de ações relevantes para o domínio. Estes tipos variam para diferentes domínios. Exemplos são diagramas de *workflow* e diagramas de seqüências de mensagens. Esta técnica de modelagem presume a elicitação da informação sobre sistemas simples em *storyboards* não formatados e conseqüente consolidação deste conhecimento em *storyboards* genéricos. Dessa forma, o componente PuLSE-CDA captura as variabilidades. Para derivar a especificação de produtos membro de *storyboards* genéricos, um modelo de decisão é criado contendo um conjunto estruturado de decisões. Cada decisão corresponde à variabilidade no *storyboard,* juntamente com um conjunto de possíveis soluções [GIM 2002].

PuLSE-DSSA implica numa especificação incremental da arquitetura guiada por cenários. PuLSE-DSSA categoriza cenários como genéricos (descrevendo requisitos funcionais) ou relacionadas à propriedade (descrevendo aspectos de qualidade independente do domínio) Os cenários genéricos são derivados do *storyboard* genérico que são representações de modelo de domínio em PuLSE, e juntamente com o modelo de decisão, compõem o resultado final da fase PuLSE-CDA. O desenvolvimento da arquitetura começa com um conjunto inicial de cenários genéricos que é utilizado para criar a estrutura inicial. Em seguida, um conjunto de cenários genéricos é escolhido de acordo com sua importância para a arquitetura e aplicados à estrutura da arquitetura inicial. Dessa forma, é possível o refinamento da estrutura da arquitetura inicial e sua extensão para que suporte todos os cenários genéricos, evoluindo assim para seu estágio final.

A aplicação dos cenários genéricos pode resultar em mais de uma alternativa de arquitetura. Neste caso, os cenários são relacionados às propriedades e adicionados aos cenários genéricos corretamente utilizados, são aplicados e classificados visando avaliar a aderência do candidato à estrutura arquitetural. O resultado desta classificação é um candidato a fazer parte da estrutura arquitetural e que deverá ser estendido.

O modelo de decisão produzido durante a fase PuLSE-CDA é também melhorado no contexto PuLSE-DSSA. Algumas decisões específicas de implementação são colecionadas enquanto a estrutura arquitetural está evoluindo. Estas decisões terão que ser resolvidas durante a instanciação da estrutura arquitetural com o propósito de criação de uma arquitetura para o produto membro. Capturando estas decisões ao longo do processo com suas possíveis resoluções formam um modelo de configuração que estende o modelo de decisão.

3.6.4 FODA: *Feature Oriented Domain Analysis*

O método de análise de domínio FODA [KAN 90] foi criado no contexto dos projetos de pesquisa do SEI, tornado-se um dos métodos de análise de domínio mais populares no início da década de 90.

Os autores descrevem o FODA como um método para dar suporte à reutilização em nível arquitetural e funcional. No entanto, os estudos apresentados na literatura descrevem principalmente a modelagem de estudos de caso na fase de análise, dando ênfase à modelagem funcional, mas não apresentando com detalhes a parte arquitetural e nem as ligações entre os diversos modelos gerados.

A abordagem adotada pelo FODA é bastante interessante do ponto de vista do desenvolvimento baseado em componentes (DBC), uma vez que a captura das funcionalidades relevantes ao domínio auxilia na identificação de que componentes,

disponibilizando quais funcionalidades mais adequadas, estariam disponíveis no domínio. Outro ponto interessante, mas que não é muito ressaltado nos estudos de caso conduzidos, é a possibilidade de modelagem de características adicionais ao domínio como a modelagem dos ambientes operacionais disponíveis e as técnicas de implementação. Estas informações auxiliam o desenvolvedor na seleção das funcionalidades mais adequadas, levando-se em consideração o ambiente operacional e as questões implementacionais.

Segundo Werner [WER 2000], uma crítica que pode ser feita ao método FODA é que por ser a modelagem centrada na captura das funcionalidades do domínio, todos os modelos utilizados carregam esta característica. O método é dito ser muito preocupado com a geração de código, e não se preocupa com a devida ênfase na captura de todas as abstrações do domínio. A preocupação é somente com características funcionais das aplicações no domínio. Características ditas não codificáveis acabam sendo descartadas na modelagem do FODA. Estas características não codificáveis também são importantes no contexto do domínio, principalmente para facilitar o entendimento dos conceitos e assim, conseqüentemente, facilitar o entendimento dos componentes, promovendo de forma mais eficiente a reutilização de componentes no desenvolvimento de aplicações no domínio. Além disso, este método não apresenta um detalhamento maior em relação a componentes reutilizáveis. O método se atém à modelagem das características (*features*), mas não especifica como criar componentes.

3.6.5 A Iniciativa PLP (Product Line Practice) do SEI/CMU

Um outro exemplo de estratégia que pode ser utilizada para implementação de linha de produto é a estrutura para prática de linha de produto de software (PLP) desenvolvida pelo *Software Engineering Institute* [CLE 2001]. Neste contexto, o desenvolvimento de uma linha de produto envolve atividades relacionadas ao desenvolvimento de artefatos centrais e desenvolvimento e gerenciamento do produto utilizando estes artefatos centrais. Para executar estas atividades torna-se necessária a definição de áreas de trabalho que representam conjuntos menores de atividades, porém mais gerenciáveis. Cada área de trabalho (*Practice Area*) possui um plano de trabalho e métricas associadas que permitem acompanhar sua execução e avaliar o sucesso dos trabalhos realizados. Usualmente estas áreas de trabalho produzem artefatos concretos que levam a criação, de alguma forma, dos artefatos centrais que serão reutilizados na linha de produto. É uma descrição de abordagem similar a do CMM (*Capability Maturity Model*). Basicamente, a estratégia PLP define três áreas de trabalho [GIM 2002]:

- Engenharia de Software: necessária para aplicar a tecnologia apropriada à criação e evolução dos artefatos ou componentes centrais e do produto.

- Gerenciamento Técnico: relacionada à criação e evolução dos artefatos ou componentes centrais.

- Gerenciamento Organizacional: utilizada para o gerenciamento dos esforços relacionados à linha de produto como um todo.

A versão corrente da iniciativa PLP define todas as atividades para áreas de trabalho e vem evoluindo continuamente através das experiências obtidas de diferentes estudos de caso realizados com a colaboração extensa existente entre o SEI e as organizações que estudam a utilização de linha de produto, além de informações recebidas da comunidade de software.

3.6.6 A Abordagem Bosch

Bosch [BOS 2000] considera uma linha de produto em três dimensões que estão organizadas em três conjuntos de elementos, a saber, (1) arquitetura, componente e sistema; (2) negócios, organização, processo e tecnologia; (3) desenvolvimento, aplicação e evolução.

A primeira dimensão divide o domínio da linha de produto no que diz respeito aos artefatos iniciais que são parte do desenvolvimento baseado em reutilização. Esta abordagem é adotada por [JAC 97]. Os artefatos principais são resumidos a seguir.

- **Arquitetura:** o primeiro artefato de uma linha de produto é a arquitetura de software. O objetivo é projetar uma arquitetura que cubra todos os requisitos dos produtos da linha de produto e inclua características que possam ser compartilhadas entre esses produtos.
- **Componentes:** o segundo conjunto de artefatos de uma linha de produto contém os componentes identificados para a arquitetura. A arquitetura de linha de produto identifica os componentes e a variabilidade requerida para esses. Segundo Bosch [BOS 2000], componentes tendem a ser entidades relativamente grandes, chegando a ter mais de cem mil linhas código sendo mais relacionados a *frameworks* orientados a objetos do que a classe nos sistemas tradicionais.
- **Sistemas:** o conjunto final de artefatos compreende os sistemas construídos e baseados na arquitetura de linha de produto e seus componentes. A construção do sistema requer uma adaptação da arquitetura de linha do produto para adequar a arquitetura no sistema. Este processo pode requerer a adição ou remoção de relacionamentos, bem como a adição ou remoção de relacionamentos entre componentes de arquitetura. Pode ainda ser necessário o desenvolvimento de extensões específicas do sistema para os componentes restantes, a configuração dos componentes do sistema e o desenvolvimento de código específico para esses componentes.

A segunda dimensão, de acordo com que se pode ver a linha de produto está relacionada com as diferentes visões de uma organização. Bosch [BOS 2000], relata que os *workshops* do SEI utilizam essas visões para descrever questões sobre linha de produto. Estas questões são discutidas a seguir com mais detalhes:

- **Negócio:** Todas as atividades de uma organização devem de alguma, ser justificadas em termos de benefícios de negócio. Com respeito às linhas de produto, um importante aspecto é que geralmente se requer um investimento

considerável para se converter de uma abordagem orientada a um produto para uma abordagem de desenvolvimento orientada à reutilização e baseada em linha do produto [BOS 2000].O projeto da arquitetura de linha do produto e o desenvolvimento dos componentes geralmente são caros, e o mais importante, normalmente atrasa o tempo inicial de comercialização dos produtos.

- **Organização:** a abordagem tradicional de linha de produto sugere o uso de um domínio que cria um departamento responsável pelo desenvolvimento e evolução da linha de produto e dos componentes reutilizáveis. Abordagens alternativas não empregam necessariamente um domínio que cria unidades, mas geralmente têm efeitos em toda organização. Por exemplo, componentes reutilizáveis são desenvolvidos por projetos de engenharia de domínio e pessoas que se envolveram nesse processo. Assim, a mudança da abordagem tradicional de desenvolvimento para o desenvolvimento de software baseado em reutilização também cria efeitos organizacionais, que necessitam ser tratados.
- **Processos:** a terceira questão diz respeito aos processos orientados a reutilização de uma organização. O fato de se ter disponível uma arquitetura de linha de produto e um conjunto de componentes implica em efeitos consideráveis no processo associado a um produto ou ao desenvolvimento de um sistema. Existem também processos específicos de linha de produto relacionados ao projeto da arquitetura de linha de produto e ao desenvolvimento dos componentes reutilizáveis.
- **Tecnologia:** a questão final de uma organização orientada a reutilização está relacionada com a tecnologia utilizada para apoiar o desenvolvimento de software baseado em reutilização. Bosch [BOS 2000] emprega a noção de tecnologia em sentido amplo, incluindo os artefatos da organização, como os componentes reutilizáveis, por exemplo. A tecnologia ou técnicas utilizadas no desenvolvimento de componentes também está incluída. Exemplo disso são os *frameworks* orientados a objetos e os padrões de projeto.

A terceira dimensão pela qual se pode descrever a linha de produto de software está baseada no ciclo de vida de cada um dos artefatos da organização e envolve desenvolvimento, aplicação e evolução. Em [BOS 2000], o autor enfatiza esta dimensão como sendo preliminar porque representa as fases iniciais para que uma organização adote a abordagem baseada em linha de produto.

Como resultado dos processos de desenvolvimento, aplicação e evolução, vários artefatos são produzidos. A Figura 3.9 apresenta uma visão geral desses artefatos.

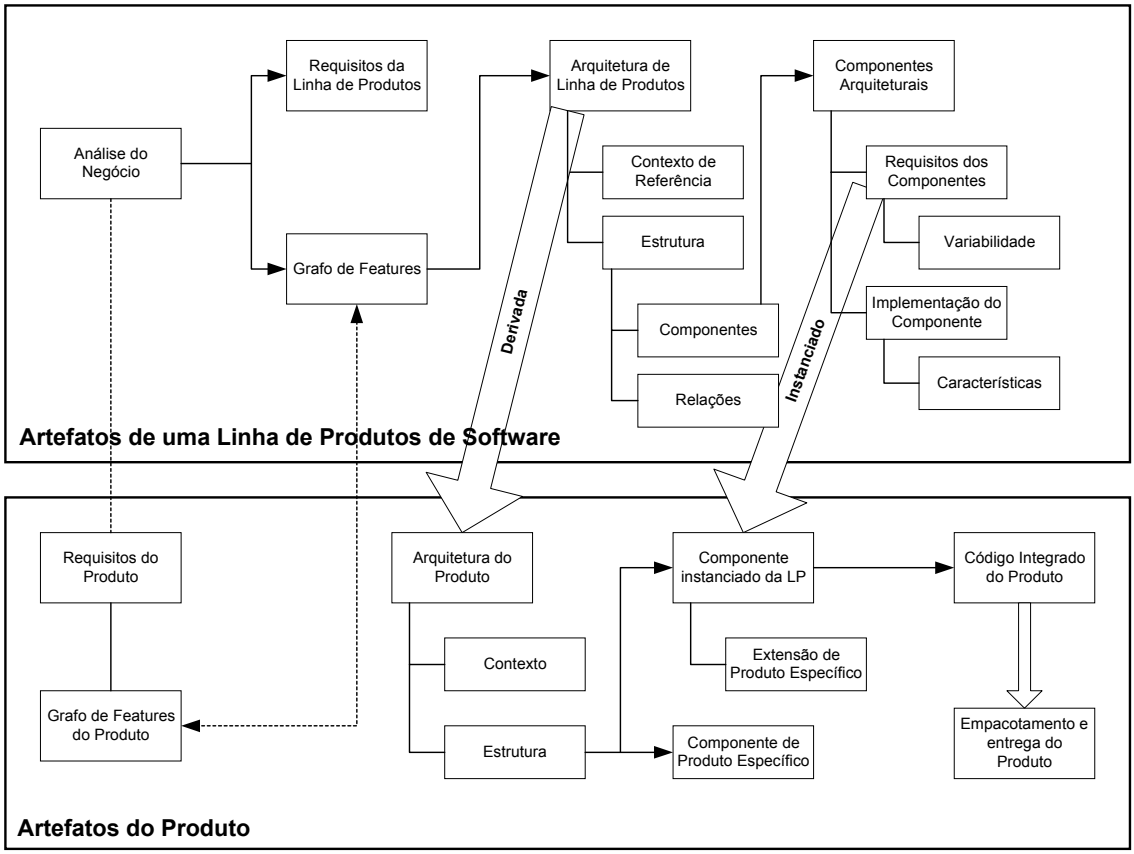

FIGURA 3.9 – Artefatos de uma linha de produto de Software [BOS 2000]

Os artefatos são divididos em duas categorias principais; os artefatos que são compartilhados pelos membros da linha de produto e os artefatos de produtos específicos. Os artefatos compartilhados entre produtos e linha de produto são conseqüência da análise do negócio e dos requisitos da linha de produto. Além disso, os artefatos devem incluir a arquitetura de software em termos dos componentes e suas relações.

A arquitetura de linha de produto define a associação dos componentes arquiteturais com os requisitos desses componentes e suas possíveis implementações. Cada implementação de componente possui características associadas, descrevendo semelhanças com produtos particulares [LAZ 2001].

Os artefatos de produtos específicos constituem dos requisitos desses produtos e de um grafo de *features* do produto. Além disso, define-se a arquitetura para um produto derivado da arquitetura de linha de produto, incluindo a descrição do contexto e a estrutura.

De acordo com a arquitetura, as implementações de componentes instanciados da linha de produto de software podem ser configuradas por extensões de produtos específicos.

3.6.7 Outras Abordagens

Kobra [ATK 2000] é um método que integra os conceitos de linha de produto e desenvolvimento baseado em componentes. Os princípios de construção e utilização de linhas de produto são baseados na metodologia PulSE. KobrA considera o desenvolvimento e manutenção de *Framework* o que inclui todas as alternativas possíveis de realização da arquitetura de produtos membro. O conceito central do método é um componente KobrA (*Komponent*) descrito em diferentes níveis de abstração e organizados em forma de uma árvore. Cada *Komponent* é descrito em dois níveis de abstração – uma especificação definindo as propriedades externas visíveis de um *Komponent* e comportamentos, e uma realização descrevendo como um Komponent preenche suas responsabilidades quando relacionados a outros *Komponents* de mais baixo nível. A descrição da especificação e realização de um *Komponent* é realizada através da utilização de diagramas UML.

A especificação de um *Komponent* é derivada da estrutura arquitetural da linha de produto e fornece a definição da interface do *Komponent* utilizando quatro modelos básicos: estrutural, comportamental, funcional e decisão. Estes modelos representam respectivamente o estado do *Komponent*, sua reação a estímulos externos, seu impacto sobre entidades externas e a natureza de trocas no comportamento do *Komponent* dependendo do ambiente.

A realização do Komponent é também representada por quatro modelos principais: interação, estrutural, atividade e decisão. Estes modelos são relacionados aos modelos de especificação, mas ao contrário deles, representam como um Komponent realiza sua funcionalidade ao invés de mostrar o que é a funcionalidade. Se a interface especificada de um Komponent está de acordo com a interface fornecida por um componente pré-existente, como por exemplo, COTS ou algum sistema legado reestruturado, ele poderia ser utilizado normalmente com algum ajuste [GIM 2002].

O método KobrA está sendo correntemente avaliado num contexto industrial através do desenvolvimento de um estudo de caso para o domínio de Planejamento de Recursos Organizacionais.

GenVoca [BAT 98] é uma metodologia baseada nos conceitos de máquina virtual, camadas de componentes (representando uma implementação da máquina virtual) e arquitetura-*realm* (um conjunto de componentes do mesmo tipo). GenVoca considera também os mecanismos de integração e operação para esses conceitos os quais incluem equações de tipos, regras de projeto, gramáticas e geradores. A utilização da metodologia GenVoca tem mostrado algumas vantagens como por exemplo, melhor desempenho da aplicação produzida, aumento de produtividade e redução de erros de desenvolvimento em comparação com técnicas de desenvolvimento convencionais. Entretanto, existem ainda algumas barreiras técnicas e não-técnicas para as quais os autores da metodologia intencionam resolver futuramente.

A tecnologia de linha de produto vem sendo aplicada com sucesso em diferentes projetos na indústria. Por exemplo, [CZA 99] aplicaram a tecnologia de linha de montagem para a criação de alguns sistemas em escala real no domínio de bibliotecas para tratamento de matrizes, ou mesmo o programa STARS (*Software Technology for Adaptables, Reliable Systems*) vem sendo desenvolvido de acordo com a tecnologia de linha de produto.

3.7 Considerações Finais

Este capítulo apresenta conceitos importantes para o contexto deste trabalho. Apresentou os conceitos básicos relacionados à linha de produto de software, apresentando seus elementos. Caracterizou ainda as três atividades essenciais da construção de uma linha de produto, especialmente o desenvolvimento do núcleo de artefatos de uma linha de produto, o qual se insere o escopo deste trabalho. Apresentou conceitos de arquitetura de software e desenvolvimento baseado em componentes, necessários ao processo de desenvolvimento da arquitetura de linha de produto para WfMS.

O método de DBC Catalysis será utilizado para guiar o processo obtenção da arquitetura de linha de produtos proposta, porém serão necessárias algumas modificações para representação de variabilidade. Catalysis foi escolhido por ser um método geral de DBC que envolve conceitos de arquitetura de software, *frameworks* e padrões. Assim, o objetivo é utilizar um método de propósito geral, examinando as alterações necessárias ao invés de se criar um método específico para linha de produtos. As abordagens de engenharia de domínio fornecem recursos para representação de variabilidade, porém são mais deficientes na representação e implementação de arquiteturas de componentes. Assim, os métodos de DBC surgem como uma boa solução para as fases de projeto e implementação da linha de produto. Iniciativas similares já podem ser encontradas na literatura, tais como Kobra [ATK 2000] e GenVoca[BAT 98].

Este capítulo incluiu também uma caracterização sobre ADL's, especialmente a ADL Rapide, que foi selecionada para descrever e validar a arquitetura proposta. Esta ADL foi escolhida por permitir a simulação da arquitetura, permitindo avaliar a comunicação entre os componentes da arquitetura e o efeito geral das funcionalidades do sistema, sem que houvesse a necessidade de aprofundamento em detalhes de implementação.

Por fim, a última seção apresentou um *overview* das principais abordagens para linha de produto existente atualmente.

Outros trabalhos importantes relacionados a este são: a padronização de arquiteturas de software com UML da OMG [OMG 2002] e as investigações dedicadas a articular o relacionamento entre especificações de requisitos com o projeto da arquitetura de software [CAJ 2001].

O capítulo seguinte apresenta o processo seguido para a definição da arquitetura de linha de produto para WfMS proposta. Apresenta também as extensões necessárias para representar variabilidade no processo de desenvolvimento. As fases de desenvolvimento do produto e gerenciamento da linha de produto estão fora do escopo deste trabalho.

4 Uma Arquitetura de Linha de Produto para Sistemas de Gerenciamento de *Workflow*

Este capítulo apresenta uma arquitetura de linha de produto para sistemas de gerenciamento de *workflow*, o processo seguido para sua definição e a notação utilizada para a representação de suas variabilidades.

No capítulo 3, foram apresentadas algumas abordagens existentes para a especificação de linhas de produto, contudo, observa-se uma carência de técnicas que venham a facilitar o processo de desenvolvimento da arquitetura e de seus componentes. O processo proposto neste trabalho envolve a exploração do domínio por meio do modelo de referência e da arquitetura genérica para WfMS da WfMC, o padrão *Process Manager* [GIM 99b], o método Catalysis [D'SO 99] e a linguagem Rapide [CSL 97] para simulação e avaliação da arquitetura. O método Catalysis foi escolhido porque é uma abordagem geral de desenvolvimento baseado em componentes, que envolve conceitos de arquitetura de software, *frameworks* e padrões. Catalysis é baseado em UML [BOO 99] [FOW97] [RUM 99]. Assim, o objetivo é aplicar um método de propósito geral, examinando as alterações necessárias para linha de produto ao invés de se criar um método específico para esta abordagem. Além disso, as abordagens de engenharia de domínio fornecem recursos para representação e gerenciamento de variabilidade, porém são mais deficientes na representação e implementação de arquiteturas de componentes.

O processo proposto é composto das seguintes fases: análise dos requisitos, especificação do sistema, projeto da arquitetura, projeto interno dos componentes e validação da arquitetura. As seções 4.2, 4.3, 4.4 e 4.5 apresentam uma caracterização dessas fases, bem como apresentam os artefatos gerados em cada uma delas.

Antes de iniciar a descrição das fases do processo, a seção 4.1 apresenta os mecanismos utilizados neste trabalho para representação de variabilidades, uma vez que o método Catalysis não oferece estes recursos.

4.1 Extensões Necessárias para Representar Variabilidade

A primeira necessidade de representação de variabilidade ocorre na fase de análise de requisitos que é representada por meio de casos de uso. Neste caso, a notação seguida foi a proposta por Jacobson et al [JAC 97] que sugeriu 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 variabilidade. Como exemplo, pode-se observar na Figura 4.1 o caso de uso Agendar Aula que possui um ponto de variação, que representa neste caso duas possíveis extensões: Agendar Aula Teórica e Agendar Aula Prática. Na seção 4.2, os diagramas de caso de uso elaborados na fase de análise de requisitos seguem esta representação.

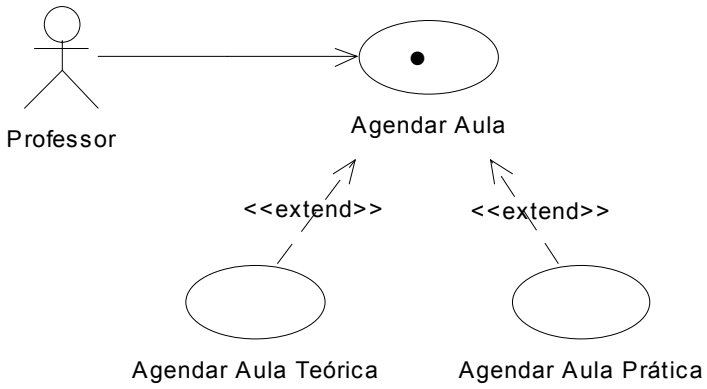

FIGURA 4.1 – Exemplo de representação de variabilidade em casos de uso.

A segunda representação de variabilidade seguida neste trabalho foi a proposta por Morisio et al [MOR 2000], que estenderam a notação UML com um estereótipo de variabilidade, indicado pelo rótulo <<V>> significando que zero ou mais classes do conjunto podem estar presentes em uma determinada aplicação. Como o estereótipo <<V>> foi usado no contexto da notação orientada a objetos, este está relacionado a conceitos como especialização e agregação. Neste mesmo trabalho, os autores apresentam outras extensões na UML, como por exemplo, para representar situações em que em uma especialização somente uma classe pode estar presente em uma determinada aplicação. Neste caso as classes são marcadas com o estereótipo <<xorV>>. O estereótipo <<V>> também é utilizado pelos autores para representar variabilidade em operações de uma determinada classe. Um detalhamento maior sobre outras representações de variabilidade pode ser encontrado em [MOR 2000].

Como se pode verificar na Figura 4.2, o <<V>> indica tipos que podem variar conforme as características dos produtos. Por exemplo, o estereótipo <<V>> foi usado para representar variabilidade no tipo Recurso, que pode ser especializado em Assistente e Áudio Visual. O tipo Áudio Visual por sua vez pode ser especializado em Projetor Multimídia, Retroprojetor ou Vídeo. A aplicação prática desta representação pode ser observada no diagrama estático de tipos, na fase de Especificação do Sistema, apresentado na seção 4.3.

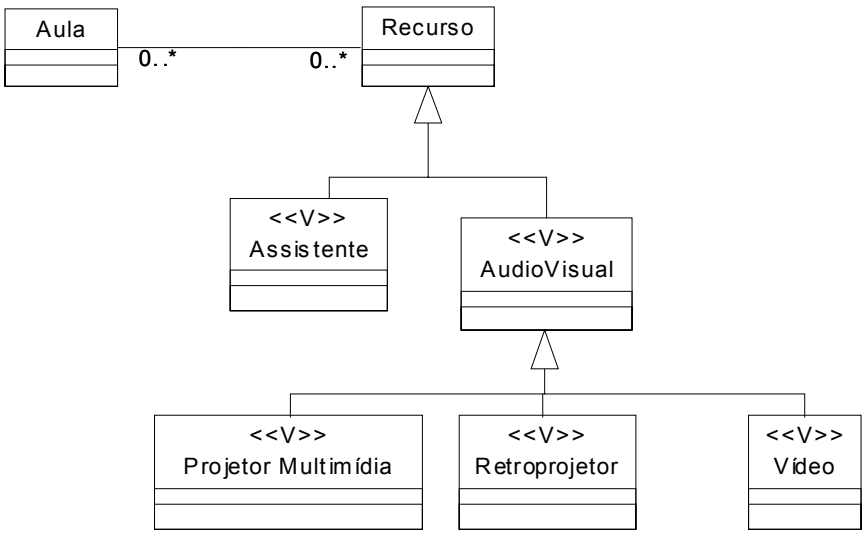

FIGURA 4.2 – Exemplo de representação de variabilidade, estendendo a notação UML

As seções seguintes apresentam as fases seguidas para se projetar a arquitetura de linha de produto para sistemas de gerenciamento de *workflow*, baseada no método Catalysis.

4.2 Análise de Requisitos

Para elaboração de uma arquitetura de linha de produto para WfMS, deve-se primeiramente elaborar um modelo representando os objetos e ações do domínio. Nesta fase é possível identificar aspectos comuns entre os produtos e os pontos de variação.

O modelo de referência e a arquitetura genérica para WfMS da WfMC serviram de base para extrair o conjunto de funcionalidades necessárias para os produtos de *workflow*. O modelo e a arquitetura indicaram quais os potenciais componentes de um WfMS e suas interfaces. Eles foram projetados para que fabricantes de produtos e componentes pudessem desenvolver diversos componentes de WfMS independentemente e integrá-los conforme a demanda das organizações. Além disso, no modelo de referência são definidas características importantes para que produtos específicos possam ser construídos de acordo com as especificidades dos negócios modelados, por exemplo, *workflow* para produção de software ou *workflow* para administração financeira. A partir destes modelos, foram analisadas as possibilidades de extensão da arquitetura e análise de semelhanças e variabilidade. O padrão *Process Manager* [GIM 99a] também foi utilizado no desenvolvimento da arquitetura proposta. Este padrão define um gerenciador de processos para PSEEs (Ambiente de Engenharia de Software Orientado a Processo).

O padrão *Process Manager* propõe um modelo de processos no qual arquiteturas de processos podem ser definidas e reutilizadas, além de permitir a definição de processos de software através da instanciação destas arquiteturas. Este padrão foi definido e documentado utilizando-se UML. As avaliações realizadas por Weiss [WEG 98] com o padrão mostram que este pode servir como base para definição de ambientes de propósito genérico e específicos, facilitando assim a tarefa dos implementadores de ambientes dessa natureza.

A representação do modelo do domínio pode ser feita utilizando-se objetos e ações, em um nível de abstração independente da eventual solução de software que venha a ser encontrada para o problema. Diagramas de casos de uso da UML dão suporte a esta etapa. As Figuras 4.3, 4.4 e 4.5 apresentam os diagramas de caso de uso para um WfMS. Os atores representados nos diagramas são o Gerente de Arquitetura de *Workflow*, o Gerente de *Workflow* e o Usuário do *Workflow*, respectivamente.

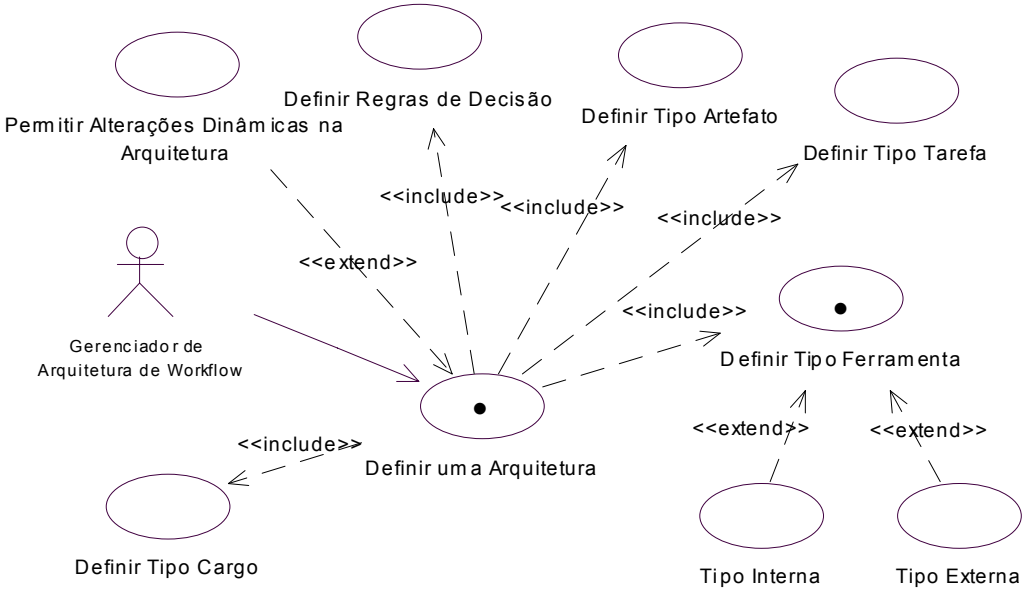

FIGURA 4.3 – Diagrama de Caso de Uso do Gerente de Arquitetura de *Workflow*.

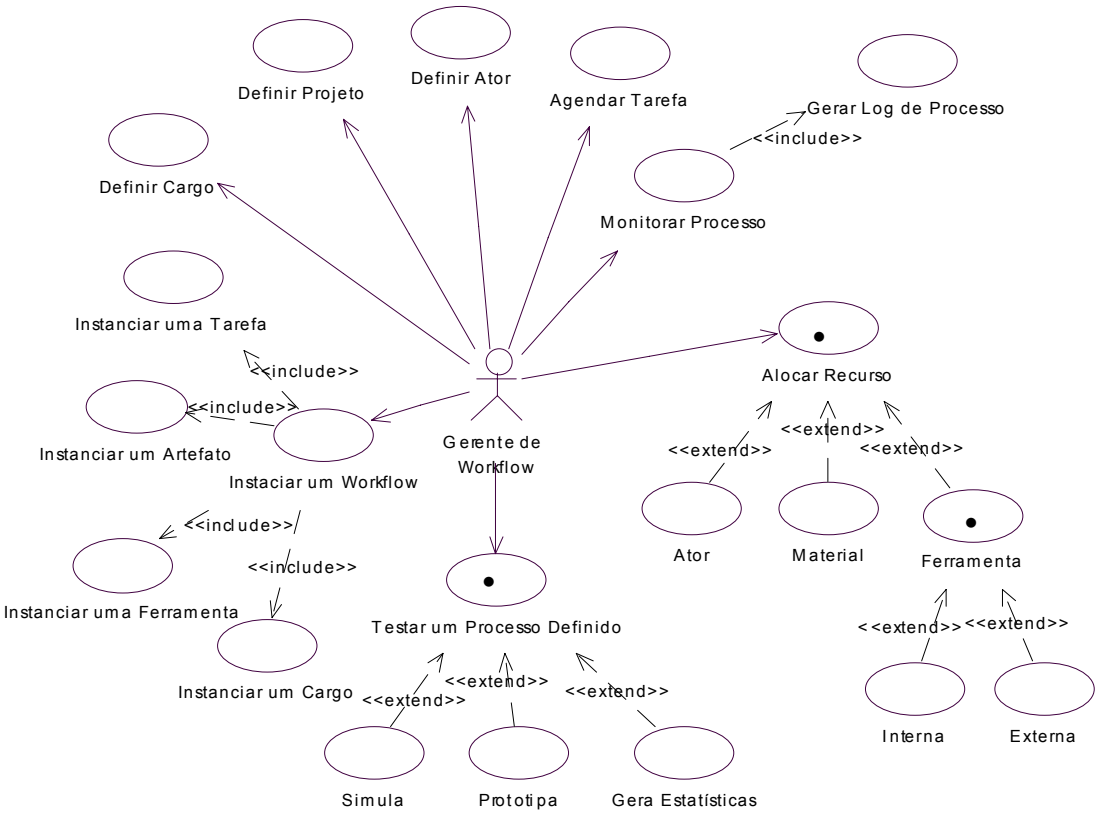

FIGURA 4.4 – Diagrama de Caso de Uso do Gerente de *Workflow*.

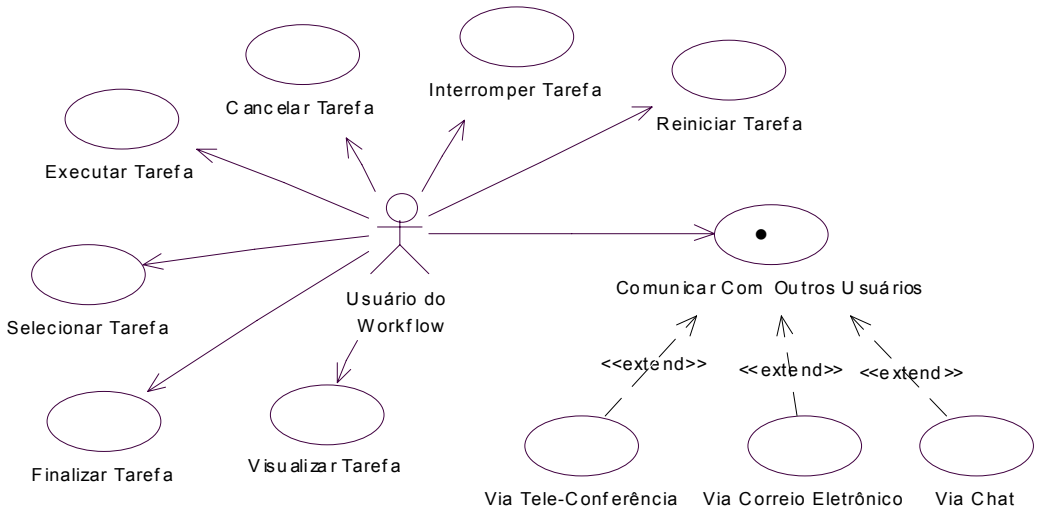

FIGURA 4.5 – Diagrama de Caso de Uso do Usuário do *Workflow*.

Uma descrição mais detalhada de cada caso de uso apresentado acima é apresentada em [UKU 98] e [WEG 98]. Estes trabalhos foram desenvolvidos no contexto do projeto ExPSEE e incluem os diagramas de seqüência e de colaboração correspondentes a cada caso de uso. Estes diagramas são importantes para a identificação dos tipos na fase de especificação do sistema, conforme apresenta a seção seguinte, mas não apresentam representações de variabilidade, que foram incluídas a partir deste trabalho.

4.3 Especificação do Sistema

Nesse estágio do desenvolvimento deve-se tratar da modelagem da solução de software identificada nos modelos de domínio obtidos na fase de especificação de requisitos. A análise das ações do sistema representadas nos diagramas de caso de uso (ver Figuras 4.3, 4.4 e 4.5) possibilitaram a identificação dos tipos, permitindo a associação das referidas ações aos respectivos tipos relacionados. Um tipo em Catalysis não é o mesmo que uma classe. Uma classe descreve a implementação de um objeto; pode-se ter várias implementações da mesma especificação de tipo. Os tipos são especificações de comportamento dos objetos que documentam e mostram somente a visão externa destes. É importante também neste estágio que se consiga enxergar o comportamento externo do sistema, procurando identificar atores, cargos e outras características do modelo de negócio, para que se chegar a uma especificação completa do sistema.

O uso de invariantes, pré-condições e pós-condições são importantes neste estágio. Pode-se utilizar para isso simplesmente uma descrição textual, ou ainda recursos mais poderosos como é o caso da OCL (*Object Constraint Language*) [WAR 99] da UML. Pode-se considerar esta linguagem um formalismo importante, que possibilita a especificação de sentenças bem definidas nos primeiros estágios do processo de desenvolvimento. O uso da linguagem OCL está fora do escopo deste trabalho.

O artefato mais importante extraído neste estágio é o diagrama estático de tipos. A Figura 4.6 apresenta o Diagrama Estático de Tipos para Sistemas de Gerenciamento de *Workflow*.

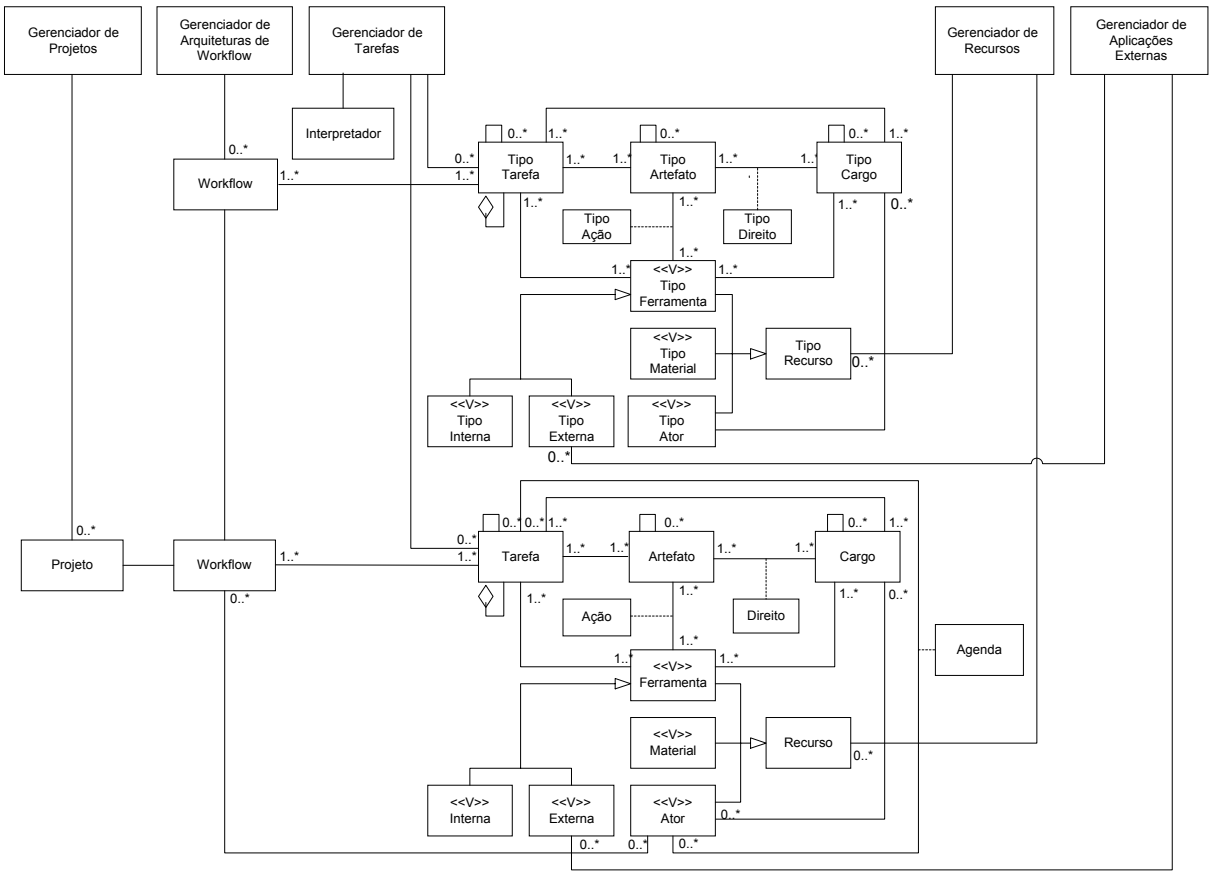

FIGURA 4.6 – Diagrama Estático de Tipos para Sistemas de Gerenciamento de *Workflow*.

Como pode ser observado na Figura 4.6, as partes central e inferior do diagrama estático de tipos foram definidas com base no conceito de reutilização de arquiteturas de workflow. A parte central envolve a definição de arquiteturas de workflow e os tipos de objetos relacionados a esta arquitetura, enquanto a parte inferior envolve a instanciação de arquiteturas de workflow já definidas, bem como os seus tipos de objetos. A parte superior representa os tipos referentes aos módulos do gerenciador de workflow.

De acordo com o diagrama de tipos, uma arquitetura de processo pode possuir um ou mais tipos de tarefas, artefatos, ferramentas ou cargos. A arquitetura tem um relacionamento direto com os tipos de tarefas e indireto, através dos tipos de tarefas, com os demais tipos do diagrama, que são recursos e restrições para a sua realização.

Na parte central do diagrama, existe uma relação entre os tipos envolvidos na arquitetura. Um tipo de tarefa pode ser precedido e/ou composto por zero ou mais tipos de tarefas. Este tipo de tarefa pode utilizar um ou mais tipos de artefatos na produção de um ou mais tipos de artefatos. Esta ação de transformação pode requerer o emprego de um ou mais tipos de cargos. Além disso, esta ação deverá ser realizada por um ou mais tipos de cargos. Um tipo de cargo define os direitos e deveres de um cargo, que será ocupado por um ou mais atores na parte inferior do diagrama. Um tipo de cargo pode

ser liderado por outro tipo de cargo. Como parte de seus direitos, os tipos de cargos podem fazer uso de um ou mais tipos de ferramentas, e/ou manipular um ou mais tipos de artefatos, segundo uma lista de direitos (ex. leitura, escrita e/ou execução). Tipos de ferramentas podem operar sobre tipos de artefatos segundo uma lista de tipos de ações válidas sobre estes tipos de artefatos.

Os objetos presentes na parte inferior do diagrama são instâncias dos tipos definidos na arquitetura de software na parte central do diagrama. Dessa forma, os relacionamentos entre esses objetos comportam-se da mesma maneira descrita para a arquitetura. Assim, nesta etapa tem-se um *workflow* definido, com suas tarefas, artefatos, ferramentas, ações e cargos. Além disso, tem-se também os atores responsáveis pelo desenvolvimento do *workflow*. Cada ator tem associado a si uma agenda onde estão relacionadas as tarefas destinadas a ele e os dados mais importantes relacionados a estas tarefas. Outro ponto diferencial entre esta etapa e a etapa de definição da arquitetura de *workflow* é o fato de que a lista de tipos e ações que um tipo de ferramenta pode executar sobre um tipo deverá ser refinada para uma lista de ações contendo os parâmetros necessários para a realização de ações efetivas sobre os artefatos.

O diagrama estático de tipos apresentado na Figura 4.6 representa a estrutura de um WfMS, que é composta por cinco módulos gerenciadores. Estes módulos estão representados na parte superior do diagrama e são brevemente descritos a seguir:

- Gerenciador de Projetos: responsável pelo controle e gerenciamento da execução dos *workflows*;
- Gerenciador de Arquiteturas de *Workflow*: responsável pelo controle e gerenciamento da construção e manutenção de arquiteturas de *workflow* e sua instanciação através da definição de *workflows*;
- Gerenciador de Tarefas: responsável pelo controle e gerenciamento das tarefas e ações a serem realizadas no *workflow*;
- Gerenciador de Recursos: responsável pelo controle e gerenciamento das ferramentas utilizadas no *workflow*, dos atores e suas agendas, e dos materiais envolvidos no *workflow*;
- Gerenciador de Aplicações Externas: responsável pelo controle e gerenciamento das ferramentas externas invocadas durante o *workflow*;

4.4 Projeto da Arquitetura

Após a modelagem de tipos, uma seqüência de refinamentos desde os níveis mais altos até o nível de componentes deve ser realizada. Catalysis identifica pacotes como a unidade de decomposição de mais alto nível, considerando que eles representam uma parte do sistema que pode ser tratada de forma independente com o explícito estabelecimento de dependências do restante do sistema. Da modelagem de tipos apresentada no Diagrama Estático de Tipos (Figura 4.6) até o refinamento final da arquitetura de linha de produto, são necessários cinco passos, a saber:

1. Identificar os pacotes à partir de uma análise do modelo de negócios e das dependências entre os tipos identificados no Diagrama Estático de Tipos;

2. Estabelecer o relacionamento entre os pacotes de forma a manter uma boa estrutura de importação;
3. A partir do diagrama de pacotes (Diagrama de Camadas Verticais de Alto Nível) e do Diagrama Estático de Tipos, projetar o Diagrama de Camadas Verticais;
4. A partir do Diagrama de Camadas Verticais, projetar a Arquitetura de Componentes;
5. Considerar mecanismos de implementação para o projeto da arquitetura técnica.

O primeiro passo para realizar o particionamento em pacotes consiste em analisar as dependências existentes entre os tipos representados na Figura 4.6, juntamente com uma análise e refinamento do modelo de negócios. Uma visão do modelo de negócios é representada pelo Diagrama de Colaboração de Alto Nível apresentado na Figura 4.7.

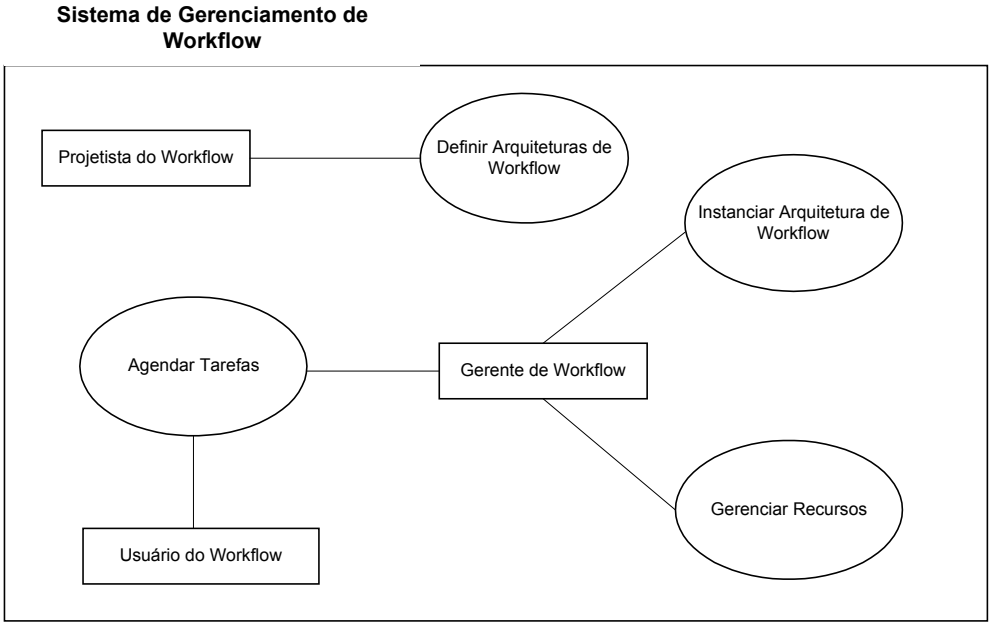

FIGURA 4.7 – Diagrama de Colaboração de Alto Nível para WfMS

O particionamento dos pacotes segue a abordagem de camadas verticais e horizontais proposta por Catalysis. As camadas verticais representam o particionamento em nível de negócio, de acordo com as ações compreendidas pelos principais agentes que interagem com WfMS. Esses agentes são o projetista do *workflow*, que define as arquiteturas reutilizáveis de *workflow*; o gerente de *workflow* (supervisor) que controla a instanciação, alocação de recursos e agendamento das tarefas para o *workflow*; e o usuário do *workflow*, que executa as tarefas do *workflow*. No diagrama, esses agentes são representados através de retângulos e as ações executadas através de elipses, e são representados no Diagrama de Pacotes (Diagrama de Camadas Verticais de Alto Nível) através dos pacotes Gerenciador de Arquitetura de *Workflow*, Gerenciador de *Workflow* e Gerenciador de Execução do *Workflow*.

As camadas horizontais representam o particionamento da arquitetura, separando os pacotes desde os de mais alto nível até os pacotes de infra-estrutura. Este particionamento identifica pacotes de serviço que são compartilhados com os pacotes de

mais alto nível. Esta divisão deve buscar a otimização da importação de pacotes. O pacote Gerenciador de Tarefas está principalmente relacionado com o pacote Gerenciador de *Workflow*, mas também compartilha serviços com os outros dois pacotes de mais alto nível. Os pacotes Interpretador e Gerenciador de Recursos representam serviços independentes oferecidos ao pacote Gerenciador de *Workflow*. Os pacotes Gerenciador de Aplicações Externas e Gerenciador de Objetos do *Workflow* são pacotes de infra-estrutura e estão relacionados com a invocação de ferramentas externas e com o gerenciamento de objetos persistentes, respectivamente.

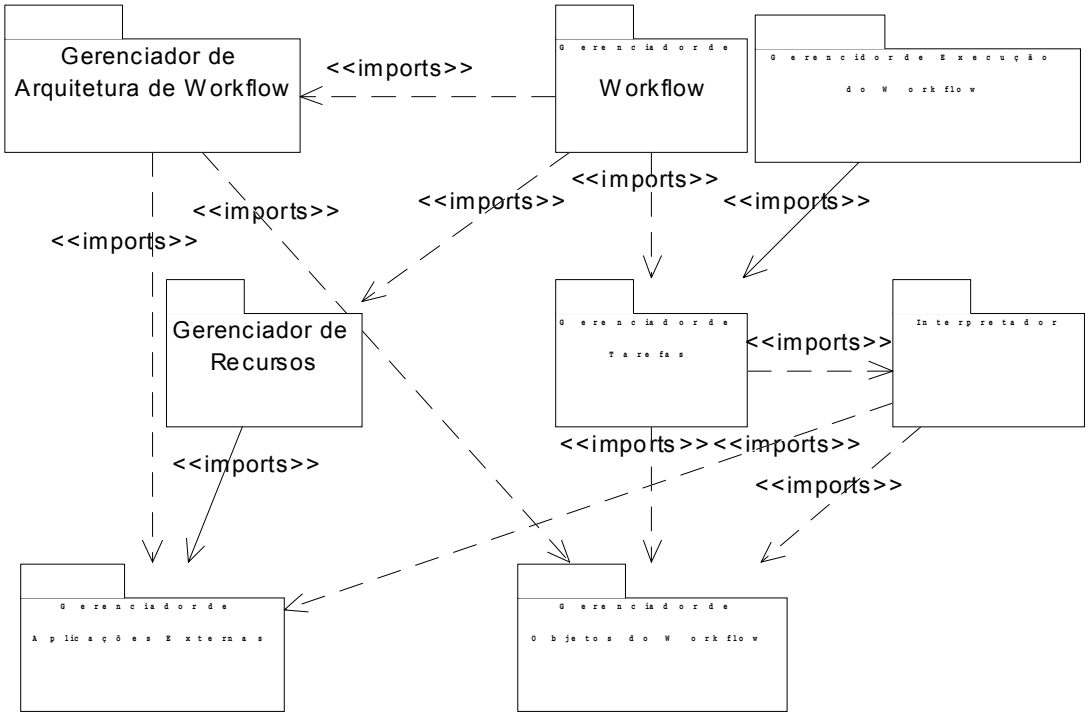

FIGURA 4.8 – Diagrama de Camadas Verticais de Alto Nível.

A correspondência entre os pacotes do diagrama de camadas verticais de alto nível e a arquitetura genérica dos WfMSs (Figura 2.3) são representados como segue: o pacote Gerenciador de Arquitetura de *Workflow*, corresponde à Ferramenta de Definição; os pacotes Gerenciador de *Workflow* e Gerenciador de Execução do *Workflow* correspondem a uma ou mais máquinas de *workflow*; o pacote Gerenciador de Recursos e Interpretador também são parte da(s) máquina(s) de *workflow*, utilizados pelo pacote Gerenciador de *Workflow*; o pacote Gerenciador de Tarefas corresponde ao Gerenciador das Listas de Trabalho, mas também está relacionado com a(s) máquina(s) de *workflow*. Os três principais pacotes Gerenciador de Arquitetura de *Workflow*, Gerenciador de *Workflow* e Gerenciador de Execução do *Workflow* podem utilizar o pacote Gerenciador de Aplicações Externas. Isto corresponde às três referências para aplicações externas da arquitetura genérica: Ferramenta de Definição, Máquina(s) de *Workflow* e Gerenciador da Lista de Trabalho. Os objetos (dados de controle ou relevantes) do WfMS são representados no diagrama pelo pacote Gerenciador de Objetos do *Workflow*. Este pacote controla todos os dados e objetos manipulados no WfMS [GIM 01].

A partir do diagrama de camadas verticais de alto nível, foi possível projetar o Diagrama de Camadas Verticais. A construção deste modelo tomou como base os

pacotes definidos anteriormente pelo Diagrama de Camadas Verticais de Alto Nível juntamente com a especificação dos tipos de cada pacote e o relacionamento entre eles. A Figura 4.9 apresenta o Diagrama de Camadas Verticais, juntamente com a especificação de tipos de cada pacote.

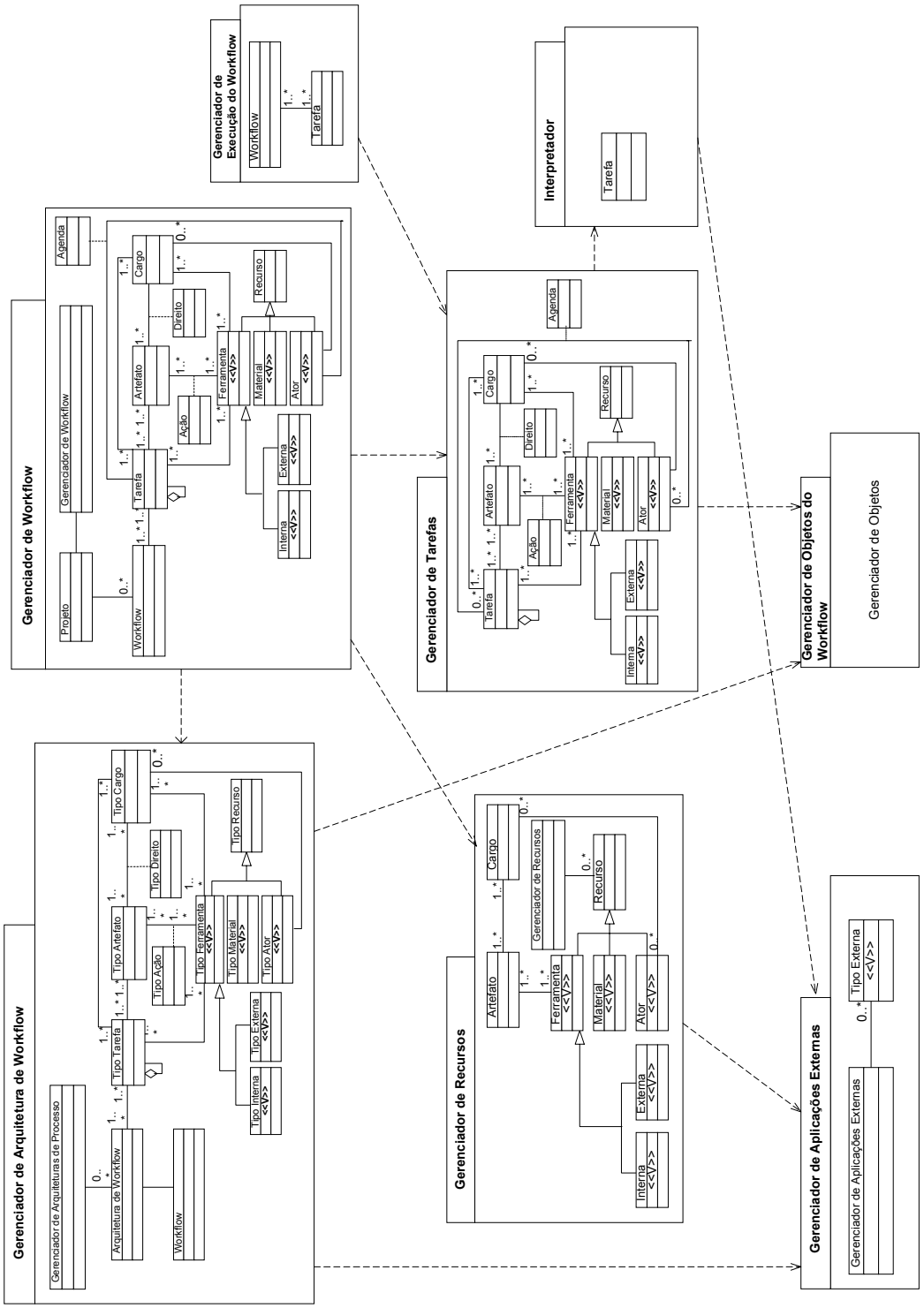

FIGURA 4.9 – Diagrama de Camadas Verticais

4.4.1 Arquitetura de Componentes

A construção do Diagrama de Camadas Verticais apresentado na Figura 4.9 finaliza o processo de particionamento em componentes, apresentando os pacotes juntamente com a especificação dos tipos de cada pacote e o relacionamento entre eles. Os componentes podem então ser vistos como pacotes genéricos, constituídos por tipos e relacionamentos. A partir deste diagrama foi possível projetar a arquitetura de componentes para WfMS conforme apresenta a Figura 4.10.

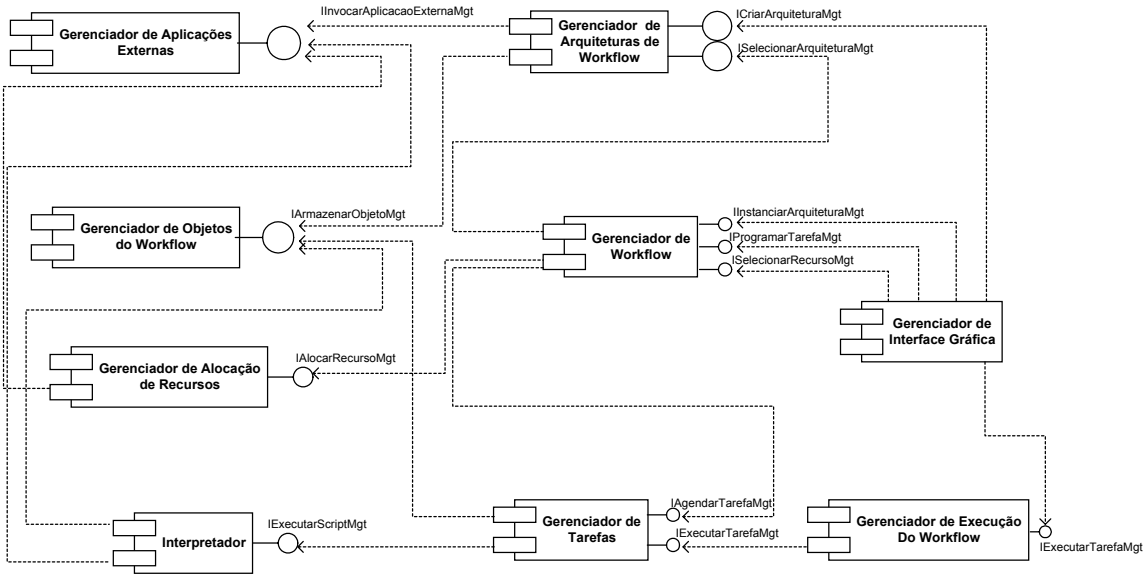

FIGURA 4.10 – Arquitetura de Componentes para WfMS.

Observando a arquitetura de componentes pode-se notar que o componente Gerenciamento de Interface Gráfica foi incluído neste artefato. Este componente tem a funcionalidade de gerenciar a interação com os usuários do WfMS.

A partir da arquitetura de componentes, foi possível elaborar diagramas de seqüência que representam a comunicação entre os componentes do sistema de gerenciamento de *workflow* para cada caso de uso principal, conforme apresentado nas Figura 4.11, Figura 4.12 e Figura 4.13. Um diagrama de seqüência mostra a colaboração dinâmica entre os vários objetos de um sistema. O mais importante aspecto deste diagrama é que a partir dele pode-se perceber a seqüência de mensagens enviadas entre os objetos. Neste caso, a mesma idéia é aplicada aos componentes.

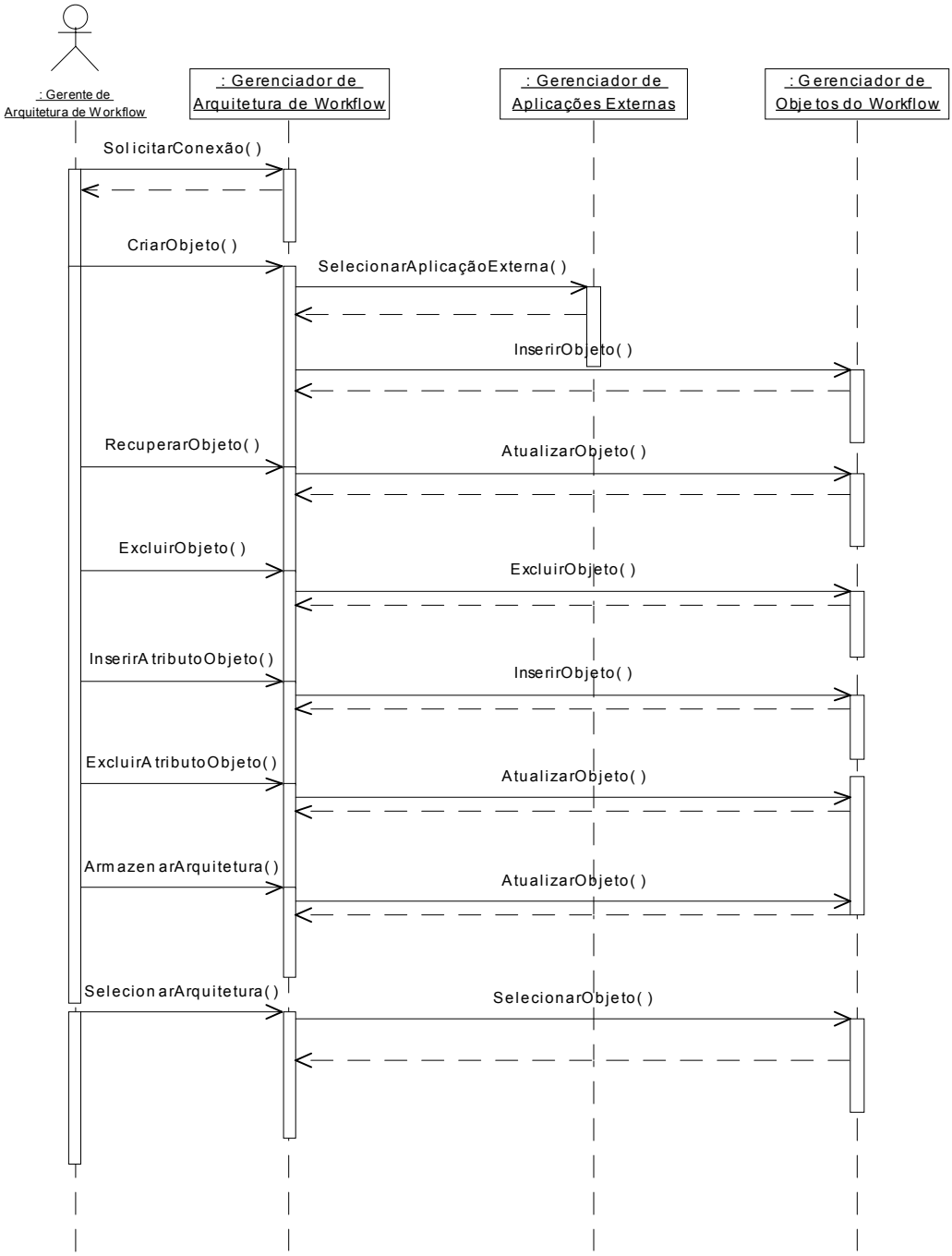

FIGURA 4.11 – Diagrama de Seqüência/Gerente de Arquitetura de *Workflow*.

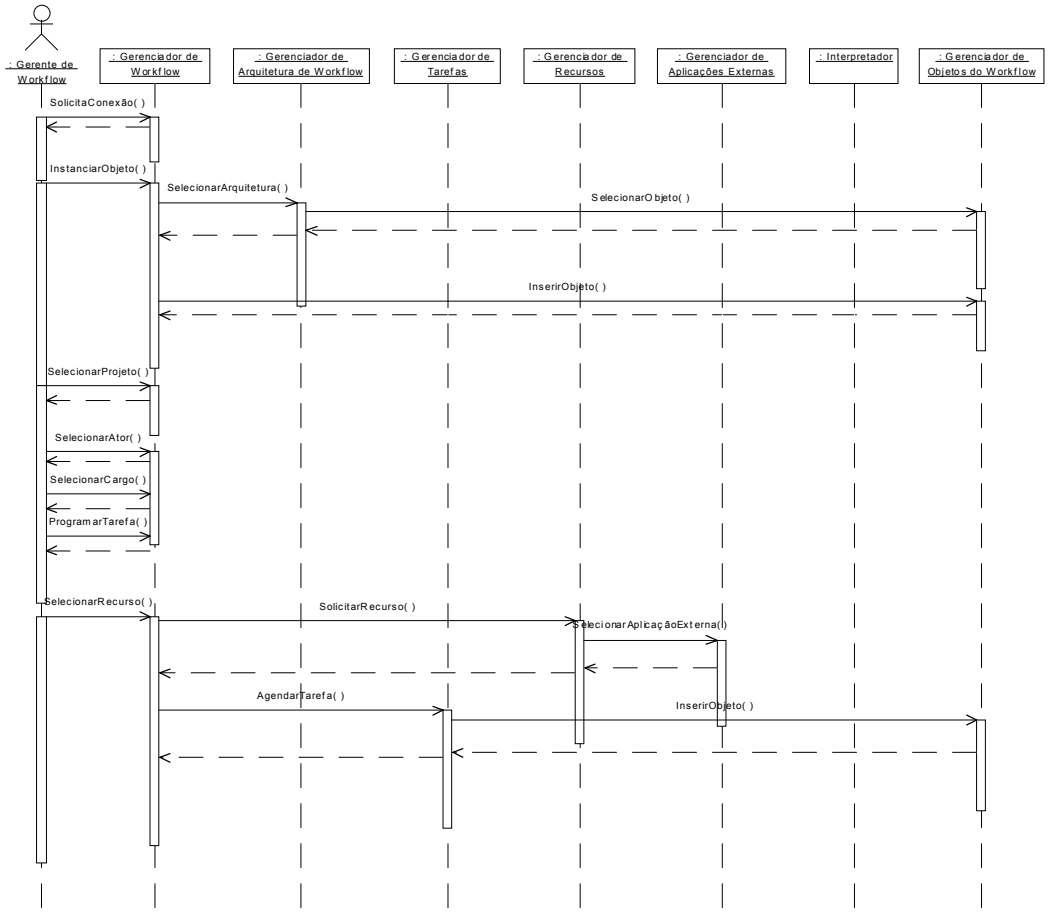

FIGURA 4.12 – Diagrama de Seqüência/Gerente de *Workflow*.

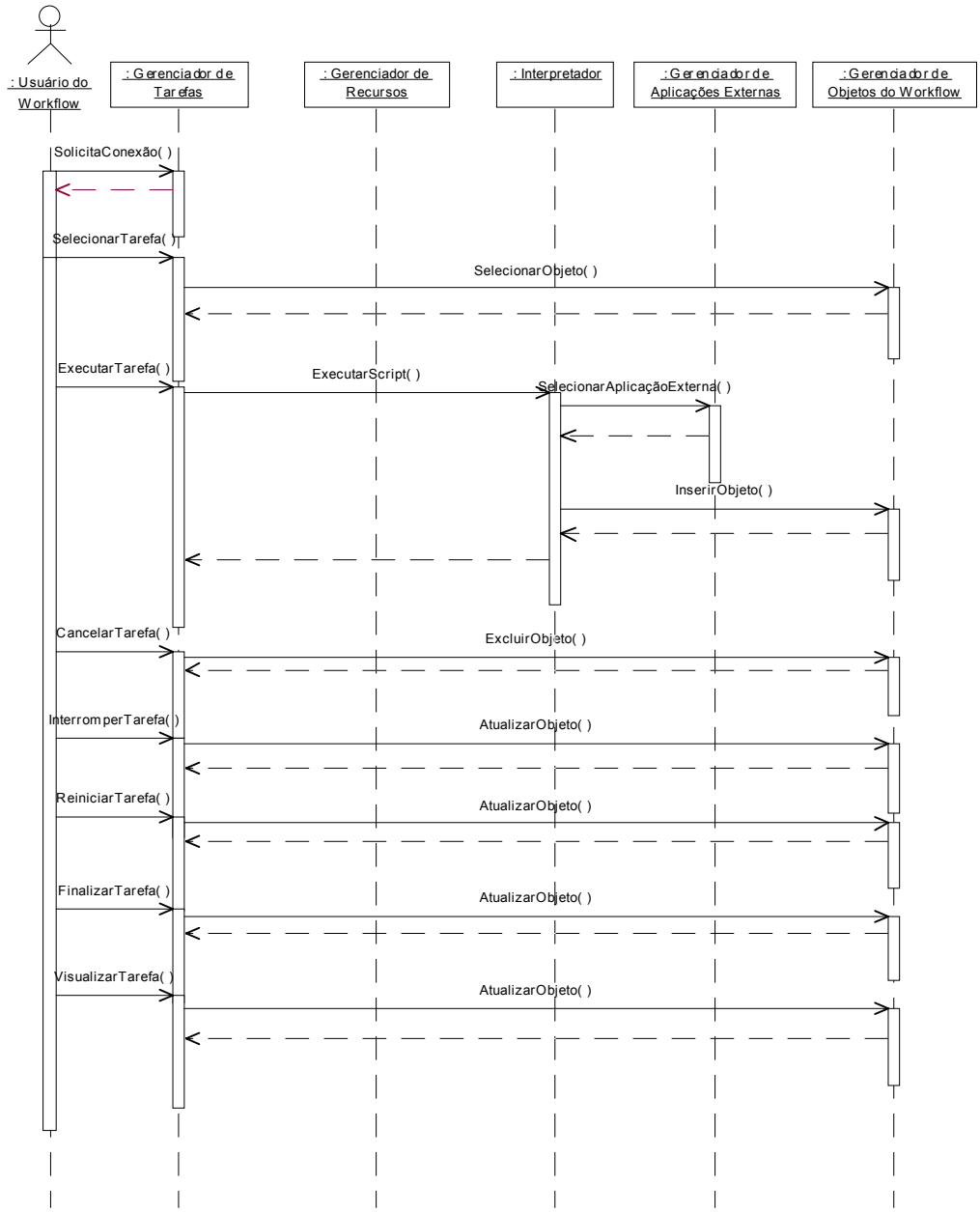

FIGURA 4.13 – Diagrama de Seqüência/Usuário do *Workflow*

Os itens seguintes trazem uma descrição de cada um dos componentes da arquitetura proposta com suas respectivas interfaces e métodos, além de descrever a funcionalidade e aspectos de variabilidade associados a estes componentes.

Gerenciador de Interface Gráfica: este componente é responsável pelo gerenciamento da interface com o usuário do sistema. O ponto de variação deste componente está no fato de que a interface com o usuário pode ocorrer via interface gráfica convencional ou via *browser*.

Gerenciador de Arquitetura de Workflow: este componente é responsável pelo controle e gerenciamento da construção e manutenção de arquiteturas de *workflow*. Este componente suporta as funções relacionadas à definição de arquiteturas de *workflow* e

dos tipos de objetos relacionados a esta. A utilização deste componente torna a definição de *workflows* mais flexível, visto que desta forma, os tipos de objetos não são pré-fixados. Os pontos de variação associados a este componente indicam o tipo de recurso que pode ser utilizado. Este tipo recurso pode ser especializado em ator, ferramenta e material e o tipo de ferramenta pode ser especializado em interna e externa. Diferentes linguagens de programação de processos podem ser utilizadas para a definição das tarefas, que posteriormente serão executadas pelo interpretador.

ICriarArquiteturaMgt	
SolicitarConexão()	Permite a conexão de um usuário
CriarObjeto()	Cria um novo objeto de definição de *workflow* (ex. um ator, cargo, uma ferramenta)
ExcluirObjeto()	Exclui um objeto de definição de *workflow*
RecuperarObjeto()	Recupera um objeto de definição de *workflow*
InserirAtributoObjeto()	Insere um atributo específico em um objeto de definição
DeletarAtributoObjeto()	Exclui um atributo específico em um objeto de definição
ArmazenarArquitetura()	Salva uma arquitetura de referência

IselecionarArquiteturaMgt	
SelecionarArquitetura()	Seleciona uma arquitetura de *workflow*

Gerenciador de Workflow: este componente é 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 executadas e gerenciadas, fazendo-se a alocação de recursos e demais tomadas de decisão. Não foram definidos pontos de variação associados a este componente.

IInstanciarArquiteturaMgt	
SolicitarConexão()	Permite a conexão de um usuário
SelecionarArquitetura()	Seleciona uma arquitetura de *workflow* para instanciação
InstanciarObjeto()	Instancia objetos definidos na arquitetura de *workflow* selecionada

IProgramarTarefaMgt	
SelecionarProjeto()	Seleciona projeto a ser executado
SelecionarTarefa()	Seleciona tarefa a ser codificada
SelecionarCargo()	Seleciona cargo ao qual a tarefa a ser codificada está associada
SelecionarAtor()	Seleciona ator para a execução da tarefa
ProgramarTarefa()	Programa uma determinada tarefa

ISelecionarRecursoMgt	
SelecionarRecurso()	Seleciona recursos que serão usados na execução de uma determinada tarefa

Gerenciador de Execução do Workflow: este componente é responsável pelo controle e gerenciamento das tarefas a serem realizadas no *workflow*. É através dele que os usuários identificam as suas tarefas no *workflow*. Não foram definidos pontos de variação associados a este componente.

IExecutarTarefaMgt	
SolicitarConexão()	Permite a conexão de um usuário
SelecionarTarefa()	Seleciona uma determinada tarefa
ExecutarTarefa()	Executa uma determinada tarefa
VisualizarTarefa()	Mostra as tarefas pertencentes a um determinado usuário de acordo com o seu cargo

Gerenciador de Tarefas: este componente é responsável pelo controle e gerenciamento das tarefas e ações a serem realizadas. O Gerenciador de Tarefas permite que os usuários identifiquem as tarefas geradas pelo Gerenciador de *Workflow*. Os pontos de variação deste componente estão associados aos tipos de recursos que podem ser utilizados (material, ferramenta e ator); a ferramenta utilizada pode ser do tipo interna ou externa. As tarefas podem ter prioridades de execução diferentes e, os algoritmos utilizados para escalonamento das tarefas também podem variar.

IAgendarTarefaMgt	
SolicitarConexão()	Permite a conexão de um usuário
AgendarTarefa()	Agenda uma determinada tarefa

IExecutarTarefaMgt	
SelecionarTarefa()	Seleciona uma determinada tarefa
ExecutarTarefa()	Executa uma determinada tarefa
CancelarTarefa()	Cancela uma determinada tarefa a ser executada ou que já está em execução por um determinado usuário
InterromperTarefa()	Interrompe uma determinada tarefa
ReiniciarTarefa()	Reinicia uma determinada tarefa
FinalizarTarefa()	Finaliza uma determinada tarefa
VisualizarTarefa()	Mostra as tarefas pertencentes a um determinado usuário de acordo com o seu cargo

Gerenciador de Alocação de Recursos: este componente é responsável pela alocação de recursos (atores, ferramentas ou material). Além da variabilidade associada ao tipo de recurso e ao tipo de ferramenta, as políticas de alocação de recurso podem variar.

IAlocarRecursoMgt	
SolicitarRecurso()	Solicita um determinado recurso (ex. ferramenta, ator)
LiberarRecurso()	Libera um determinado recurso para efetiva utilização
ExcluirRecurso()	Exclui um determinado recurso
RecuperarEstadoRecurso()	Recupera o estado de um determinado recurso (ex. reservado)

Gerenciador de Aplicações Externas: este componente é responsável pelo gerenciamento das aplicações externas invocadas durante a definição do processo e execução das tarefas. Pontos de variabilidade incluem as diferentes formas de adaptação da aplicação externa ao sistema gerenciador de *workflows*.

IinvocarAplicaçõesExternasMgt	
SelecionarAplicaçãoExterna()	Seleciona uma aplicação externa
InvocarAplicaçãoExterna()	Invoca uma aplicação externa

Gerenciador de Objetos do Workflow: este componente é responsável pelo relacionamento com os mecanismos de armazenamento de objetos manipulados pelo sistema. Suas funções trabalham com objetos, que podem ser considerados desde dados de controle do processo, dados relevantes do processo ou até mesmo instâncias de *workflow* e tarefas. Todos os componentes pertencentes à arquitetura de componentes proposta utilizam seus serviços. 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. Pontos de variação deste componente incluem os adaptadores para os gerenciadores de bancos de dados.

IArmazenarObjetoMgt	
InserirObjeto()	Insere um novo objeto de *workflow*
AtualizarObjeto()	Atualiza um objeto de *workflow*
ExcluirObjeto()	Exclui um objeto de *workflow*
SelecionarObjeto()	Seleciona um objeto de *workflow*

Interpretador: uma linguagem para programação de processos é necessária para que o processo possa ser programado e executado pelo gerenciador de *workflow*. As tarefas que compõem o *workflow* são então programadas utilizando uma linguagem de programação de processos e recursos que permitem a execução cooperativa dessas tarefas. Para a execução dessas tarefas, o gerenciador de tarefas realiza chamadas ao interpretador da linguagem para que este possa executá-las.

IExecutarScriptMgt	
ExecutarScript()	Executa *Script*
InvocarAplicaçãoExterna()	Invoca uma aplicação externa

4.4.2 Arquitetura de Componentes com CORBA

Segundo o processo seguido pelo Catalysis, após a modelagem da chamada arquitetura lógica (Figura 4.10), deve-se tomar decisões sobre os mecanismos de implementação a serem utilizados, como por exemplo, o *middleware*. Estas decisões levam ao projeto da arquitetura técnica, conforme apresentada na Figura 4.14. Para a arquitetura de linha de produto proposta foi considerado o ORB (*Object Request Broker*) do CORBA [OMG 2000] como *middleware*, que ocupa o centro da figura. Os componentes são estruturados em camadas de modo a minimizar as importações.

Os componentes da arquitetura possuem interfaces padronizadas e bem definidas através da linguagem IDL (*Interface Definition Language*) do CORBA. Isso faz com

que os componentes possam ser acessados dentro da arquitetura independentemente da linguagem de programação, sistema operacional e rede utilizados.

As comunicações entre os componentes são feitas através do ORB e o componente requisitado atende ou não à requisição, baseando-se em restrições de comunicação que são definidas para cada componente da arquitetura. Por exemplo, o componente *Gerenciador de Interface Gráfica* comunica-se apenas com os componentes *Gerenciador de Arquitetura de Workflow*, *Gerenciador de Workflow* e *Gerenciador de Execução do Workflow*. Qualquer solicitação de comunicação de outro componente da arquitetura com o componente *Gerenciador de Interface Gráfica* será recusada porque existem restrições de comunicação definidas no componente requisitado.

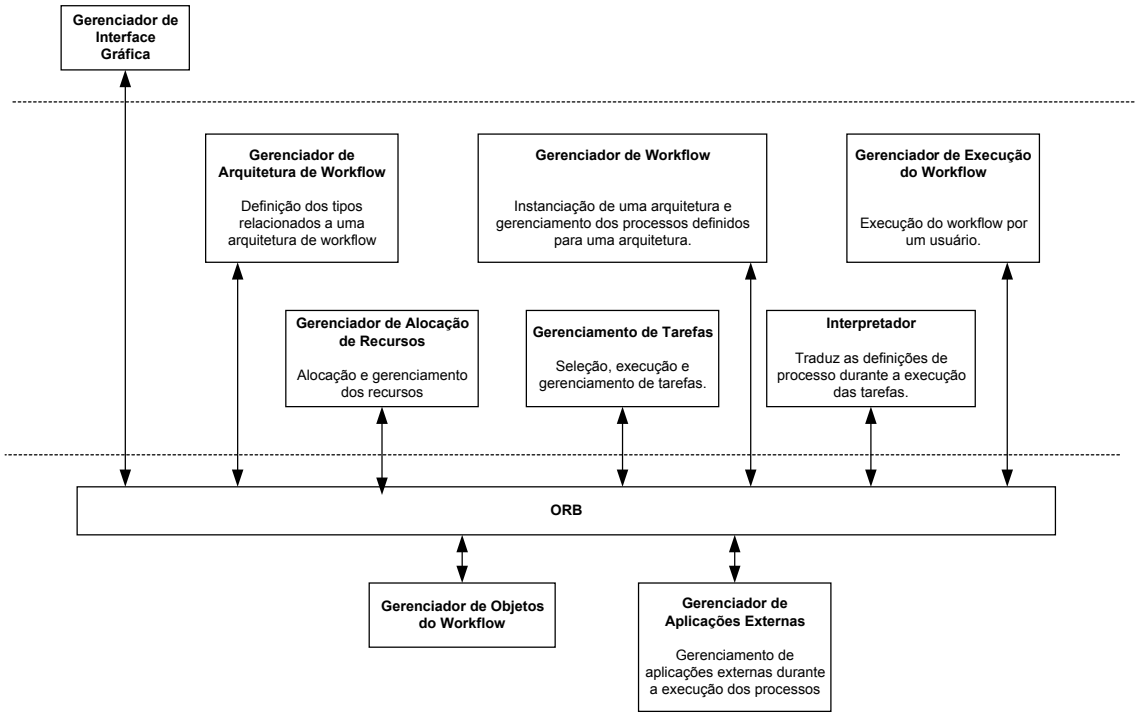

FIGURA 4.14 – Arquitetura de Linha de produto para WfMS proposta.

Nesta fase do processo, os componentes da arquitetura estão especificados com suas interfaces definidas. Segundo o processo seguido para este trabalho, a etapa seguinte consiste no projeto interno dos componentes da arquitetura. Porém, a implementação desses componentes está fora do contexto deste trabalho.

4.5 Validação da Arquitetura Proposta

Bosch [BOS 2000] identificou quatro formas de validação de uma arquitetura de linha de produto: uso de cenários, simulação, modelos matemáticos e avaliação baseada em experiências passadas. As seções anteriores descreveram o processo de obtenção da arquitetura de linha de produto para sistemas de gerenciamento de *workflow*. Este processo é baseado no método Catalysis [D'SO 99], que utiliza a UML [BOO 99] [FOW 97] [RUM 99] como notação que não possui recursos para simulação da arquitetura. Assim, para que pudéssemos avaliar a arquitetura proposta sem nos

aprofundar em detalhes de implementação, optamos por utilizar uma ADL (*Architecture Description Language*) para especificar e simular a arquitetura. A ADL Rapide [CSL 97] foi identificada como adequada para especificar a arquitetura proposta neste trabalho. Esta linguagem foi escolhida por ser caracterizada como uma linguagem de descrição arquitetural com finalidade geral, que permite modelar interfaces de componentes e seus comportamentos externamente visíveis, e ainda por ser uma linguagem que permite simulação e modelagem do comportamento dinâmico descrito por uma arquitetura.

A arquitetura de linha de produto proposta para Sistemas de Gerenciamento de *Workflow* foi simulada utilizando a linguagem Rapide e suas ferramentas. A Figura 4.15 apresenta a arquitetura de componentes proposta (ver Figura 4.10) modelada na ferramenta *Raparch*.

A simulação foi baseada na seleção de cenários relevantes para os sistemas de gerenciamento de *workflow*. Os Diagramas de Seqüência (ver Figuras 4.11, 4.12, 4.13) mostram as interações para cenários específicos de uma certa atividade do sistema, especificando a comunicação entre os componentes. Esses diagramas serviram de base para realizar a simulação da arquitetura de linha de produto proposta. Dessa maneira, a arquitetura é executada conforme diferentes cenários para simular o comportamento do sistema. Cada cenário representou uma das visões dos diferentes usuários do sistema. Esses usuários são: o gerente de arquitetura de *workflow*, que define a arquitetura de *workflow*; o gerente de *workflow*, responsável pelas atividades de supervisão e administração e, por fim, o usuário que executa as tarefas do *workflow*. Além disso, a modelagem do comportamento dinâmico descrito pela arquitetura de linha de produto proposta, possibilita avaliar a comunicação entre os componentes e o efeito geral das funcionalidades do sistema.

O cenário descrito na simulação da arquitetura de linha de produto realizada neste trabalho descreve a execução contínua das atividades do gerenciador de arquitetura de *workflow*, gerenciador de *workflow* e usuário do *workflow*. O gerenciador de arquitetura de *workflow* cria a arquitetura de *workflow* e a armazena. Logo o gerenciador de *workflow* pode instanciar esta arquitetura e, em seguida, codificar e alocar recursos às tarefas pertencentes a ela. Tão logo, essas tarefas são disponibilizadas na agenda do usuário do *workflow* para que sejam executadas.

Outro cenário possível descreve a execução parcial das atividades realizadas pelo gerenciador de *workflow*. O gerenciador de arquiteturas de *workflow* cria a arquitetura de *workflow* e a armazena. Quando necessário, o gerenciador de *workflow* pode selecionar uma arquitetura criada no banco de dados do sistema – Gerenciador de Objetos do *Workflow* - e instanciá-la. Da mesma forma, quando achar conveniente, o gerenciador de *workflow* pode codificar as tarefas pertencentes à arquitetura de *workflow* instanciada. Em seguida, essas tarefas são colocadas na agenda do usuário do *workflow* para que sejam executadas.

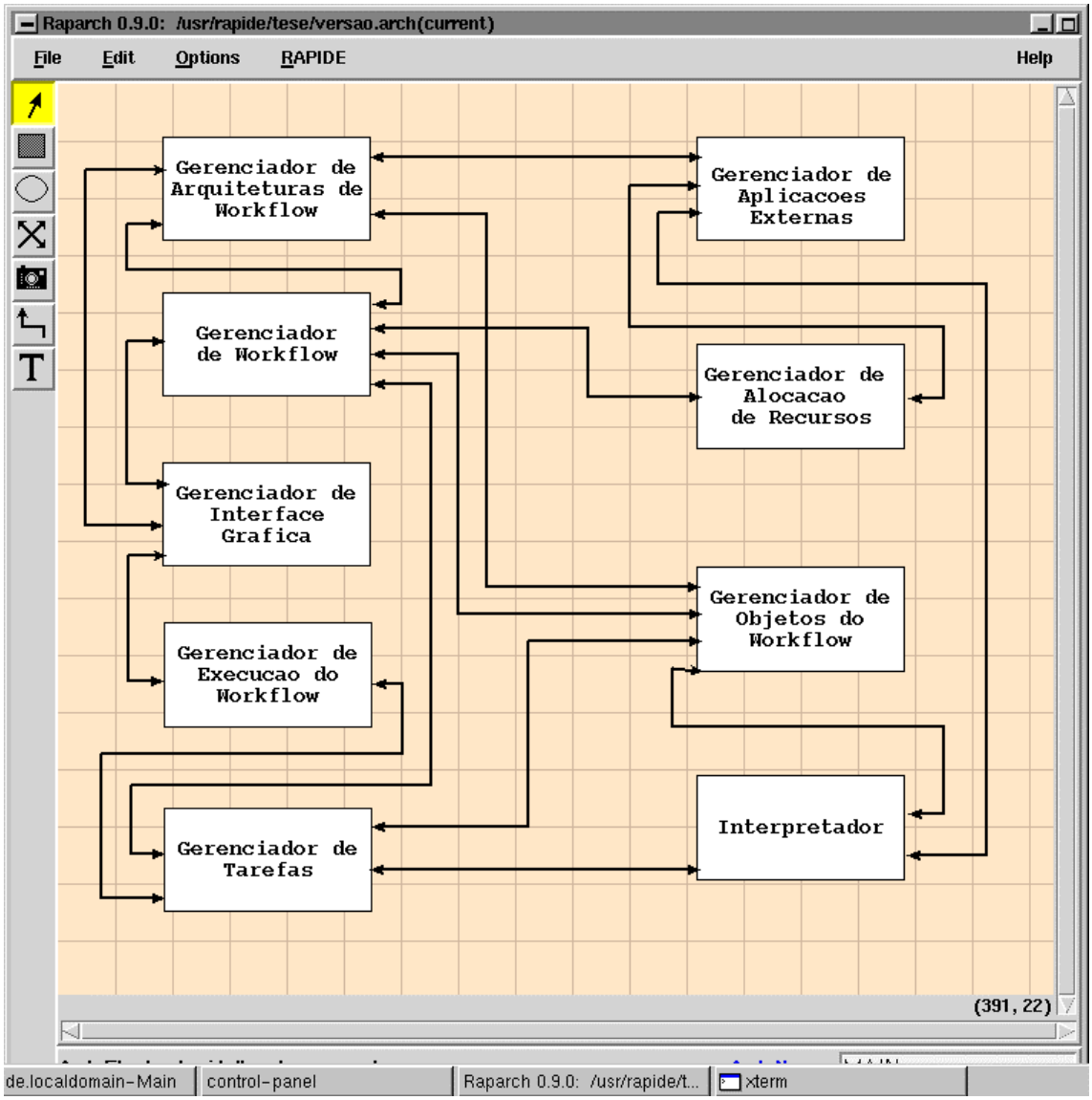

FIGURA 4.15 – Arquitetura de linha de produto proposta modelada na ferramenta *Raparch*.

Pode-se pensar também em um cenário no qual ocorrem erros. Por exemplo, para que uma arquitetura de *workflow* seja instanciada em um *workflow*, esta arquitetura deve estar completa. Sendo assim, deveria existir um módulo no gerenciador de *workflow* responsável por verificar quais os tipos de objetos de uma arquitetura de *workflow* que estão completos e quais não estão, informando quais são os relacionamentos que estão faltando para que estes se tornem completos. Dessa maneira, seria possível simular um cenário no qual a falta de um relacionamento seja detectada pelo Inspetor. Outro exemplo seria quando os recursos (humano, material, ator) não estão disponíveis a uma determinada tarefa, ou ainda quando o usuário do *workflow* não pode executar sua tarefa dentro do prazo previsto.

Devido ao fato de um sistema de gerenciamento de *workflow* controlar o acesso a suas funcionalidades através do tipo de cargo que um ator assume, a simulação da arquitetura de linha de produto foi desenvolvida a partir da visão dos cargos existentes no ambiente (Gerente de Arquitetura de *Workflow*, Gerente de *Workflow* e Usuário do

Workflow). A Figura 4.16 apresenta um instante da animação da arquitetura de linha de produto proposta.

De acordo com a animação, as mensagens de confirmação (OK) foram criadas para auxiliar na visualização. Além disso, a linguagem Rapide não suporta dois eventos com o mesmo nome, dessa forma, para eventos semelhantes precisamos nomeá-los de modos distintos, como por exemplo: *Selecionar_Arquitetura1()* e *Selecionar_Arquitetura2()*.

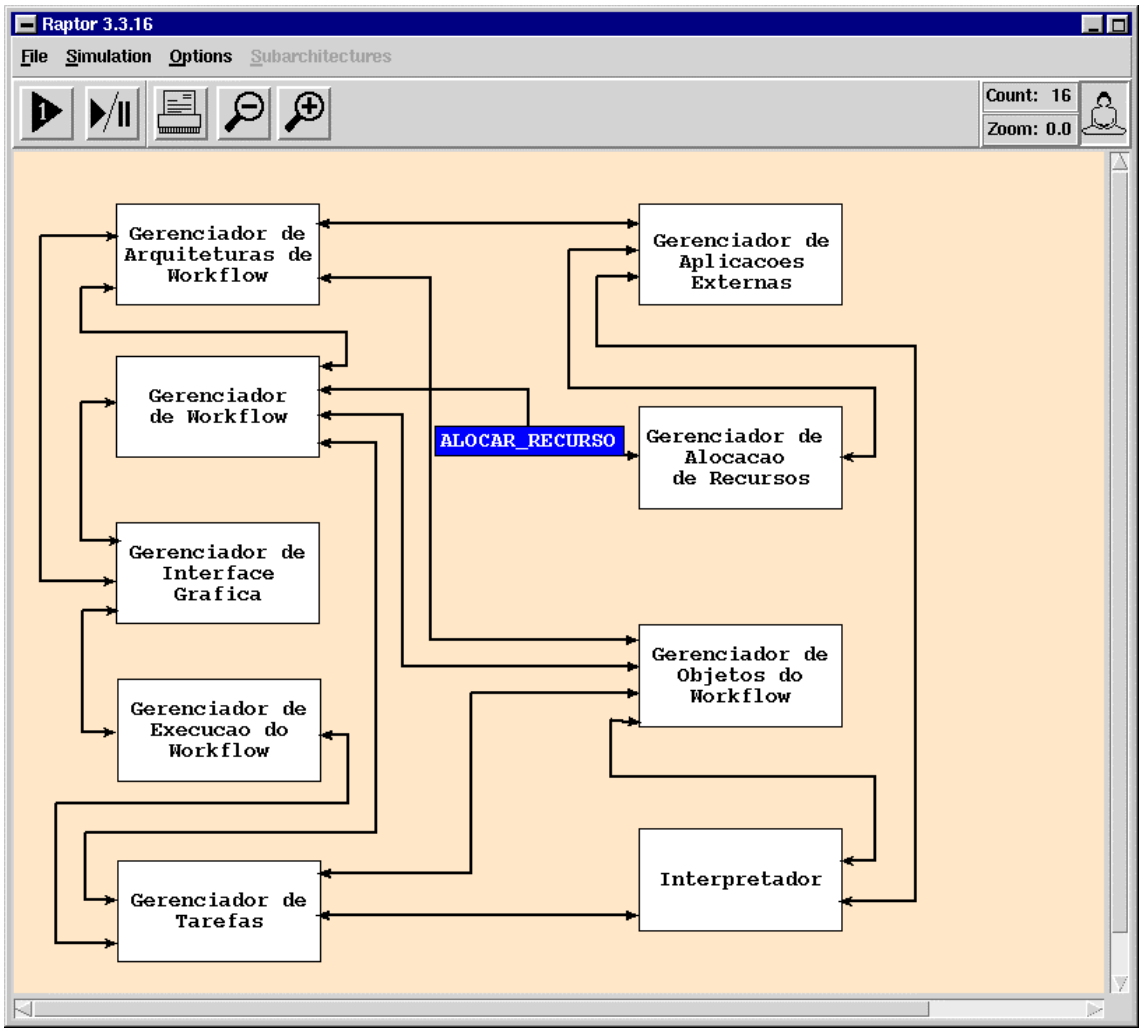

FIGURA 4.16 – Animação da arquitetura de linha de produto proposta.

Conforme a Figura 4.16, a simulação da arquitetura apresenta um cenário contínuo com a seguinte descrição: o gerente de arquitetura de *workflow* através do componente Gerenciador de Interface Gráfica, envia uma mensagem – *Criar_Arquitetura()* – ao componente Gerenciador de Arquitetura de *Workflow* solicitando uma conexão para que uma arquitetura de *workflow* seja criada. Quando a conexão é estabelecida, o gerente de arquitetura de *workflow* inicia a criação da arquitetura. Durante a criação da arquitetura, o gerente de arquitetura de *workflow* pode necessitar de alguma ferramenta externa ao ambiente. Assim o componente *Gerenciador de Arquitetura de Workflow* estabelece uma comunicação – *Selecionar_Aplicação_Externa()* – com o componente Gerenciador de Aplicações Externas, solicitando informações sobre a aplicação externa requisitada. Quando a

aplicação externa é selecionada, uma mensagem de confirmação – *Selecionar_Aplicação_Exeterna_OK()* – é retornada para o componente *Gerenciador de Arquitetura de Workflow* para que o gerente de arquitetura de *workflow* possa utilizar a nova ferramenta e continuar com a sua tarefa. Logo que a arquitetura de *workflow* tiver sido criada, ela é armazenada na base de dados do sistema de gerenciamento de *workflow*. Existe então, uma comunicação – *Inserir_Arquitetura()* – com o componente *Gerenciador de Objetos do Workflow* para que a arquitetura seja armazenada. Quando a arquitetura é armazenada, o componente *Gerenciador de Objetos do Workflow* retorna uma mensagem de confirmação – *Inserir_Arquitetura_OK()* – para o componente Gerenciador de Arquitetura de *Workflow*. Após o processo de criação da arquitetura ter sido realizado com sucesso, uma mensagem – *Criar_Arquitetura_OK()*– é enviada ao componente Gerenciador de Interface Gráfica, finalizando o processo. Neste momento, tem-se uma arquitetura de *workflow* criada pelo gerente de arquitetura de *workflow* com sucesso.

Após criar a arquitetura é preciso instanciá-la. Assim, o gerente de *workflow*, através do componente Gerenciador de Interface Gráfica, envia uma mensagem – *Instanciar_Arquitetura()* – ao componente Gerenciador de *Workflow* solicitando uma conexão para que uma arquitetura de *workflow* existente seja instanciada. Quando a conexão é estabelecida, o gerente de *workflow* inicia a instanciação da arquitetura. Inicialmente, o componente Gerenciador de *Workflow* faz uma comunicação – *Selecionar_Arquitetura1()* – com o componente Gerenciador de Arquitetura de *Workflow* para que a arquitetura de *workflow* a ser instanciada seja selecionada. O componente Gerenciador de Arquitetura de *Workflow* por sua vez envia uma mensagem – *Selecionar_Arquitetura2()* – ao componente Gerenciador de Objetos do *Workflow* solicitando a seleção da arquitetura a ser instanciada, que retorna uma mensagem de confirmação – *Selecionar_Arquitetura2_OK()*. Quando a arquitetura é selecionada, o componente Gerenciador de Arquitetura de *Workflow* também retorna uma mensagem – *Selecionar_Arquitetura1_OK()* – ao componente Gerenciador de *Workflow*. Logo que a arquitetura de *workflow* tiver sido instanciada, o componente *Gerenciador de Workflow* envia uma mensagem de sucesso – *Instanciar_Arquitetura_OK()*– ao componente *Gerenciador de Interface Gráfica*, finalizando o processo de instanciação. Neste momento, tem-se uma arquitetura de *workflow* instanciada pelo gerente de *workflow* com sucesso.

Em seguida, o gerente de *workflow* precisa programar e alocar recursos às tarefas pertencentes à arquitetura de *workflow* instanciada. Dessa maneira, o gerente de *workflow*, através do componente Gerenciador de Interface Gráfica, envia uma mensagem – *Programar_Tarefa()* – ao componente Gerenciador de *Workflow* solicitando uma conexão para programar a tarefa. Quando a conexão é estabelecida, o gerente de *workflow* inicia a programação da tarefa. Após a tarefa ter sido programada, o componente Gerenciador de *Workflow* envia uma mensagem de sucesso – *Programar_Tarefa_OK()* – ao componente Gerenciador de Interface Gráfica, finalizando o processo de codificação. Logo em seguida, o gerente de *workflow* novamente através do componente Gerenciador de Interface Gráfica, envia uma mensagem – *Selecionar_Recurso()* – ao componente Gerenciador de *Workflow* solicitando uma conexão para alocar recursos à tarefa que acabou de ser programada. Quando a conexão é estabelecida, o componente Gerenciador de *Workflow*, envia uma mensagem – *Alocar_Recurso()* – ao componente Gerenciador de Alocação de Recurso solicitando a alocação de um recurso à tarefa. O componente Gerenciador de Alocação de Recurso envia uma mensagem – *Selecionar_Aplicacao_Externa1()* – ao componente

Gerenciador de Aplicações Externas para que uma aplicação externa seja selecionada. A aplicação externa sendo selecionada, o componente Gerenciador de Aplicações Externas retorna uma mensagem de confirmação – *Selecionar_Aplicação_Externa1_OK()*. Quando o recurso é alocado, o componente Gerenciador de Alocação de Recurso envia uma mensagem de confirmação – *Alocar_Recurso_OK()* – ao componente Gerenciador de *Workflow*. A tarefa codificada e os recursos alocados são armazenados na agenda do usuário do *workflow*, através de uma comunicação – *Agendar_Tarefa()* – com o componente Gerenciador de Tarefas. Logo que a tarefa tiver sido escalonada, o componente Gerenciador de Tarefas envia uma mensagem – *Inserir_Tarefa()*– ao componente *Gerenciador de Objetos do Workflow* solicitando o armazenamento da tarefa no banco de dados do ambiente. Após a tarefa e recursos terem sido armazenados, o componente Gerenciador de Objetos do *Workflow* retorna uma mensagem de confirmação – *Inserir_Tarefa_OK()* – que, por sua vez, também envia uma mensagem – *Agendar_Tarefa_OK()* – ao componente Gerenciador de *Workflow*, que finaliza o processo de alocação de recursos enviando uma mensagem de sucesso – *Selecionar_Recurso_OK()* – ao componente Gerenciador de Interface Gráfica. Esta etapa de codificação e alocação de recursos é repetida para o número de tarefas a serem codificadas. Na simulação realizada neste trabalho, foram realizadas três repetições.

Na última etapa, o usuário do *workflow* através do componente Gerenciador de Interface Gráfica, envia uma mensagem – *Executar_Tarefa()* – ao componente Gerenciador de Tarefas solicitando uma conexão para executar uma tarefa. Quando a conexão é estabelecida, o componente Gerenciador de Tarefas envia uma mensagem – *Selecionar_Tarefa()* – ao componente Gerenciador de Objetos do *Workflow* para que a tarefa a ser executada seja selecionada. Quando a tarefa é selecionada, uma mensagem de confirmação – *Selecionar_Tarefa_OK()*– é retornada ao componente Gerenciador de Tarefas, que logo em seguida, se comunica – *Executar_Script()*– com o componente Interpretador para que se inicie o processo de execução. O componente Interpretador ativa uma ferramenta externa a ser utilizada na execução da tarefa enviando uma mensagem – *Invocar_Aplicação_Externa()* – ao componente Gerenciador de Aplicações Externas. Quando a ferramenta é ativada, uma mensagem de confirmação – *Invocar_Aplicação_Externa_OK()* – é retornada ao componente Interpretador que continua seu processo de execução. Após a tarefa ter sido executada, os resultados precisam ser armazenados. Assim, o componente *Interpretador* envia uma mensagem – *Inserir_Tarefa_Executada()* – ao componente Gerenciador de Objetos do *Workflow* que retorna uma mensagem de confirmação – *Inserir_Tarefa_Executada_OK()*. Logo, o componente Interpretador também envia uma mensagem – *Executar_Script_OK()* – ao componente Gerenciador de Tarefas, que finaliza o processo de execução da tarefa enviando uma mensagem de sucesso – *Executar_Tarefa_OK()*– ao componente Gerenciador de Interface Gráfica.

A Figura 4.17 apresenta o código da declaração do componente Gerenciador de Tarefas utilizado na simulação da arquitetura de linha de produto para WfMS proposta.

```
TYPE GERENCIADOR_TAREFAS is INTERFACE
action
  in Agendar_Tarefa();
  out Inserir_Tarefa();
  in Inserir_tarefa_OK();
  out Agendar_Tarefa_OK();
  in Executar_Tarefa1();
  out Selecionar_Tarefa();
  in Selecionar_Tarefa_OK();
  out Executar_Script();
  in Executar_Script_OK();
  out Executar_Tarefa1_OK();

BEHAVIOR
  action animation_Iam (name:string);

BEGIN
  start => animation_Iam("GERENCIADOR_TAREFAS");;
  Agendar_Tarefa() => Inserir_Tarefa();;
  Inserir_Tarefa_OK() => Agendar_Tarefa_OK();;
  Executar_Tarefa1() => Selecionar_Tarefa();;
  Selecionar_Tarefa_OK() => Executar_Script();;
  Executar_Script_OK() => Executar_Tarefa1_OK();;
END;
```

FIGURA 4.17 – Código Rapide do Componente Gerenciador de Tarefas

De acordo com o código apresentado na Figura 4.17 temos um tipo de interface GERENCIADOR_TAREFAS. Ela define a interface do objeto (componente) Gerenciador de Tarefas. Esta interface define as características assíncronas de comunicação (*actions*) entre os objetos. A interface GERENCIADOR_TAREFAS contém dez ações, uma ação **in** Agendar_Tarefa(), uma ação **out** Inserir_Tarefa(), uma ação **in** Inserir_Tarefa_OK(), uma ação **out** Agendar_Tarefa_OK(), uma ação **in** Executar_Tarefa(), uma ação **out** Selecionar_Tarefa(), uma ação **in** Selecionar_Tarefa_OK(), uma ação **out** Executar_Script(), uma ação **in** Executar_Script_OK(), e por fim, uma ação **out** Executar_Tarefa_OK(). Objetos deste tipo podem gerar eventos, tais como: Inserir_Tarefa(), Agendar_Tarefa_OK(), Selecionar_Tarefa(), Executar_Script() e Executar_Script_OK(); e podem receber eventos como: Agendar_Tarefa(), Inserir_Tarefa_OK(), Executar_Tarefa(), Selecionar_Tarefa_OK() e Executar_Script_OK().

A interface GERENCIADOR_TAREFAS também contém regras de transição de comportamento reativa, que definem a seqüência de eventos entre os objetos. A Tabela 4.1 apresenta a seqüência de eventos entre os objetos. A coluna Recebe apresenta as chamas recebidas pelo componente GERENCIADOR_TAREFAS e a coluna Reage apresenta os eventos requisitados pelo componente.

TABELA 4.1 – Seqüência de eventos do componente GERENCIADOR_TAREFAS

Recebe	Reage
Agendar_Tarefa()	Inserir_Tarefa()
Inserir_Tarefa_OK()	Agendar_Tarefa_OK()
Executar_Tarefa1()	Selecionar_Tarefa()
Selecionar_Tarefa_OK()	Executar_Script()
Executar_Script_OK()	Executar_Tarefa1_OK()

De acordo com a regra de comportamento, espera-se até que o evento Start seja observado e reage gerando o evento *animation_Iam* recebe o evento Agendar_Tarefa(), que reage gerando o evento **out** Inserir_Tarefa(); recebe o evento Inserir_Tarefa_OK() e reage gerando o evento **out** Agendar_Tarefa_OK(); recebe o evento Executar_Tarefa() e reage gerando o evento **out** Selecionar_Tarefa(); recebe o evento Selecionar_Tarefa_OK() e reage gerando o evento **out** Executar_Script(); recebe o evento Executar_Script() e reage gerando o evento **out** Executar_Tarefa_OK(). O evento *animation_Iam* é parametrizado com a string "GERENCIADOR_TAREFAS".

Baseado na simulação da arquitetura proposta, o componente Gerenciador de Tarefas estabelece uma comunicação direta com os componentes Gerenciador de *Workflow*, Interpretador e Gerenciador de Objetos do *Workflow*. O código completo da simulação da arquitetura proposta está documentado no Anexo 1.

4.5.1 Dificuldades Encontradas

A arquitetura de linha de produto proposta para sistemas de gerenciamento de *workflow* foi simulada utilizando a linguagem Rapide e suas ferramentas, conforme apresentado na seção 4.5. A animação da simulação permitiu visualizar a execução das atividades do gerente de *workflow*, sendo este o responsável por controlar a criação, modificação e execução dos *workflows*. Além disso, a modelagem do comportamento dinâmico descrito pela arquitetura de linha de produto proposta possibilitou avaliar a comunicação entre os componentes e o efeito geral das funcionalidades do sistema.

Porém, a simulação não foi suficiente para validar completamente o modelo proposto. A validação completa ainda depende do estabelecimento de critérios apropriados que permitam quantificar o comportamento do sistema real.

Segundo Brown [BRO 96], a chave para a validação baseada em simulação é criar testes que exercitem um erro, de modo a torná-lo aparente para o usuário. Atualmente, testes são criados manualmente e direcionados para casos de teste particulares ou são gerados de modo aleatório na esperança de alcançar situações não testadas de modo probabilístico. Ambos os métodos falham em fornecer uma medida confiável de que um sistema complexo tenha sido testado de modo adequado.

Portanto, a simulação, por sua vez, não faz com que a atividade de teste seja facilmente avaliada e quantificada. Dessa maneira, uma outra forma de se validar modelos arquiteturais é através da aplicação do critério Análise de Mutantes, da técnica de teste baseada em erros, que permite avaliar a atividade de teste de forma quantitativa, através do escore de mutação.

A análise de Mutantes tem por objetivo avaliar a qualidade de um conjunto de casos de teste **T** em relação a um programa **P**. Para isso, utiliza um conjunto de programas ligeiramente diferentes de **P**, denominados mutantes de **P**, e busca obter um conjunto de casos de teste que consiga revelar as diferenças de comportamento existentes entre **P** e seus mutantes. Os mutantes são gerados através de alterações no programa original. Essas alterações são feitas com base em um conjunto de operadores denominados operadores de mutação. A cada operador pode-se associar um tipo ou uma classe de erros que se pretende revelar no programa [CAR 99].

Dessa maneira, uma proposta para trabalhos futuros seria aplicar técnicas de teste, entre os quais o critério Análise de Mutantes, de forma a validar a arquitetura de linha de produto proposta para sistemas de gerenciamento de *workflow*. Para isso,

pretende-se criar critérios para que a Análise de Mutantes seja aplicada, tais como: criar operadores de mutação para Rapide; analisar as seqüências de eventos e sincronização, mais especificamente, a troca de mensagens entre os componentes; comparar os *posets* gerados durante a simulação; criar ferramentas para alterar o Rapide. Isto é possível e, como exemplo, podemos citar a utilização do teste de mutação na validação de modelos baseado em Máquinas de Estados [FAB 94] [MAL 98], Redes de Petri [FAB 95]. Takano [TAK 2002] implementará estas técnicas como continuação da avaliação da arquitetura proposta neste trabalho.

4.6 Considerações Finais

Este capítulo apresentou a arquitetura de linha de produto para sistemas de gerenciamento de *workflow* proposta. O processo seguido para a definição da arquitetura de linha de produto e a notação utilizada para a representação das variabilidades foram descritos.

O processo proposto neste trabalho envolveu a exploração do domínio por meio do modelo de referência e da arquitetura genérica para WfMS da WfMC, o padrão *Process Manager* [GIM 99b], o método Catalysis [D'SO 99] e a linguagem Rapide [CSL 97] para simulação e avaliação da arquitetura. O método Catalysis foi escolhido porque é uma abordagem geral de desenvolvimento baseado em componentes, que envolve conceitos de arquitetura de software, *frameworks* e padrões. Catalysis é baseado em UML [BOO 99] [FOW97] [RUM 99]. Assim, foi aplicado um método de propósito geral, examinando as alterações necessárias para linhas de produto ao invés de se criar um método específico para esta abordagem.

As abordagens de engenharia de domínio existentes fornecem recursos para representação e gerenciamento de mecanismos de variabilidade, porém são mais deficientes na representação e implementação de arquiteturas de componentes.

O processo proposto é composto das seguintes fases: análise dos requisitos, especificação do sistema, projeto da arquitetura, projeto interno dos componentes e validação da arquitetura. Este capítulo apresentou cada uma dessas fases e apresentou os artefatos gerados em cada uma delas.

Dentre os trabalhos relacionados a este, pode-se destacar a implementação do componente agenda de tarefas por Tanaka [TAN 2000b] e a proposta de um *enterprise framework* para WfMS proposto por Gimenes [GIM 01].

O capítulo seguinte apresenta as conclusões deste trabalho e os trabalhos futuros que podem ser realizados a partir desta experiência.

5 Conclusões e Trabalhos Futuros

As tecnologias de *workflow* e gerenciamento de *workflow* estão em contínuo crescimento e vêm sendo exploradas cada vez mais em diversos domínios de negócios diferentes. Muitos produtos de gerenciamento de *workflow* estão disponíveis no mercado, cada um deles dando ênfase em suas características particulares.

É evidente a necessidade existente atualmente nas organizações de aumentarem sua produtividade e reduzir o custo do desenvolvimento de software. A engenharia de software vem buscando desenvolver métodos, técnicas, ferramentas e notações que sirvam de suporte para alcançar estes objetivos. Uma técnica importante para melhorar a qualidade e a produtividade do processo de software é a reutilização de soluções. Conceitos como arquitetura de software, desenvolvimento baseado em componentes, engenharia de domínio, padrões e frameworks estão sendo estudados e aprimorados para que possam ser aplicados na produção de uma família de produtos. Esses conceitos oferecem mecanismos que facilitam a especificação e a construção dos elementos de uma linha de produto de software. Esses elementos incluem a arquitetura da linha de produto e os componentes que preenchem esta arquitetura.

Uma linha de produto é 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 [GIM 2002].

A aplicação da técnica de linha de produto apresenta uma importante abordagem para aumentar a reutilização de software, diminuir o *time-to-market* e a necessidade de manutenção desses produtos por parte das companhias de software. Esta técnica é aplicável aos domínios em que existe uma demanda por produtos específicos, no entanto existe um conjunto de características comuns e pontos de variabilidade bem definidos. O domínio dos WfMS é altamente favorável à aplicação da abordagem de linha de produtos, devido aos esforços da WfMC, que viabilizou a construção de uma arquitetura genérica para WfMS que pode ser ajustada à maioria dos produtos do mercado.

Esta dissertação propõe uma arquitetura de linha de produto para sistemas de gerenciamento de *workflow*. O desenvolvimento da arquitetura tomou como base a arquitetura genérica e o modelo de referência da WfMC e o padrão de arquitetura *Process Manager* desenvolvido no contexto do projeto ExPSEE. O projeto da arquitetura seguiu o processo sugerido pelo Catalysis [D'SO 99] com algumas modificações para representar variabilidade. As extensões foram aplicadas com base nos diagramas de caso de uso propostos por Jacobson [JAC 97] e extensão da notação UML proposta por Morisio [MOR 2000]. Essas modificações foram necessárias porque a abordagem Catalysis não possui recursos para representar variabilidade. O processo proposto é composto das seguintes fases: análise dos requisitos, especificação do sistema, projeto da arquitetura, projeto interno dos componentes e validação da arquitetura.

Catalysis foi utilizado por ser uma abordagem geral de DBC, baseado em UML [BOO 99] [FOW 97] [RUM 99], que envolve conceitos de arquitetura de software, *frameworks* e padrões. Assim, o objetivo foi utilizar um método de DBC, examinando as alterações necessárias ao invés de se criar um método específico para linha de produto. Apesar das abordagens de engenharia de domínio fornecerem recursos para representação de variabilidade, essas abordagens são mais deficientes na representação

e implementação de arquiteturas de componentes. Assim, os métodos de DBC surgem como uma boa solução para as fases de projeto e implementação da linha de produto. Iniciativas similares já podem ser encontradas na literatura, tais como as abordagens Kobra [ATK 2000] e GenVoca [BAT 98].

Porém, para fins de avaliação da arquitetura proposta optou-se por utilizar a ADL Rapide que possui um conjunto de ferramentas de suporte à simulação de arquiteturas. A escolha desta linguagem permitiu a simulação, na qual foi possível modelar o comportamento dinâmico da arquitetura, possibilitando avaliar a comunicação entre os componentes e o efeito geral das funcionalidades do sistema sem que houvesse necessidade de aprofundamento em detalhes de implementação.

A principal contribuição deste trabalho é a arquitetura de linha de produto para sistemas de gerenciamento de *workflow*. Esta arquitetura pode ser usada para facilitar o processo de produção de diferentes produtos de *workflow* que possuam características comuns, mas que também têm aspectos diferentes de acordo com as necessidades da indústria. Pode-se destacar também contribuições para a sistematização de um processo de desenvolvimento de arquiteturas de linha de produto e também um melhor entendimento dos conceitos e abordagens relacionados à prática de linha de produto. Baseado neste processo é possível definir a arquitetura de software utilizando a notação UML com extensões para representação de variabilidade. A exploração de uma ADL para especificar e validar arquiteturas de linha de produto pode também ser vista como contribuição deste trabalho.

A abordagem de linha de produto de software é recente na comunidade de engenharia de software, porém é vista como uma área muito promissora, pois oferece uma maneira sistemática, planejada e prática de reutilização de software. Ainda existe uma grande controvérsia sobre este enfoque, considerando técnicas anteriores com propósitos similares, principalmente: engenharia de domínio e *frameworks*. O que parece mais produtivo é considerar o enfoque de linha de produto e essas técnicas como complementares. O enfoque de linha de produto pode ser visto como uma forma de organização dessas técnicas que nasceu da necessidade evidente de organizações que tinham uma alta demanda por seus produtos, mas que não conseguiriam atender essa demanda se não fosse encontrada uma forma de gerenciar sua família de produtos. Poulin [POU 97] afirma que linha de produto e domínio representam o mesmo nível de abstração e que o termo linha de produto foi criado para facilitar a comunicação entre o pessoal técnico e os gerentes que pensam em termos de produtos. O termo linha de produto parece ser mais adequado para o contexto industrial. Os conceitos de engenharia de domínio são amplamente utilizados no enfoque de linha de produto.

Dentre os trabalhos futuros provenientes deste esforço pode-se destacar o povoamento da arquitetura explorando as variabilidades de cada componente e suas conexões. Assim, seria possível a realização de experimentos para gerar produtos de *workflow* a partir da arquitetura proposta. Tanaka [TAN 99a] implementou um framework da agenda de tarefas para WfMS, porém não foi empregada a abordagem de linha de produto. Outro trabalho que pode ser proveniente deste seria a construção de um ambiente para desenvolvimento de linhas de produto (ferramentas de apoio) e que permitisse a geração de produtos. Pode-se dizer ainda que a questão de como representar variabilidades durante o processo e como tomar decisões sobre qual deve ser selecionada ainda não é um tópico fechado, sendo necessário um aprofundamento maior nesta questão.

Os resultados deste trabalho serão apresentados no XVI SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE [LAZ 2002], por meio do artigo apresentado no anexo 2.

Anexo 1 Código-Fonte da Simulação

```
TYPE GERENCIADOR_ARQ_WF is INTERFACE
action
  in Criar_Arquitetura();
  out Selecionar_Aplicacao_Externa();
  in Selecionar_Aplicacao_Externa_OK();
  out Inserir_Arquitetura();
  in Inserir_Arquitetura_OK();
  out Criar_Arquitetura_OK();
  in Selecionar_Arquitetura1();
  out Selecionar_Arquitetura2();
  in Selecionar_Arquitetura2_OK();
  out Selecionar_Arquitetura1_OK();

BEHAVIOR
  action animation_Iam(name: string);

BEGIN
  start => animation_Iam("GERENCIADOR_ARQ_WF");;
  Criar_Arquitetura() => Selecionar_Aplicacao_Externa();;
  Selecionar_Aplicacao_Externa_OK() => Inserir_Arquitetura();;
  Inserir_Arquitetura_OK() => Criar_Arquitetura_OK();;
  Selecionar_Arquitetura1() => Selecionar_Arquitetura2();;
  Selecionar_Arquitetura2_OK() => Selecionar_Arquitetura1_OK();;
END;


TYPE GERENCIADOR_AP_EXT is INTERFACE
action
  in Selecionar_Aplicacao_Externa();
  out Selecionar_Aplicacao_Externa_OK();
  in Selecionar_Aplicacao_Externa1();
  out Selecionar_Aplicacao_Externa1_OK();
  in Invocar_Aplicacao_Externa();
  out Invocar_Aplicacao_Externa_OK();

BEHAVIOR
  action animation_Iam(name: string);

BEGIN
  start => animation_Iam("GERENCIADOR_AP_EXT");;
  Selecionar_Aplicacao_Externa() => Selecionar_Aplicacao_Externa_OK();;
  Selecionar_Aplicacao_Externa1() => Selecionar_Aplicacao_Externa1_OK();;
  Invocar_Aplicacao_Externa() => Invocar_Aplicacao_Externa_OK();;
END;

TYPE GERENCIADOR_INTERFACE is INTERFACE
action
  out Criar_Arquitetura();
  in Criar_Arquitetura_OK();
  out Instanciar_Arquitetura();
  in Instanciar_Arquitetura_OK();
  out Programar_Tarefa();
```

```
  in Programar_Tarefa_OK();
  out Selecionar_Recurso();
  in Selecionar_Recurso_OK();
  out Executar_Tarefa();
  in Executar_Tarefa_OK();

BEHAVIOR
  action animation_Iam(name: string);
  i : var Integer := 0;

BEGIN
  start => animation_Iam("GERENCIADOR_INTERFACE") -> Criar_Arquitetura();;
  Criar_Arquitetura_OK() => Instanciar_Arquitetura();;
  Instanciar_Arquitetura_OK() => Programar_Tarefa();;
  Programar_Tarefa_OK() => Selecionar_Recurso();;
  Selecionar_Recurso_OK() =>
    if ($i < 2) then
      i:=$i+1;
      Programar_Tarefa();
    elsif ($i = 2) then
      Executar_Tarefa();
    end if;;
END;

TYPE GERENCIADOR_OBJ_WF is INTERFACE
action
  in Inserir_Arquitetura();
  out Inserir_Arquitetura_Ok();
  in Selecionar_Arquitetura2();
  out Selecionar_Arquitetura2_OK();
  in Inserir_Tarefa();
  out Inserir_Tarefa_OK();
  in Selecionar_Tarefa();
  out Selecionar_Tarefa_OK();
  in Inserir_Tarefa_Executada();
  out Inserir_Tarefa_Executada_OK();

BEHAVIOR
  action animation_Iam(name: string);

BEGIN
  start => animation_Iam("GERENCIADOR_OBJ_WF");;
  Inserir_Arquitetura() => Inserir_Arquitetura_OK();;
  Selecionar_Arquitetura2() => Selecionar_Arquitetura2_OK();;
  Inserir_Tarefa() => Inserir_Tarefa_OK();;
  Selecionar_Tarefa() => Selecionar_Tarefa_OK();;
  Inserir_Tarefa_Executada() => Inserir_Tarefa_Executada_OK();;
END;

TYPE GERENCIADOR_WF is INTERFACE
action
  in Instanciar_Arquitetura();
  out Selecionar_Arquitetura1();
  in Selecionar_Arquitetura1_OK();
  out Instanciar_Arquitetura_OK();
```

```
  in Programar_Tarefa();
  out Programar_Tarefa_OK();
  in Selecionar_Recurso();
  out Alocar_Recurso();
  in Alocar_Recurso_OK();
  out Agendar_Tarefa();
  in Agendar_Tarefa_OK();
  out Selecionar_Recurso_OK();

BEHAVIOR
  action animation_Iam (name:string);

BEGIN
  start => animation_Iam("GERENCIADOR_WF");;
  Instanciar_Arquitetura() => Selecionar_Arquitetura1();;
  Selecionar_Arquitetura1_OK() => Instanciar_Arquitetura_OK();;
  Programar_Tarefa() => Programar_Tarefa_OK();;
  Selecionar_Recurso() => Alocar_Recurso();;
  Alocar_Recurso_OK() => Agendar_Tarefa();;
  Agendar_Tarefa_OK() => Selecionar_Recurso_OK();;
END;

TYPE GERENCIADOR_ALOC_REC is INTERFACE
action
  in Alocar_Recurso();
  out Selecionar_Aplicacao_Externa1();
  in Selecionar_Aplicacao_Externa1_OK();
  out Alocar_Recurso_OK();

BEHAVIOR
  action animation_Iam(name: string);

BEGIN
  start => animation_Iam("GERENCIADOR_ALOC_REC");;
  Alocar_Recurso() => Selecionar_Aplicacao_Externa1();;
  Selecionar_Aplicacao_Externa1_OK() => Alocar_Recurso_OK();;
END;


TYPE GERENCIADOR_TAREFAS is INTERFACE
action
  in Agendar_Tarefa();
  out Inserir_Tarefa();
  in Inserir_tarefa_OK();
  out Agendar_Tarefa_OK();
  in Executar_Tarefa1();
  out Selecionar_Tarefa();
  in Selecionar_Tarefa_OK();
  out Executar_Script();
  in Executar_Script_OK();
  out Executar_Tarefa1_OK();

BEHAVIOR
  action animation_Iam (name:string);
```

```
BEGIN
  start => animation_Iam("GERENCIADOR_TAREFAS");;
  Agendar_Tarefa() => Inserir_Tarefa();;
  Inserir_Tarefa_OK() => Agendar_Tarefa_OK();;
  Executar_Tarefa1() => Selecionar_Tarefa();;
  Selecionar_Tarefa_OK() => Executar_Script();;
  Executar_Script_OK() => Executar_Tarefa1_OK();;
END;

TYPE INTERPRETADOR is INTERFACE
action
  in Executar_Script();
  out Invocar_Aplicacao_Externa();
  in Invocar_Aplicacao_Externa_OK();
  out Inserir_Tarefa_Executada();
  in Inserir_Tarefa_Executada_OK();
  out Executar_Script_OK();

BEHAVIOR
  action animation_Iam(name: string);

BEGIN
  start => animation_Iam("INTERPRETADOR");;
  Executar_Script() => Invocar_Aplicacao_Externa();;
  Invocar_Aplicacao_Externa_OK() => Inserir_Tarefa_Executada();;
  Inserir_Tarefa_Executada_OK() => Executar_Script_OK();;
END;

TYPE GERENCIADOR_EXE_WF is INTERFACE
action
  in Executar_Tarefa();
  out Executar_Tarefa1();
  in Executar_Tarefa1_OK();
  out Executar_Tarefa_OK();

BEHAVIOR
  action animation_Iam(name: string);

BEGIN
  start => animation_Iam("GERENCIADOR_EXE_WF");;
  Executar_Tarefa() => Executar_Tarefa1();;
  Executar_Tarefa1_OK() => Executar_Tarefa_OK();;
END;

ARCHITECTURE MAIN () is
  GI  : GERENCIADOR_INTERFACE;
  GA  : GERENCIADOR_ARQ_WF;
  GAP : GERENCIADOR_AP_EXT;
  GOB : GERENCIADOR_OBJ_WF;
  GWF : GERENCIADOR_WF;
  GAL : GERENCIADOR_ALOC_REC;
  GT  : GERENCIADOR_TAREFAS;
  GEW : GERENCIADOR_EXE_WF;
  INT : INTERPRETADOR;
```

```
CONNECT
  GI.Criar_Arquitetura() => GA.Criar_Arquitetura();
  GA.Selecionar_Aplicacao_Externa() => GAP.Selecionar_Aplicacao_Externa();
  GAP.Selecionar_Aplicacao_Externa_OK() => GA.Selecionar_Aplicacao_Externa_OK();
  GA.Inserir_Arquitetura() => GOB.Inserir_Arquitetura();
  GOB.Inserir_Arquitetura_OK() => GA.Inserir_Arquitetura_OK();
  GA.Criar_Arquitetura_OK() => GI.Criar_Arquitetura_OK();

  GI.Instanciar_Arquitetura() => GWF.Instanciar_Arquitetura();
  GWF.Selecionar_Arquitetura1() => GA.Selecionar_Arquitetura1();
  GA.Selecionar_Arquitetura2() => GOB.Selecionar_Arquitetura2();
  GOB.Selecionar_Arquitetura2_OK() => GA.Selecionar_Arquitetura2_OK();
  GA.Selecionar_Arquitetura1_OK() => GWF.Selecionar_Arquitetura1_OK();
  GWF.Instanciar_Arquitetura_OK => GI.Instanciar_Arquitetura_OK();
  GI.Programar_Tarefa() => GWF.Programar_Tarefa();
  GWF.Programar_Tarefa_OK() => GI.Programar_Tarefa_OK();
  GI.Selecionar_Recurso() => GWF.Selecionar_Recurso();
  GWF.Alocar_Recurso() => GAL.Alocar_Recurso();
  GAL.Selecionar_Aplicacao_Externa1() => GAP.Selecionar_Aplicacao_Externa1();
  GAP.Selecionar_Aplicacao_Externa1_OK() => GAL.Selecionar_Aplicacao_Externa1_OK();
  GAL.Alocar_Recurso_OK() => GWF.Alocar_Recurso_OK();
  GWF.Agendar_Tarefa() => GT.Agendar_Tarefa();
  GT.Inserir_Tarefa() => GOB.Inserir_Tarefa();
  GOB.Inserir_Tarefa_OK() => GT.Inserir_Tarefa_OK();
  GT.Agendar_Tarefa_OK() => GWF.Agendar_Tarefa_OK();
  GWF.Selecionar_Recurso_OK() => GI.Selecionar_Recurso_OK();

  GI.Executar_Tarefa() => GEW.Executar_Tarefa();
  GEW.Executar_Tarefa1() => GT.Executar_Tarefa1();
  GT.Selecionar_Tarefa() => GOB.Selecionar_Tarefa();
  GOB.Selecionar_Tarefa_OK() => GT.Selecionar_Tarefa_OK();
  GT.Executar_Script() => INT.Executar_Script();
  INT.Invocar_Aplicacao_Externa() => GAP.Invocar_Aplicacao_Externa();
  GAP.Invocar_Aplicacao_Externa_OK() => INT.Invocar_Aplicacao_Externa_OK();
  INT.Inserir_Tarefa_Executada() => GOB.Inserir_Tarefa_Executada();
  GOB.Inserir_Tarefa_Executada_OK() => INT.Inserir_Tarefa_Executada_OK();
  INT.Executar_Script_OK() => GT.Executar_Script_OK();
  GT.Executar_Tarefa1_OK() => GEW.Executar_Tarefa1_OK();
  GEW.Executar_Tarefa_OK() => GI.Executar_Tarefa_OK();
END MAIN;
```

Anexo 2 Artigo aceito para publicação no XVI Simpósio Brasileiro de Engenharia de Software (SBES 2002), 16-18 outubro, 2002, Gramado, RS

Uma Proposta de Arquitetura de Linha de Produtos para Workflow Management Systems

Fabrício R. Lazilha[1,3] Itana M. S. Gimenes[2] R. T. Price[3]
[1]Centro Universitário de Maringá
[2]Universidade Estadual de Maringá
[3]Programa de Pós-Graduação em Computação – Universidade Federal do Rio Grande do Sul
E-mail: fabrício@cesumar.br / itana@din.uem.br / tomprice@inf.ufrgs.br

Resumo

Este artigo apresenta uma proposta de arquitetura de linha de produtos para sistemas de gerenciamento de workflow. O processo seguido para definição da arquitetura de linha de produtos e a notação utilizada para a representação das variabilidades são descritos. O domínio de sistemas de gerenciamento de workflow tem se mostrado altamente favorável à aplicação da abordagem de linha de produtos, pois existe uma arquitetura padrão e uma demanda por produtos similares, porém com características diferentes.

Palavras-chave: Sistema de Gerenciamento de *Workflow*, Linha de produto, Arquitetura de software, Reutilização.

Abstract

This paper proposes a software product line architecture for workflow management systems. It describes the process follow to obtain the architecture as well as the notation used to represent variabilities. The workflow management system domain has proved to be feasible to the application of the product line approach. This is because there is a standard architecture and a call for similar products but which have main different features. The architecture was validated simulating it using RAPIDE tools.

Keywords: Workflow Management Systems, Product Line, Software Architecture, Reuse.

1. Introdução

De acordo com Bass [1], a abordagem de linha de produtos de software cria uma coleção de sistemas que compartilham um conjunto gerenciado de características entre seus principais artefatos. Estes artefatos incluem uma arquitetura base e um conjunto de componentes comuns para preencher esta arquitetura. O projeto de uma arquitetura para uma família de produtos deve considerar as semelhanças e variabilidades entre os produtos desta família.

A abordagem de linha de produtos é aplicável aos domínios em que existe uma demanda por produtos específicos, no entanto existe um conjunto de características comuns e pontos de variabilidade bem definidos. O domínio dos Sistemas de Gerenciamento de *Workflow* (WfMS) é altamente favorável à aplicação da abordagem de linha de produtos, devido aos esforços da *Workflow Management Coalition* (WfMC) [2], que viabilizou a construção de uma arquitetura genérica para WfMS que pode ser

ajustada à maioria dos produtos do mercado. A partir disso a WfMC estabeleceu um modelo de referência para WfMS. Cada implementação de um WfMS pode adaptar componentes ou interfaces de acordo com as necessidades da aplicação. Produtos de *workflow* com características similares, porém com diferentes especificidades são necessários nas mais diversas empresas que utilizam esta tecnologia.

Este artigo apresenta o processo seguido para o projeto de uma arquitetura de linha de produtos para WfMS. Apresenta também as extensões utilizadas para representar variabilidade no processo de desenvolvimento. A seção 2 descreve o processo de desenvolvimento da arquitetura proposta e as extensões utilizadas para a representação da variabilidade nos modelos gerados. A seção 3 apresenta a validação da arquitetura proposta. A seção 4 apresenta trabalhos relacionados. Por fim, a seção 5 apresenta as conclusões.

2. O Processo de Desenvolvimento da Arquitetura

O conceito de arquitetura de linha de produtos é recente e ainda existe uma carência de técnicas que venham a facilitar o processo de desenvolvimento da arquitetura e de seus componentes. Algumas abordagens existentes são a Systhesis que foi documentada pelo *Software Productivity Consortium* [3], *Family-Oriented Abstraction, Specification and Translation* (FAST) [4], *Product Line Software Engineering* (PuLSE) do Centro de Fraunhofer [5] e o *Feature-Oriented Domain Analysis* (FODA) documentado pelo SEI [6].

O processo proposto neste trabalho envolve a exploração do domínio por meio do modelo de referência e da arquitetura genérica para WfMS da WfMC, o método Catalysis [7] e a linguagem Rapide [8] [9] para simulação e avaliação da arquitetura. O método Catalysis foi utilizado porque é uma abordagem geral de desenvolvimento baseado em componentes que envolve conceitos de arquitetura de software, *frameworks* e *patterns*. Catalysis é baseado em UML [10]. Assim, o objetivo é aplicar um método de propósito geral, examinando as alterações necessárias ao invés de se criar um método específico para linha de produtos. As abordagens de engenharia de domínio fornecerem recursos para representação de variabilidade, porém são mais deficientes na representação e implementação de arquiteturas de componentes. Assim, os métodos de DBC surgem como uma boa solução para as fases de projeto e implementação da linha de produto. Iniciativas similares já podem ser encontradas na literatura, tais como Kobra [11] e GenVoca[12].

O processo proposto pelo Catalysis que foi seguido neste trabalho para o desenvolvimento de uma arquitetura de linha de produtos é composto das seguintes fases: análise dos requisitos, especificação do sistema, projeto da arquitetura e projeto interno dos componentes. As seções seguintes apresentam uma caracterização dessas fases, bem como apresenta os artefatos gerados em cada uma delas.

2.1 Especificação de Requisitos

Para a elaboração de uma arquitetura de linha de produtos para WfMS deve-se primeiramente elaborar o modelo do domínio representando objetos e ações do domínio em questão. Nesta fase é possível identificar aspectos comuns entre os produtos e os pontos de variação.

O modelo de referência e a arquitetura genérica para WfMS da WfMC serviram de base para extrair o conjunto de funcionalidades necessárias para os produtos de *workflow*. O modelo e a arquitetura indicaram quais os potenciais componentes de um

WfMS e suas interfaces. Eles foram projetados para que fabricantes de produtos e componentes pudessem desenvolver diversas partes de WfMS independentemente e integrá-las conforme a demanda das organizações. Além disso, são definidas características importantes para que produtos específicos possam ser construídos de acordo com as especificidades dos negócios modelados, por exemplo, *workflow* para produção de software ou *workflow* para administração financeira. A Figura 1 apresenta a arquitetura genérica para WfMS proposta pela WfMC. A partir destes modelos, foram analisadas as possibilidades de extensão da arquitetura e análise de semelhanças e variabilidade. O padrão Process Manager [13] também foi utilizado no desenvolvimento da arquitetura proposta. Este padrão foi desenvolvido para a definição de um gerenciador de processos em PSEEs (Ambiente de Engenharia de Software Orientado a Processo).

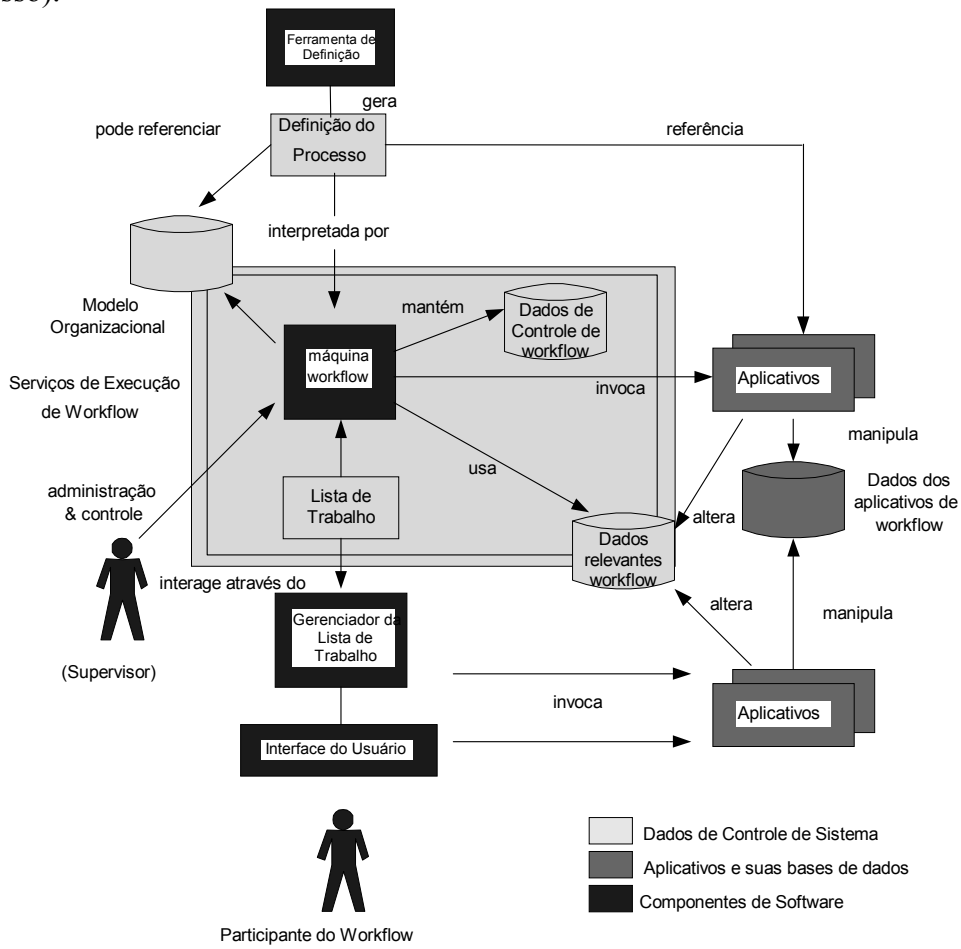

Figura 1 – Arquitetura Genérica para Sistemas de Gerenciamento de *Workflow* [2].

A representação do modelo do domínio pode ser feita utilizando-se objetos e ações, em um nível de abstração independente da eventual solução de software que venha a ser encontrada para o problema. Diagramas de casos de uso da UML dão suporte a esta etapa. As Figuras 2, 3 e 4 apresentam os diagramas de caso de uso para um WfMS. Os objetos representados nos diagramas são o Gerente de Arquitetura de *Workflow*, o Gerente de *Workflow* e o Usuário do *Workflow*, respectivamente.

Durante o desenvolvimento de uma linha de produtos, um dos pontos principais é abstrair e representar as variabilidades associadas à arquitetura e aos seus componentes. A primeira representação de variabilidade ocorreu nesta etapa, nos diagramas de casos de uso. Nesta fase, a notação seguida foi a de representação de

variabilidade em casos de uso proposta por Jacobson et al. [14] que, sugeriu a utilização do esteriótipo <<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 variabilidade, como pode ser observado nas Figuras 2, 3 e 4. Como exemplo, pode-se observar na Figura 2 o caso de uso Definir Arquitetura possui um ponto de variação, que representa neste caso a característica opcional de permitir alteração dinâmica da arquitetura. O mesmo pode ser observado para o caso de uso Definir tipo Ferramenta, que possui duas extensões: tipo interna e tipo externa. Outras representações de variabilidade foram necessárias nos modelos definidos na fase de especificação do sistema, como serão apresentadas mais adiante na seção 2.2.

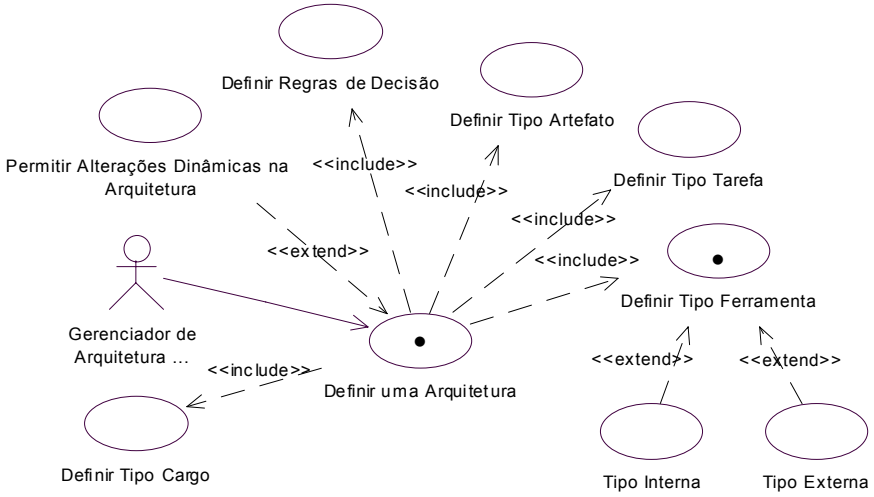

Figura 2 – Diagrama de Caso de Uso do Gerente de Arquitetura de *Workflow*.

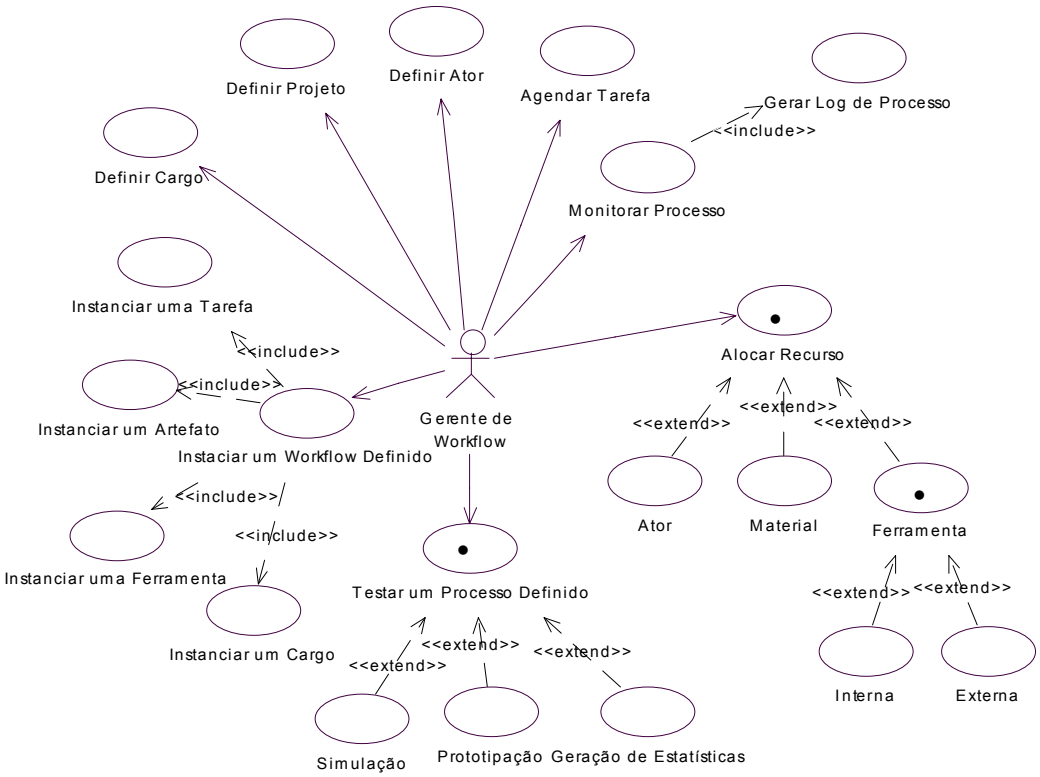

Figura 3 – Diagrama de Caso de Uso do Gerente de *Workflow*.

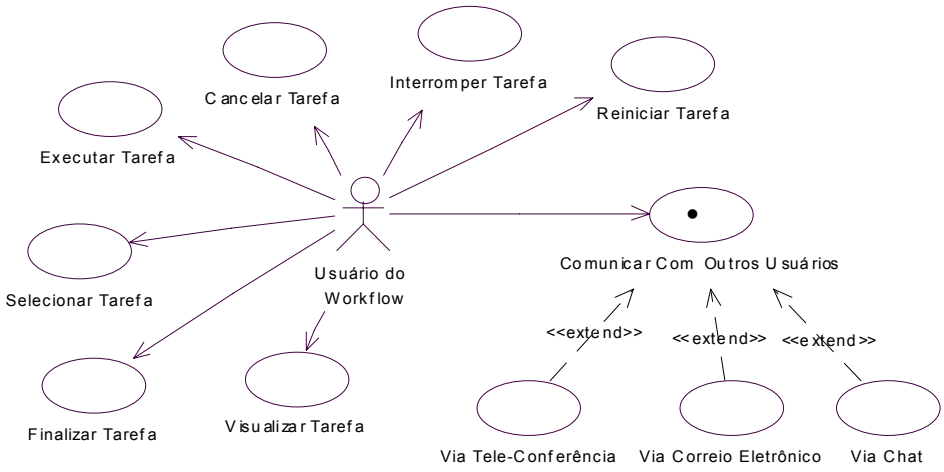

Figura 4 – Diagrama de Caso de Uso do Usuário do *Workflow*.

2.2. Especificação do Sistema

Nesse estágio do desenvolvimento deve-se tratar da modelagem da solução de software identificada nos modelos de domínio obtidos na fase de especificação de requisitos. A análise das ações do sistema representadas nos diagramas de caso de uso torna possível a identificação dos tipos, permitindo a associação das referidas ações aos respectivos tipos relacionados. Tipos são especificações de comportamento dos objetos que documentam e mostram somente a visão externa de um determinado objeto. O artefato mais importante extraído neste estágio é o diagrama estático de tipos.

Outras representações de variabilidade foram necessárias nos modelos definidos nesta fase do processo, como por exemplo, para o diagrama estático de tipos apresentado na Figura 5. Morisio [15] estendeu a notação UML com um esteriótipo de variabilidade, indicado pela notação <<V>>. Como o esteriótipo <<V>> foi usado no contexto da notação orientada a objetos, este esteriótipo está relacionado aos conceitos como especialização e agregação. Como se pode verificar na Figura 5, o <<V>> indica tipos que podem variar conforme as características dos produtos. Por exemplo, o esteriótipo <<V>> foi usado para representar variabilidade no tipo recurso, que pode ser especializado em *material, ator* e *ferramenta*. O tipo *ferramenta* por sua vez, pode ser especializado em *interna* e *externa*.

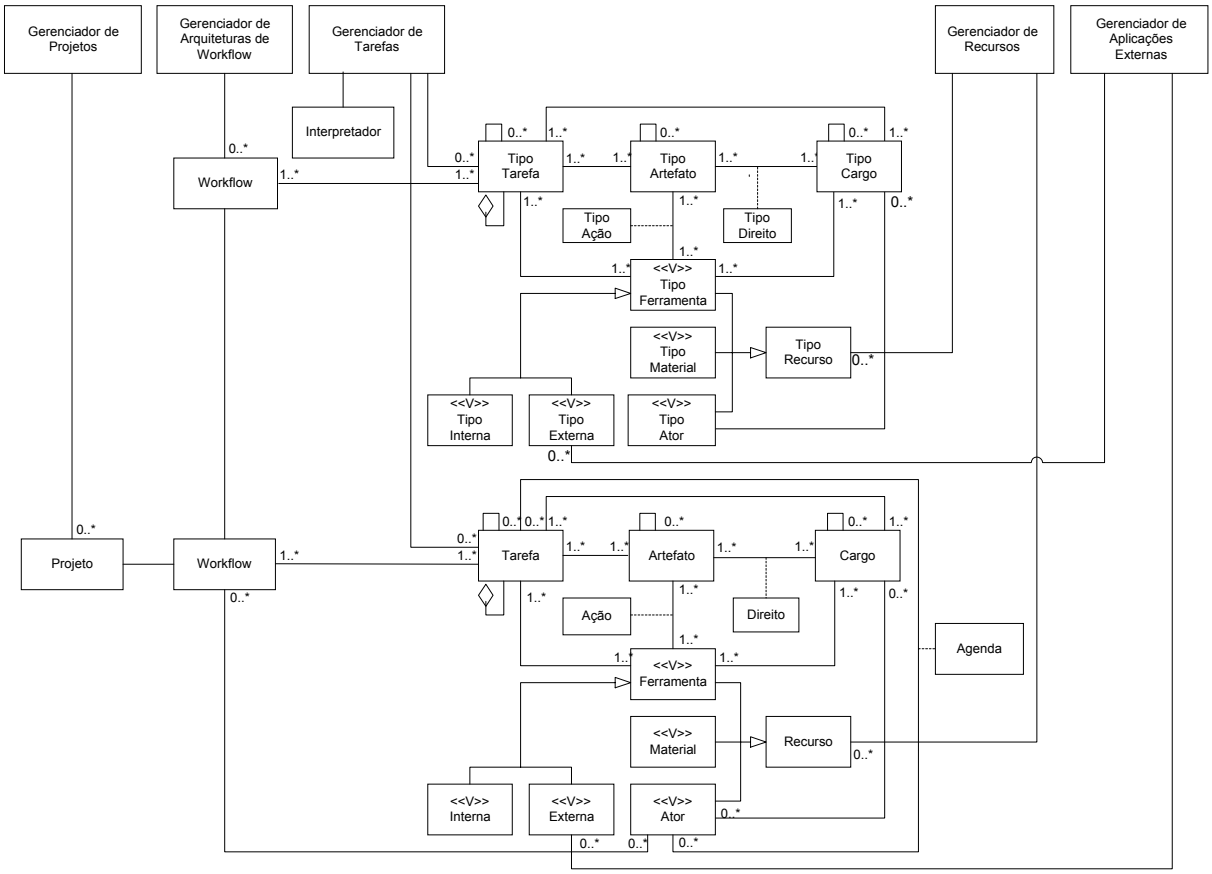

Figura 5 – Diagrama Estático de Tipos para Sistemas de Gerenciamento de *Workflow*.

2.3 Projeto da Arquitetura

Após a modelagem de tipos, uma seqüência de refinamentos desde os níveis mais altos até o nível de componentes deve ser realizado. Catalysis identifica os pacotes como a unidade de decomposição de mais alto nível, considerando que eles representam uma parte do sistema que pode ser tratada de forma independente com o explícito estabelecimento de dependências do restante do sistema. Esses pacotes podem ser obtidos através da análise e refinamento do modelo de negócios e seus relacionamentos podem ser feitos através de um esquema de importação. O particionamento dos pacotes segue a abordagem de camadas verticais e horizontais proposta por Catalysis. As camadas verticais visam refletir o fato de que usuários diferentes têm diferentes visões de um sistema ou negócio. Os usuários aqui são o projetista de *workflow*, que define a arquitetura do *workflow*; o gerente de *workflow*, responsável pelas atividades de supervisão e administração e, por fim, o usuário que executa as tarefas do *workflow*. Essas três visões são representadas pelos pacotes *Gerenciador de Arquitetura de Workflow*, Gerenciador de *Workflow* e *Gerenciador de Execução do Workflow*, respectivamente. A Figura 6 apresenta o diagrama de camadas verticais de alto nível.

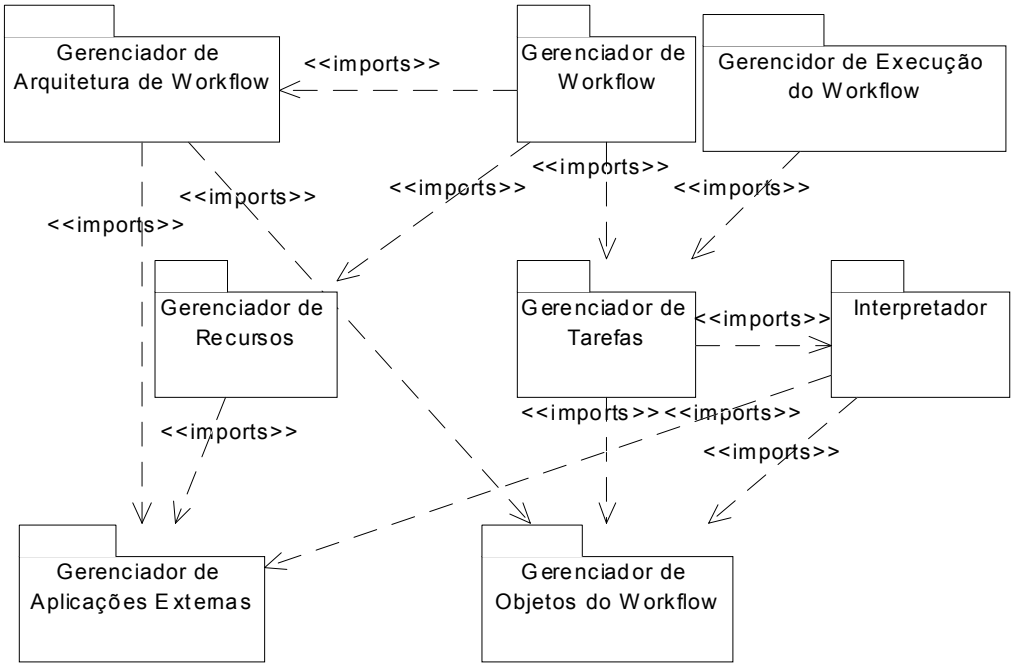

Figura 6 – Diagrama de Camadas Verticais de Alto Nível.

As camadas horizontais visam organizar os pacotes em vários níveis, desde as ações de mais alto nível até as de infra-estrutura. A divisão deve buscar a otimização da importação de pacotes. O relacionamento entre os elementos do diagrama de pacotes e a arquitetura genérica dos WfMSs são representados pelos pacotes Gerenciador de Arquitetura de *Workflow*, Gerenciador de *Workflow*, Gerenciador de Execução do *Workflow*, Gerenciador de Recursos, Gerenciador de Tarefas, Interpretador, Gerenciador de Aplicações Externas e Gerenciador de Objetos do *Workflow*.

A partir do diagrama de camadas verticais de alto nível, foi possível projetar o Diagrama de Camadas Verticais. A construção deste modelo finaliza o processo de particionamento em componentes, apresentando os pacotes definidos anteriormente pelo Diagrama de Camadas Verticais de Alto Nível juntamente com a especificação dos tipos de cada pacote e o relacionamento entre eles. A Figura 7 apresenta o pacote *Gerenciador de Tarefas*, que é parte do Diagrama de Camadas Verticais, juntamente com a especificação de tipos deste pacote. O diagrama completo não é mostrado por questão de espaço.

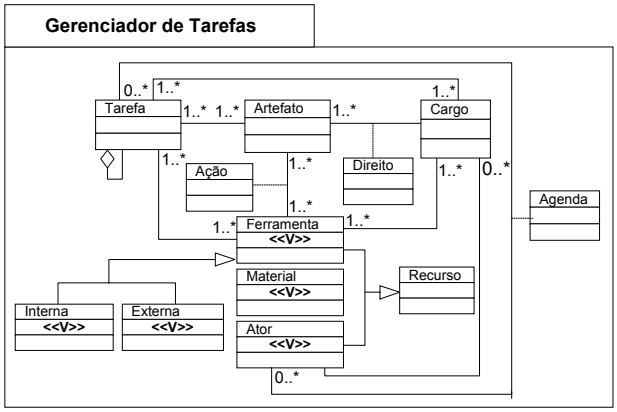

Figura 7 – Especificação do Pacote Gerenciador de Tarefas

2.3.1 Arquitetura de Componentes

O Diagrama de Camadas Verticais apresenta-se como o resultado da identificação e especificação de componentes. Os componentes são representados através de pacotes genéricos, constituídos por tipos e relacionamentos. A arquitetura de componentes para WfMS foi então projetada a partir deste diagrama, conforme apresentado na Figura 8.

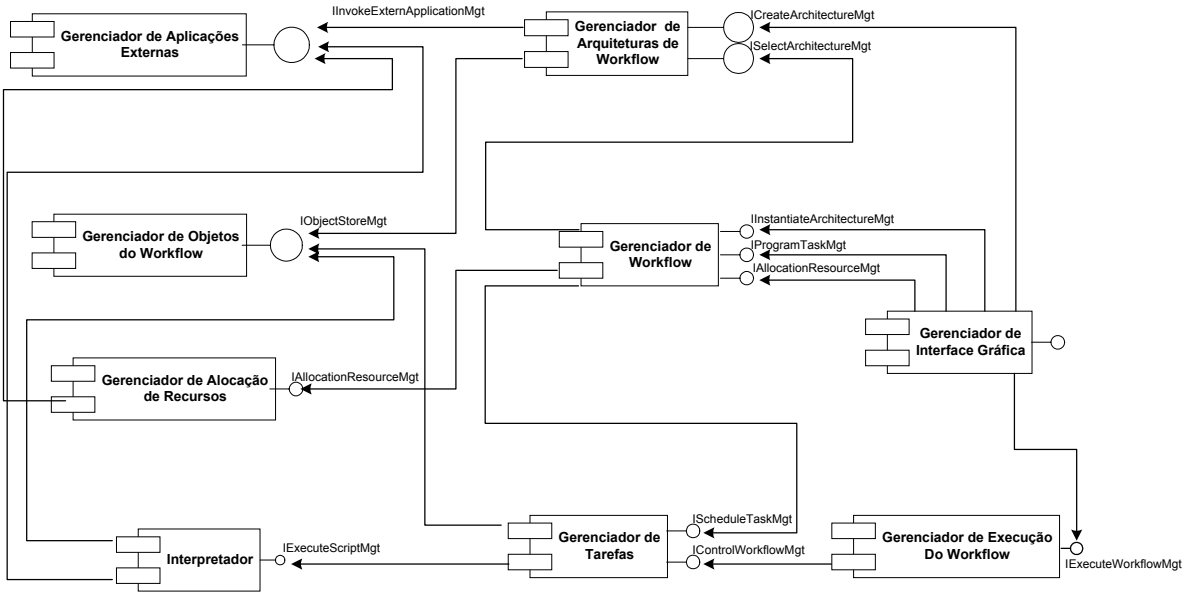

Figura 8 – Arquitetura de Componentes para WfMS.

Uma descrição mais detalhada sobre a funcionalidade dos componentes da arquitetura e aspectos de variabilidade associados a estes componentes serão apresentados mais adiante na seção 2.3.2. Os itens seguintes trazem uma descrição sucinta de cada um dos componentes da arquitetura proposta com suas respectivas interfaces e métodos:

Gerenciador de Interface Gráfica: Este componente é responsável pela interação com o usuário do WfMS.

Gerenciador de Arquitetura de Workflow: Este componente é responsável pelo controle e gerenciamento da construção e manutenção de arquiteturas de *workflow*.

ICriarArquiteturaMgt

SolicitarConexão()	Permite a conexão de um usuário
CriarObjeto()	Cria um novo objeto de definição de *workflow* (ex. um ator, cargo, uma ferramenta)
ExcluirObjeto()	Exclui um objeto de definição de *workflow*
RecuperarObjeto()	Recupera um objeto de definição de *workflow*
InserirAtributoObjeto()	Insere um atributo específico em um objeto de definição
DeletarAtributoObjeto()	Exclui um atributo específico em um objeto de definição
ArmazenarArquitetura()	Salva uma arquitetura de referência

IselecionarArquiteturaMgt

SelecionarArquitetura()	Seleciona uma arquitetura de *workflow*

Gerenciador de Workflow: Este componente é responsável pela criação e gerenciamento de *workflows*.

IInstanciarArquiteturaMgt

SolicitarConexão()	Permite a conexão de um usuário
SelecionarArquitetura()	Seleciona uma arquitetura de *workflow* para instanciação
InstanciarObjeto()	Instancia objetos definidos na arquitetura de *workflow* selecionada

IProgramarTarefaMgt

SelecionarProjeto()	Seleciona projeto a ser executado
SelecionarTarefa()	Seleciona tarefa a ser codificada
SelecionarCargo()	Seleciona cargo ao qual a tarefa a ser codificada está associada
SelecionarAtor()	Seleciona ator para a execução da tarefa
ProgramarTarefa()	Programa uma determinada tarefa

ISelecionarRecursoMgt

SelecionarRecurso()	Seleciona recursos que serão usados na execução de uma determinada tarefa

Gerenciador de Execução do Workflow: este componente apoia a interação com os usuários do WfMS. É através dele que os usuários identificam as suas tarefas no *workflow*.

IExecutarTarefaMgt

SolicitarConexão()	Permite a conexão de um usuário
SelecionarTarefa()	Seleciona uma determinada tarefa
ExecutarTarefa()	Executa uma determinada tarefa
VisualizarTarefa()	Mostra as tarefas pertencentes a um determinado usuário de acordo com o seu cargo

Gerenciador de Tarefas: Este componente é responsável pelo controle e gerenciamento das tarefas e ações a serem realizadas no *workflow*.

IAgendarTarefaMgt

SolicitarConexão()	Permite a conexão de um usuário
AgendarTarefa()	Agenda uma determinada tarefa

IExecutarTarefaMgt

SelecionarTarefa()	Seleciona uma determinada tarefa
ExecutarTarefa()	Executa uma determinada tarefa
CancelarTarefa()	Cancela uma determinada tarefa a ser executada ou que já está em execução por um determinado usuário
InterromperTarefa()	Interrompe uma determinada tarefa
ReiniciarTarefa()	Reinicia uma determinada tarefa
FinalizarTarefa()	Finaliza uma determinada tarefa
VisualizarTarefa()	Mostra as tarefas pertencentes a um determinado usuário de acordo com o seu cargo

Gerenciador de Alocação de Recursos: Este componente é responsável pela alocação de recursos necessários (ex. alocação de atores, alocação de ferramentas, alocação de artefatos e a definição das datas de execução das tarefas) na realização dos projetos.

IAlocarRecursoMgt

SolicitarRecurso()	Solicita um determinado recurso (ex. ferramenta, ator)
LiberarRecurso()	Libera um determinado recurso para efetiva utilização
ExcluirRecurso()	Exclui um determinado recurso
RecuperarEstadoRecurso()	Recupera o estado de um determinado recurso (ex. reservado)

Gerenciador de Aplicações Externas: Este componente é responsável pelo gerenciamento das aplicações externas invocadas durante a definição e execução de um *workflow*.

IinvocarAplicaçõesExternasMgt

SelecionarAplicaçãoExterna()	Seleciona uma aplicação externa
InvocarAplicaçãoExterna()	Invoca uma aplicação externa

Gerenciador de Objetos do Workflow: Este componente é responsável pelo relacionamento com o sistema gerenciador de banco de dados.

IArmazenarObjetoMgt

InserirObjeto()	Insere um novo objeto de *workflow*
AtualizarObjeto()	Atualiza um objeto de *workflow*
ExcluirObjeto()	Exclui um objeto de *workflow*
SelecionarObjeto()	Seleciona um objeto de *workflow*

Interpretador: Interpretas as tarefas definidas para que possam ser executadas.

IExecutarScriptMgt

ExecutarScript()	Executa *Script*
InvocarAplicaçãoExterna()	Invoca uma aplicação externa

2.3.2 Especificação da Arquitetura com CORBA

Segundo o processo seguido pelo Catalysis, após a modelagem da chamada arquitetura lógica (Figura 8), deve-se tomar decisões sobre os mecanismos de implementação a serem utilizados, como por exemplo, o *middleware*. Estas decisões levam ao projeto da arquitetura técnica, conforme apresentada na Figura 9. Para a arquitetura de linha de produtos proposta foi considerado o ORB (*Object Request Broker*) do CORBA [16] como *middleware*, que ocupa o centro da figura. Os componentes são estruturados em camadas de modo a minimizar as importações.

Os componentes da arquitetura possuem interfaces padronizadas e bem definidas através da linguagem IDL (*Interface Definition Language*) do CORBA. Isso faz com que os componentes possam ser acessados dentro da arquitetura independentemente da linguagem de programação, sistema operacional e rede utilizados.

As comunicações entre os componentes são feitas através do ORB e o componente requisitado atende ou não à requisição, baseando-se em restrições de comunicação que são definidas para cada componente da arquitetura. Por exemplo, o componente *Gerenciador de Interface Gráfica* comunica-se apenas com os componentes *Gerenciador de Arquitetura de Workflow*, *Gerenciador de Workflow* e *Gerenciador de Execução do Workflow*. Qualquer solicitação de comunicação de outro componente da arquitetura com o componente *Gerenciador de Interface Gráfica* será

recusada porque existem restrições de comunicação definidas no componente requisitado.

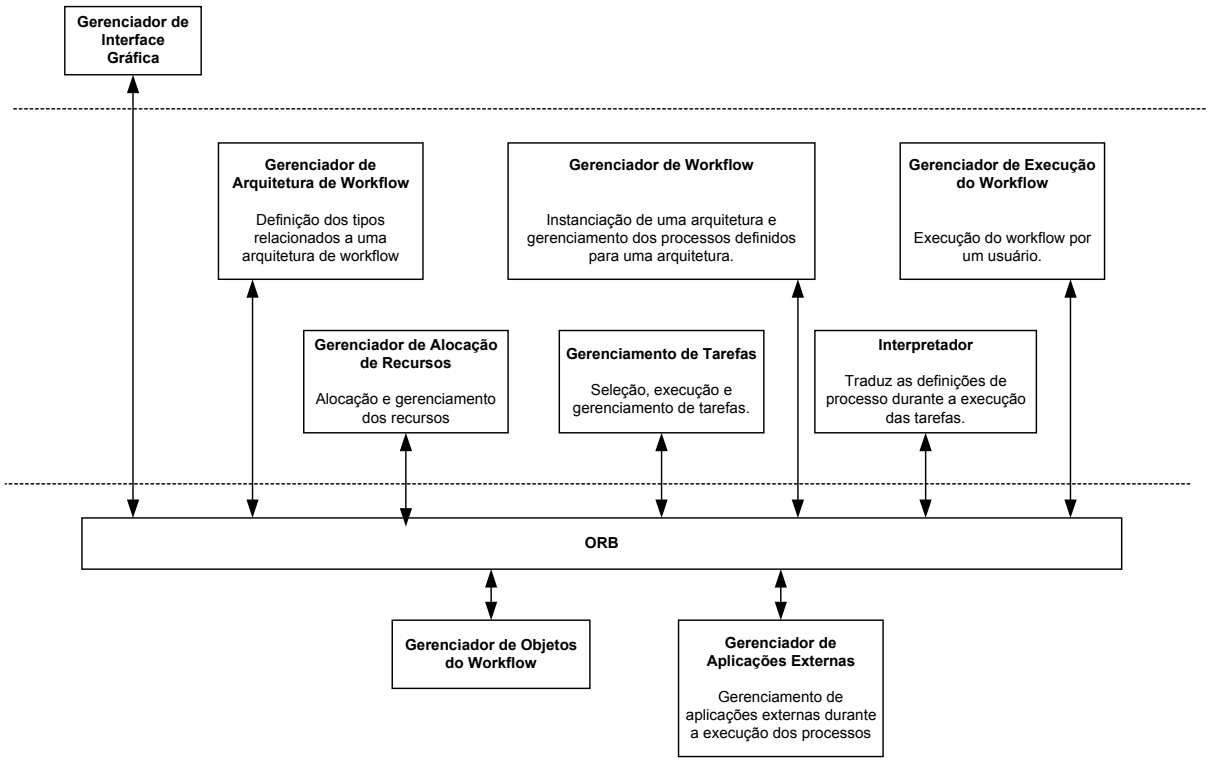

Figura 9 – Arquitetura de Linha de Produtos para WfMS proposta.

Uma descrição mais detalhada de cada um dos componentes da arquitetura, incluindo aspectos de variabilidade, é apresentada a seguir:

Gerenciador de Interface Gráfica: este componente é responsável pelo gerenciamento da interface com o usuário do sistema. O ponto de variação deste componente está no fato de que a interface com o usuário pode ocorrer via interface gráfica convencional ou via *browser*.

Gerenciador de Arquitetura de Workflow: este componente é 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 flexível, visto que desta forma, os tipos de objetos não são pré-fixados. Os pontos de variação associados a este componente indicam o tipo de recurso que pode ser utilizado. Este tipo recurso pode ser especializado em ator, ferramenta e material e o tipo de ferramenta pode ser especializado em interna e externa. Diferentes linguagens de programação de processos podem ser apoiadas para a definição das tarefas, que posteriormente serão executadas pelo interpretador.

Gerenciador de Workflow: este componente é 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 executadas e gerenciadas, fazendo-se a alocação de recursos e demais tomadas de decisão. Não existem pontos de variação associados a este componente.

Gerenciador de Execução do Workflow: este componente é responsável pelo controle e gerenciamento 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*. Não existem pontos de variação associados a este componente.

Gerenciador de Tarefas: este componente é responsável pelo controle e gerenciamento das tarefas e ações a serem realizadas. É por meio deste os usuários interagem com o sistema gerenciador de *workflow*. O Gerenciador de Tarefas permite que os usuários identifiquem suas tarefas geradas pelo Gerenciador de *Workflow*. Os pontos de variação deste componente são: os tipos de recursos que podem ser utilizados podem ser material, ferramenta e ator; a ferramenta utilizada pode ser do tipo interna ou externa; as tarefas podem ter prioridades de execução diferentes e, os algoritmos utilizados para escalonamento das tarefas também podem variar.

Gerenciador de Alocação de Recursos: este componente é responsável pela alocação de recursos (atores, ferramentas ou material). Além da variabilidade associada ao tipo de recurso e ao tipo de ferramenta, as políticas de alocação de recurso podem variar.

Gerenciador de Aplicações Externas: este componente é responsável pelo gerenciamento das aplicações externas invocadas durante a definição do processo e execução das tarefas. Pontos de variabilidade incluem as diferentes formas de adaptação da aplicação externa ao sistema gerenciador de *workflows*.

Gerenciador de Objetos do Workflow: este componente é responsável pelo relacionamento com os mecanismos de armazenamento de objetos manipulados pelo sistema. Suas funções trabalham com objetos, que podem ser considerados desde dados de controle do processo, dados relevantes do processo ou até mesmo instâncias de *workflow* e tarefas. Todos os componentes pertencentes à arquitetura de componentes proposta utilizam seus serviços. 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. Pontos de variação deste componente incluem os adaptadores para os gerenciadores de bancos de dados.

Interpretador: uma linguagem para programação de processos é necessária para que o processo possa ser programado e executado pelo gerenciador de *workflow*. As tarefas que compõem o *workflow* são então programadas utilizando uma linguagem de programação de processos e recursos que permitem a execução cooperativa dessas tarefas. Para a execução dessas tarefas, o gerenciador de tarefas realiza chamadas ao interpretador da linguagem para que este possa executá-las.

Nesta fase do processo, os componentes da arquitetura estão especificados com suas interfaces definidas. Segundo o processo seguido para este trabalho, a etapa seguinte consiste no projeto interno dos componentes da arquitetura. Porém, a

implementação desses componentes está fora do contexto deste trabalho. Dentre os trabalhos relacionados a este que está sendo desenvolvido, pode-se destacar a implementação do componente agenda de tarefas por Tanaka [17] e a proposta de um *enterprise framework* para WfMS proposto por Gimenes [18].

3. Validação da Arquitetura Proposta

Bosch [19] identificou quatro formas de validação de uma arquitetura de linha de produtos: uso de cenários, simulação, modelos matemáticos e avaliação baseada em experiências passadas. A seção 2 descreveu o processo de obtenção da arquitetura de linha de produtos para sistemas de gerenciamento de *workflow*. Este processo é baseado no método Catalysis [7], que utiliza a UML como notação. Assim, como na UML [10] não existem recursos para simulação da arquitetura e para que pudéssemos avaliar a arquitetura proposta sem nos aprofundar em detalhes de implementação optamos por utilizar uma ADL (*Architecture Description Language*) para especificar e simular a arquitetura. ADLs [20] são linguagens focadas em representar estruturas de mais alto nível, ao invés de se ater a detalhes de implementação. A ADL Rapide [8] foi identificada como adequada para especificar a arquitetura de linha de produtos para sistemas gerenciadores de *workflow*. Esta linguagem foi escolhida por ser caracterizada como uma linguagem de descrição arquitetural com finalidade geral, que permite modelar interfaces de componentes e seus comportamentos externamente visíveis, e ainda por ser uma linguagem que permite simulação e modelagem do comportamento dinâmico descrito por uma arquitetura.

A simulação foi baseada na seleção de cenários relevantes para os sistemas de gerenciamento de *workflow*. Foram elaborados diagramas de seqüência para representar as interações para cenários específicos de uma certa atividade do sistema, que especificam a comunicação entre os componentes. Esses diagramas serviram de base para realizar a simulação da arquitetura de linha de produto proposta. Dessa maneira, a arquitetura é executada conforme diferentes cenários para simular o comportamento do sistema. Cada cenário representou uma das visões dos diferentes usuários do sistema. Esses usuários são: o gerente de arquitetura de *workflow*, que define a arquitetura de *workflow*; o gerente de *workflow*, responsável pelas atividades de supervisão e administração e, por fim, o usuário que executa as tarefas do *workflow*. Além disso, a modelagem do comportamento dinâmico descrito pela arquitetura de componentes proposta, possibilitou avaliar a comunicação entre os componentes e o efeito geral das funcionalidades do sistema, livrando os detalhes de implementação para a fase de aplicação da arquitetura.

Porém, a simulação não foi suficiente para validar completamente o modelo proposto. A validação completa ainda depende do estabelecimento de critérios apropriados que permitam quantificar o comportamento do sistema real. A chave para a validação baseada em simulação é criar testes que exercitem um erro, de modo a torná-lo aparente para o usuário. Atualmente, testes são criados manualmente e direcionados para casos de teste particulares ou são gerados de modo aleatório na esperança de alcançar situações não testadas de modo probabilístico. Ambos os métodos falham em fornecer uma medida confiável de que um sistema complexo tenha sido testado de modo adequado. Portanto, a simulação, por sua vez, não faz com que a atividade de teste seja facilmente avaliada e quantificada. Dessa maneira, uma outra forma de se validar modelos arquiteturais é através da aplicação do critério Análise de Mutantes, da técnica de teste baseada em erros, que permite avaliar a atividade de teste de forma quantitativa, através do escore de mutação.

4. Trabalhos Relacionados

Não existe um entendimento comum da relação entre reuso, domínios, linhas de produtos, arquitetura de software e *frameworks* [21]. Acredita-se que a controvérsia surge no nível de abstração que cada um desses conceitos representa. O termo linha de produtos parece mais adequado para o contexto industrial. Os conceitos de engenharia de domínio são amplamente utilizados no enfoque de linha de produto.

Os métodos conhecidos de análise de domínio [22], [6], [23], [14] buscam identificar conceitos e funcionalidades requeridas para uma família de produtos e representa-los através de um modelo genérico, que é então refinado pelos estágios subseqüentes de construção de uma infraestrutura de reutilização. Esses métodos utilizam o conceito de *features* para representar as funcionalidades comuns do domínio e também para representar aspectos de variabilidade. Catalysis, apesar de não ser um método de engenharia de domínio, apresenta o conceito de tipo (análogo ao conceito de *feature*), que seria descrição de alguma funcionalidade da aplicação em termos de seu comportamento externo.

A abordagem de linha de produtos está também diretamente relacionada com as técnicas de desenvolvimento de *frameworks*. Gimenes et al. [17] [18] propõe técnicas de definição de *frameworks* no contexto de WfMS que ofereceram *guidelines* para a construção da arquitetura proposta neste trabalho. Essas técnicas envolvem a utilização do conceito de *framework* de modelos do Catalysis como base para geração de componentes.

5. Conclusões

Este artigo apresenta o processo seguido para o projeto de uma arquitetura de linha de produtos para sistemas de gerenciamento de *workflow*. As extensões utilizadas para representar variabilidade durante o processo de desenvolvimento foram aplicadas com base nos diagramas de caso de uso propostos por Jacobson [14] e extensão da notação UML proposta por Morisio [15].

O método Catalysis [7] foi utilizado para guiar o processo de extração e formalização da arquitetura de componentes dos WfMS, uma vez que os documentos da WfMC [2] especificam a arquitetura de um WfMS por meio de diagramas informais. Porém, para avaliação da arquitetura proposta optou-se por utilizar a ADL [8] Rapide que possui um conjunto de ferramentas de suporte a simulação de arquiteturas. A escolha desta linguagem permitiu a simulação sem que houvesse necessidade de aprofundamento em detalhes de implementação.

A principal contribuição deste trabalho é a arquitetura de linha de produtos para sistemas de gerenciamento de *workflow*. Pode-se destacar também contribuições para a formalização de um processo de desenvolvimento de arquiteturas de linhas de produtos e também um melhor entendimento dos conceitos e abordagens relacionados à prática de linha de produtos.

Para que se possa gerar produtos a partir da arquitetura proposta, deve-se selecionar os componentes da arquitetura que cumprem os requisitos do produto específico, bem como instanciar as variabilidades associadas aos componentes da arquitetura.

5. Referências

[1] Bass, Len et al. "*Software Architecture in Practice"*. Addison Wesley Longman, 1998. 452 p.

[2] Workflow Management Coalition. "*Workflow Reference Model*". Document number TC00-1003, January 19, 1995. 55 p.

[3] Software Productivity Consortium. "*Reuse-Driven Software Processes Guidebook*", SPC-92019-CMC version 02.00.03 November 1993.

[4] Weiss, D. M. and Chi Tau Robert Lai. "*Software Product-Line Engineering: A Family-Based Software Development Approach*". Addison-Wesley, 1999.

[5] Bayer, J., O. Flege, P. Knauber, et al., "*PuLSE: A methodology to develop software product lines*", Symposium on Software Reusability (SSR99), May 1999.

[6] Kang, K., et al. "*Feature-Oriented Domain Analysis (FODA) Feasibility Study (CMU/SEI-90-TR-21, ADA 235785)*". Pittsburgh, PA: SEI CMU, 1990.

[7] D'Souza, D. F. et al. "*Objects, Components and Frameworks with UML – The Catalysis Approach*". Addison Wesley Publishing Company, 1999.

[8] Computer Science Lab. "*DRAFT Guide to Rapide 1.0 – Language Reference Manuals*", Rapide Desigh Team – Program Analysis and Verification Group. Stanford University, 1997.

[9] Luckham, J. J. et al. "*Specification and Analysis of System Architecture Using* Rapide". IEEE Transactions on Software Engineering, Special Issue on Software Architecture, vol 21, nº4, Abril 1995.pag. 336 a 355.

[10] Rumbaugh, J. "*The Unified Modeling Language Reference Manual*", Addison-Wesley Pub. Company, 1999.

[11] Atkinson, C. et al. "*Component-Based Product Line Development: The Kobra Approach*". In: 1st INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 2000, Pittsburgh. Proceedings… Pittsburgh: [s.n], 2000.

[12] Batory, D. "*Product Line Architectures*". Erfurt, Germany. Trabalho apresentado em Smaltalk and Java in Industrie and Ausbildung. 1998.

[13] Gimenes, I.M. S. et al., "*Um Padrão para Definição de um Gerenciador de Processos de Software*", In: II Workshop Ibero Americano de Engenharia de Requisitos Y Ambientes de Software, San José, Costa Rica, Ideas'1999 Memorias, 1999, San José: Istituto Tecnológico de Costa Rica, 1999, v.1, pp. 30-46.

[14] Jacobson, I. et al. "*Software Reuse – Architecture Process and Organization for Business Success*". New York: Ed. Addison-Wesley, 1997.

[15] Morisio, M., Travassos, G.H., Stark, M. "*Extending UML to Support Domain* Analysis", IEEE International Conference on Automated Software Engineering – ASE'00. Grenoble, France, 2000.

[16] Object Management Group. "*COM versus CORBA: A Decision Framework*". Available in http://www.quoininc.com/COM_CORBA.html . Last access dez/2000.

[17] Gimenes, I. M. S., Tanaka, S. e Oliveira, J. P. M., "*An Object Oriented Framework for Task Scheduling*", In: *TOOLS Europe 2000*, 2000, Mont St. Michel, France, Tools 33 Technology of Object-oriented Languages and Systems, USA: IEEE Computer Society Press, 2000, v.1, pp. 383-394.

[18] Gimenes, I. M. S. et al. "*Enterprise Frameworks for Workflow Management Systems*". Software Practice & Experience, 2001, a ser publicado.

[19] Bosch, J., "*Design & Use Of Software Architectures. Adopting and Evolving a Product-Line Approach*", Addison-Wesley, 2000.

[20] Medividovic, N. et al. "*A Classification and Comparison Framework for Software Architecture Description Languages*", IEEE Transactions on Software Engineering, v. 26, jan. 2000.pp 70-92.

[21] Poulin, J. "*Software Architectures, Product Lines, and DSSAs: Choosing the Appropriate Level of Abstraction*", WISR8, 1997.

[22] Kang, K., et al. "*FORM: A Feature-Oriented Reuse Method with Domain-Specific Refrence Architecture*", SEI Technical Report, 1998.

[23] Griss, M., et al. "*Integrating Feature Modeling with RSEB*", 5th International Conference on Software Reuse (ICSR-5), ACM/IEEE, Victoria, Canadá, Junho 1998.

Bibliografia

[ACM 2001] ACME. **THE ACME ARCHITECTURAL DESCRIPTION LANGUAGE**. USA: School of Computer Science, Carnegie-Mellon University. Disponível em: <http://www.cs.cmu.edu/~acme/>. Acesso em: 05 de fev. 2001.

[ALL 98] ALLEN, P et al. **Component-Based Development for Enterprise Systems, Applying the Select**. [S.l.]: Cambridge University Press, 1998.

[ATK 2000] ATKINSON, C. et al. Component-Based Product Line Development: The Kobra Approach. In: INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 1., 2000, Pittsburgh. **Proceedings…** Pittsburgh: [s.n.], 2000.

[BAS 97] BASS, Len et al. **Product Line Practice Workshop Report**. Pittsburgh : [s.n.], 1997. 36 p. (Tecnhical Report CMU/SEI-97).

[BAS 98] BASS, Len et al. **Software Architecture in Practice**. [S.l.]: Addison Wesley Longman, 1998. 452 p.

[BAT 98] BATORY, D. **Product Line Architectures**. Erfurt, Germany. Trabalho apresentado em Smaltalk and Java in Industrie and Ausbildung. 1998.

[BAY 99] BAYER, J. et al. PuLSE: A methodology to develop software product lines. In: SYMPOSIUM ON SOFTWARE REUSABILITY, SSR, 1999, Los Angeles. **Proceedings…** Los Angeles: [S.l.], 1999.

[BOO 99] BOOCH, G. et al. **The Unified Modeling Language Users Guide**. [S.l.]: Addison-Wesley Publishing Company, 1999.

[BOS 2000] BOSCH, Jan. **Design & Use of Software Architectures:** adopting and evolving a product-line approach. Great Britain: Addison Wesley, 2000. 354 p.

[BRO 96] BROWN, T. **Process of Validation**. Disponível em: <http://beaver.nrri.umn.edu/~tbrown/papers/paper3/node1.html>. Acesso em: jun. 2002.

[BRO 99] BROWN, A.; BARN, B. Enterprise Scale CBD: Building Complex Computer Systems From Components. In: STEP, 1999. **Proceedings...** [S.l.]: IEEE Computer Society Press, 1999. p. 82-93.

[CAJ 2001] CASTRO, J. From Requirements to Architectures. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ICSE, 2001, Toronto. **Proceedings…** Toronto: [s.n.], 2001.

[CAR 99] CARVALHO, R. A. **Uma Avaliação da Aplicação do Critério Análise de Mutantes na Validação de Especificações e Máquinas de Estados Finitos**. 1999. Dissertação (Mestrado em Ciência da Computação) - Departamento de Computação, Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de São Carlos, São Carlos.

[CAS 95] CASATI F. et al. Conceptual Modeling of Workflows. In: OBJECT-ORIENTED AND ENTITY-RELATIONSHIP, 1995, Gold Coast, Austrália. **Proceedings...** Gold Coast: [s.n.], 1995.

[CAS 96] CASATI, F. et al. **WIDE *Workflow* model and architecture**. 1996. Disponível em: <http://dis.sema.es/projects/WIDE/Documents>. Acesso em: 29 jan. 2001.

[CLE 2001] CLEMENTS, P. et al. **Software Product Lines – Practices and Patterns**. [S.l.]: Addison Wesley, 2001. 448 p.

[CSL 97] COMPUTER SCIENCE LAB. Stanford University. **DRAFT Guide to Rapide 1.0, Language Reference Manuals**, Stanford, July 1997. 80p.

[CZA 99] CZARNECKI, K. et al. Components and Gerenative Programming. In: EUROPEAN SOFTWARE ENGINEERING CONFERENCE, ESEC, 7., 1999, Tolouse. **Proceedings…** Tolouse, France: [s.n.], 1999.

[D'SO 99] D'SOUZA, D.; WILLS, A. **Objects, Components and Frameworks with UML – The Catalysis Approach**. [S.l.]:Addison-Wesley, 1999. 816 p.

[FAB 94] FABBRI, S.C.P.F. et al. Mutation Analysis Testing for Finite State Machines. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING, 5., 1994, Califórnia, USA. **Proceedings…** California: [s.n.], 1994, p. 220-229.

[FAB 95] FABBRI, S.C.P.F. et al. Mutation Testing Applied to Validate Specifications Based on Petri Nets. In: INTERNATIONAL IFIP CONFERENCE ON FORMAL DESCRIPTION TECHNIQUES FOR DISTRIBUTED SYSTEMS AND COMMUNICATIONS PROTOCOLS, FORTE, 8., 1995, Montreal. **Proceedings…** Montreal: [s.n.], 1995. p. 329-337.

[FLA 99] FLACH, C. V. **What is an Architecture Description Language?** Rio de Janeiro: DCC da PUC- RIO, 1999. 22p. Disponível em: <http://www.inf.puc-rio.br/~flach/ativ982.html>. Acesso em: 20 jan. 2001).

[FOW 97] FOWLER, M. et al. **UML Distiled**. Reading: Addison-Wesley, 1997.

[GAM 95] GAMMA, E. et al. **Design Patterns:** elements of reusable object-oriented software. Massachusetts: Addison-Wesley, 1995.

[GEO 95] GEORGAKOPOULOS, D. et al. An Overview of *Workflow* Management: From Process Modeling to *Workflow* Automation Infrastructure. **ACM Distributed and Parallel Database**, [S.l.], n.3, p. 119-153, 1995.

[GIM 2000] GIMENES, I.M.S et al. O Processo de Desenvolvimento Baseado em Componentes Através de Exemplos. In: ESCOLA DE INFORMÁTICA DA SBC-SUL, 8., 2000, Foz do Iguaçu. **Livro Texto**. Santa Maria: UFSM, 2000.

[GIM 2001] GIMENES, I. M. S. et al. Enterprise Frameworks for *Workflow* Management Systems. **Software Practice & Experience**, [S.l.], n.32, p.755-769, 2001.

[GIM 2002] 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.

[GIM 99a] GIMENES, I. M. S. et al. Um Padrão para Definição de um Gerenciador de Processos de Software. In: WORKSHOP IBERO AMERICANO DE ENGENHARIA DE REQUISITOS Y AMBIENTES DE SOFTWARE, 2., 1999, San José. **Proceedings…** San José: Instituto Tecnológico de Costa Rica, 1999. v.1, p. 30-46.

[GIM 99b] GIMENES, I. M. S. et al. **ExPSEE – An Experimental Process Centred Software Engineering Environment**. Maringá: UEM/CTC/DIN, 1999. Relatório Técnico.

[GRI 98] GRISS, M. et al. Integrating Feature Modeling with RSEB. In: INTERNATIONAL CONFERENCE ON SOFTWARE REUSE, ICSR-5, 5. 1998, Victoria**. Proceedings...** Victoria: ACM/IEEE, 1998.

[JAC 97] JACOBSON, I. et al. **Software Reuse – Architecture Process and Organization for Business Success**. New York: Addison-Wesley, 1997. 497 p.

[JAC 99] JACOBSON, I. et al. **Rational Unified Software Development Process**. [S.l.]: Addison Wesley, 1999.

[JDC 2000] JAVA DEVELOPER CONNECTION**. Package java.rmi Description.** Disponível em: <http://software.oit.pdx.edu/java/docs/api/java/rmi/package-summary.html>. Acesso em: 10 dez. 2000.

[KAN 90] KANG, K. et al. **Feature-Oriented Domain Analysis (FODA) Feasibility Study (CMU/SEI-90-TR-21, ADA 235785)**. Pittsburgh, PA: SEI CMU, 1990. (Relatório Técnico, n. 21).

[KAS 2000] KASPRZAK, G. **Uma Proposta de Arquitetura de Componentes para Sistemas Gerenciadores de *Workflow* Baseada em Catalysis**. 2000. 64 p. Trabalho de Graduação (Bacharelado em Ciência da Computação) – Departamento de Informática, Universidade Estadual de Maringá, Maringá.

[LAZ 2001] LAZILHA, F.R. **Fundamentos para uma Proposta de Arquitetura de Linha de Produção para *Workflow* Management Systems**. 2001. 63 p. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[LAZ 2002] LAZILHA, F. R.; GIMENES, I. M. S.; PRICE, R. T. **Uma Proposta de Arquitetura de Linha de produto para *Workflow Management Systems***. Trabalho apresentado no 16. Simpósio Brasileiro de Engenharia de Software, 2002, Gramado. No prelo.

[LUC 95] LUCKHAM, John J. et al. Specification and Analysis of System Architecture Using Rapide. **IEEE Transactions on Software Engineering**, New York, v.21, n.4, p. 336-255, Apr. 1995.

[MAL 98] MALDONADO, J.C. et al. **Validação de Especificações Formais com a Aplicação do Critério Análise de Mutantes.** Trabalho apresentado no 1. WMF – Workshop de Métodos Formais, 1998, Porto Alegre.

[MED 2000] MEDVIDOVIC, N et al. A Classification and Comparison Framework for Software Architecture Description Languages. **IEEE Transactions on**

Software Engineering, New York, v.26, n.1, p.70-92, Jan. 2000.

[MIC 2000] MICROSOFT. D**istributed Component Object Model Protocol - DCOM/1.0.** Disponível em: <http://msdn.microsoft.com/library/specs/distributedcomponentobjectmodelprotocoldcom10.htm>. Acesso em: 10 dez. 2000.

[MOR 2000] MORISIO, M. et al. Extending UML to Support Domain Analysis, In: IEEE INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING, ASE, 2000, Grenoble. **Proceedings…** Grenoble: IEEE, 2000.

[OMG 2000] OBJECT MANAGEMENT GROUP. **COM versus CORBA: a decision framework**. Disponível em: <http://www.quoininc.com/COM_CORBA.html>. Acesso em: 10 dez. 2000.

[OMG 2002] OBJECT MANAGEMENT GROUP. **Model Drive Architecture**. Disponível em: <http://www.omg.org/mda/>. Acesso em : 12 maio 2002.

[POU 97] POULIN, J. Software Architectures, Product Lines, and DSSAs: Choosing the Appropriate Level of Abstraction. In: INTERNATIONAL WORKSHOP ON SOFTWARE REUSE, 8., WISR, 1997, Columbus. **Proceedings…** Columbus: [s.n.], 1997.

[RUM 99] RUMBAUGH, J. et al. **The Unified Language Reference Manual**. [S.l.]: Addison-Wesley, 1999.

[SHA 95] SHAW, M. et al. **Abstractions for Software Architecture and Tools to Support Them.** [S.l.]: Carnegie-Mellon University, 1995.

[SHA 96] SHAW, M.; GARLAN, D. **Software Architecture:** perspectives on an emergency discipline. New York: Prentice-Hall, 1996. 241 p.

[SIL 99] SILVA, R. P. **Suporte ao desenvolvimento e uso de *frameworks* e componentes**. 1999. 262p. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[SPC 93] SOFTWARE PRODUCTIVITY CONSORTIUM. **Reuse-Driven Software Processes Guidebook**. SPC-92019-CMC version 02.00.03 November 1993.

[STA 2001] STANFORD RESEARCH INSTITUTE. **A Strucutural Architecture Description Language**. Disponível em: <http://www.sdl.sri.com/dsa/sadl-main.html>. Acesso em: 05 fev. 2001.

[STE 2000] STERLING SOFTWARE. **The Sterling Software Component Standard**, Version 3.0. Disponível em: <http://www.sterling.com/cool/>. Acesso em: 05 nov. 2000.

[TAK 2002] TAKANO, E. T. **Um Ambiente Experimental de Engenharia de Software Orientado a Processos.** Maringá, PR: UEM-DIN, 2002. 56 p. Relatório Técnico.

[TAN 2000a] TANAKA, S. **Um *framework* de Agenda de Tarefas para Gerenciadores de Processos**. 2000. 124p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal

do Rio Grande do Sul, Porto Alegre.

[TAN 2000b] TANAKA, S. A. et al. An Object Oriented Framework for Task Scheduling, In: TOOLS EUROPE, 33., 2000, Mont St. Michel. **Proceedings…** Mont St. Michel: IEEE Computer Society Press, 2000. v.1, p. 383-394.

[TRA 2002] TRAVASSOS, G.H. et al. **TABA Workstation:** towards a product line for the Developing of Software Engineering Environments. Rio de Janeiro: COPPE/UFRJ, 2002. (Relatório Técnico do Projeto TABA, RT-21/02).

[UKU 98] UKUMA, L. H. **Uma Experiência na Utilização de UML na Especificação de Software**. 1998. 75p. Trabalho de Graduação (Bacharelado em Ciência da Computação) – Universidade Estadual de Maringá, Maringá.

[ULT 97] ULTIMUS LLC. **Workflow for the Information Age**. 1997. Disponível em: <http://www.workflowzone.com>. Acesso em: 30 jan. 2001.

[WAR 99] WARMER, J.; KLEPPE, A. **The Object Constraint Language – Precise Modeling with UML.** [S.l.]: Addison Wesley Longman, 1999.

[WEG 98] WEISS, G. M. **Um Padrão para Definição de um Gerenciador de Processos de Software.** 1998. 68p. Trabalho de Graduação (Bacharelado em Ciência da Computação) – Universidade Estadual de Maringá, Maringá.

[WEI 99] WEISS, D. et al. **Software Product-Line Engineering: a family-based software development process**. [S.l.]: Addison Wesley, 1999. 426 p.

[WER 2000] WERNER, C. M. L.; BRAGA, R. M. M. Desenvolvimento Baseado em Componentes. In: SBES WORKSHOPS, 14., 2000, João Pessoa. **Anais...** João Pessoa: CEFET – PB, 2000. p. 297-329.

[WMC 95] WORKFLOW MANAGEMENT COALITION. ***Workflow*** **Reference Model**. Bruxelas, Jan. 1995. 55p. (Document number TC00-1003).

[WMC 99] WORKFLOW MANAGEMENT COALITION. **Terminology & Glossary**. Bruxelas, June 1999. 52p.