

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

**Uma Arquitetura Aberta para a Integração de
Sistemas de Gerência de Documentos e
Sistemas de Gerência de *Workflow***

por

VINÍCIUS LEOPOLDINO DO AMARAL

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

Prof. Dr. José Valdeni de Lima
Orientador

Porto Alegre, janeiro de 1999.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Amaral, Vinícius Leopoldino

Uma arquitetura aberta para a integração de sistemas de gerência de documentos e sistemas de gerência de workflow / por Vinícius Leopoldino do Amaral. - Porto Alegre: CPGCC da UFRGS, 1999.

108 f.: il.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Sul. Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, BR-RS, 1998. Orientador: Lima, José Valdeni de.

1. Modelagem de workflow. 2. Gerência de documentos. 3. Gerência de workflow. 4. Arquiteturas abertas. 5. Trabalho Cooperativo. I. Lima, José Valdeni de. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Reitora: Profa. Wrana Panizzi

Pró-Reitor de Pós-Graduação: Prof. José Carlos Ferraz Hennemann

Diretor do Instituto de Informática: Prof. Philippe Oliver Alexandre Navaux

Coordenadora do CPGCC: Profa. Carla Maria Dal Sasso Freitas

Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

Agradecimentos

Ao meu orientador, prof. Valdeni, pelo apoio constante durante a escrita desta dissertação, pelas revisões atentas deste trabalho e pela amizade desenvolvida.

Ao amigo Marco Aurélio Mangan, pelas inúmeras conversas e idéias a respeito deste trabalho (e de outras coisas também). Ao amigo Marco Antônio Winckler, pelas suas pacientes e detalhadas revisões de várias versões deste e de outros trabalhos. Aos amigos Marilene Noronha (*Marie*) e Anderson Grala, pelo companheirismo e pela parceria no projeto GDOC.

Ao *monsieur* Palazzo, meu guru para a concepção deste trabalho de mestrado. Foram em suas aulas, mais do que em qualquer outro lugar, que me senti verdadeiramente um aluno de pós-graduação. E foi através de seu exemplo de verdadeiro mestre que compreendi plenamente o papel de um professor, como aquele que não apenas transmite conhecimentos, mas como aquele que instiga, semeia e faz florescer em seus alunos o desejo de explorar, desafiar e ultrapassar os limites do saber estabelecido.

Ao Chuck Fay, da FileNet, e ao Dennis Hamilton, da Xerox, pelo apoio constante na compreensão e utilização do padrão DMA. Ao Luciano Wilson de Medeiros, da FileNet Brasil, pelo software cedido. À Hella Philippe, do Secretariado da WfMC, ao Klaus-Dieter Kreplin, da Empirica e ao Marc-Thomas Schmidt, da IBM, pela colaboração na obtenção das especificações WAPI e na sua compreensão. À Ronni Marshak pela colaboração na compreensão do estado atual dos mercados de sistemas de gerência de documentos e de sistemas de gerência de documentos e na compreensão do impacto deste trabalho nesse mercado.

Ao prof. Cirano Iochpe, pela oportunidade de participar no desenvolvimento de um sistema real de workflow, o que trouxe valiosos benefícios para este trabalho. Ao pessoal da FEPAM pela colaboração durante o projeto, em especial à Sra. Mariza Wagner.

A Deus, simplesmente por existir.

Sumário

Lista de Abreviaturas	6
Lista de Figuras	7
Lista de Tabelas	9
Resumo	10
Abstract	11
1 Introdução	12
1.1 Necessidades de Integração	13
1.2 Objetivos	14
1.3 Estrutura do Trabalho	16
2 Formas Atuais de Integração	17
2.1 DMS Estendidos para <i>Workflow</i>	17
2.1.1 Vantagens	19
2.1.2 Desvantagens	19
2.2 Integração Proprietária de DMS e WFMS	20
2.2.1 Vantagens	21
2.2.2 Desvantagens	22
2.3 Integração <i>Ad hoc</i> de DMS e WFMS	22
2.3.1 Vantagens	23
2.3.2 Desvantagens	24
2.4 Conclusões	25
3 Padrões para Sistemas de Gerência de <i>Workflow</i>	26
3.1 Motivações	26
3.2 Modelo de Referência	27
3.3 Padrões para Definição de Processos	28
3.3.1 Meta-modelo	29
3.3.2 WPDL	30
3.4 Padrões para Acesso aos Serviços de <i>Workflow</i>	32
4 Padrões para Sistemas de Gerência de Documentos	35
4.1 Motivações e Objetivos	35
4.2 Conceitos Fundamentais	36
4.3 Arquitetura DMA	38
4.4 Modelo de Integração	39

4.5	Modelo de Conteúdo	40
4.6	Modelo de Versionamento	41
5	Uma Proposta de Arquitetura Aberta	44
5.1	Visão	44
5.2	Estrutura da Solução	45
5.2.1	Modelo integrado de documentos e <i>workflow</i>	45
5.2.2	Integração de serviços	46
5.2.3	Integração dos aplicativos cliente	47
6	Aspectos de Modelagem	48
6.1	Integração Conceitual	48
6.1.1	Classes de <i>Workflow</i>	50
6.1.2	Classes <i>Document</i> e <i>Version</i>	50
6.1.3	Classe <i>Doc Type</i>	50
6.1.4	Classes <i>Activity on Documents</i> e <i>Allowed Operations</i>	51
6.1.5	Conceito de ciclo de vida de um documento	51
6.2	Mapeamento para os padrões DMA e WAPI	51
6.2.1	A WPDL-DOC	53
6.2.2	Dinâmica do mapeamento	60
6.3	Exemplo de mapeamento	61
6.3.1	Definição do processo exemplo	61
6.3.2	Mapeamento para a WPDL-DOC e DMA	64
7	Descrição do Protótipo	69
7.1	Detalhamento da Arquitetura	69
7.2	A Camada de Serviços	71
7.2.1	Diagrama de classes	72
7.2.2	Implementação dos serviços	74
7.3	Aplicativo Cliente	91
7.3.1	Descrição da interface	91
7.3.2	Operação do aplicativo	93
8	Conclusões	97
8.1	Contribuições	97
8.2	Cenários de Aplicação	99
8.3	Trabalhos Futuros	100
	Bibliografia	102

Lista de Abreviaturas

AIIM	<i>Association for Information & Image Management</i>
API	<i>Application Programming Interface</i>
ASCII	<i>American Standard Code for Information Interchange</i>
BPR	<i>Business Process Reengineering</i>
COM	<i>Component Object Model</i>
CORBA	<i>Common Object Request Broker Architecture</i>
DCOM	<i>Distributed Component Object Model</i>
DMA	<i>Document Management Alliance</i>
DMS	<i>Document Management System</i>
HTML	<i>Hypertext Markup Language</i>
MIME	<i>Multipurpose Internet Mail Extensions</i>
ODA	<i>Office Document Architecture</i>
ODMA	<i>Open Document Management API</i>
OIID	<i>Object Instance Identifier</i>
OLE	<i>Object Linking and Embedding</i>
PDF	<i>Portable Document Format</i>
RMI	<i>Remote Method Invocation</i>
RPC	<i>Remote Procedure Call</i>
SPI	<i>Server Programming Interface</i>
UML	<i>Unified Modeling Language</i>
URL	<i>Uniform Resource Locator</i>
WAPI	<i>Workflow API & Interchange Formats</i>
WfMC	<i>Workflow Management Coalition</i>
WFMS	<i>Workflow Management System</i>
WPDL	<i>Workflow Process Definition Language</i>

Lista de Figuras

FIGURA 2.1 - Arquitetura dos DMS estendidos para <i>workflow</i>	Erro! Indicador não definido.
FIGURA 2.2 - Arquitetura de integração proprietária entre DMS e WFMS	21
FIGURA 2.3 - Arquitetura de integração <i>ad hoc</i> entre DMS e WFMS	23
FIGURA 3.1 - Modelo de referência de <i>workflow</i> - Componentes e Interfaces	27
FIGURA 3.2 - Meta-modelo proposto pela WfMC	30
FIGURA 3.3 - Processo de exportação/importação de <i>scripts</i> WPDL	32
FIGURA 4.1 - Modelo conceitual da DMA	37
FIGURA 4.2 - Arquitetura DMA	38
FIGURA 4.3 - Estrutura geral de um OIID	40
FIGURA 4.4 - Exemplo de modelo de documento segundo o padrão DMA	41
FIGURA 4.5 - Modelo de versões DMA	43
FIGURA 5.1 - Integração entre os serviços	46
FIGURA 6.1 - Modelo conceitual integrado de documentos e <i>workflow</i>	49
FIGURA 6.2 - Visão geral do mapeamento	52
FIGURA 6.3 - Extensão WPDL-DOC para a classe <i>Doc Type</i>	55
FIGURA 6.4 - Exemplo de dados para a extensão <i>DocTypeList</i>	55
FIGURA 6.5 - Extensão WPDL-DOC para a classe <i>Document</i>	566
FIGURA 6.6 - Exemplo de dados para a extensão <i>DocList</i>	566
FIGURA 6.7 - Extensão WPDL-DOC para instâncias de documentos	57
FIGURA 6.8 - Extensão WPDL-DOC para as classes <i>Activity on Documents</i> e <i>Allowed Operations</i>	58
FIGURA 6.9 - Exemplo de dados para a extensão <i>ActivityDocs</i>	59
FIGURA 6.10 - Processo de modelagem de documentos e <i>workflow</i>	61
FIGURA 6.11 - Representação gráfica para o processo de licenciamento ambiental	64
FIGURA 6.12 - <i>Script</i> WPDL-DOC referente ao processo exemplo	68
FIGURA 7.1 - Arquitetura detalhada da proposta de integração	700
FIGURA 7.2 - Diagrama de classes para a camada de serviços	733
FIGURA 7.3 - Diagrama de seqüência para o método <i>Login</i>	76
FIGURA 7.4 - Diagrama de seqüência para o método <i>Logout</i>	77
FIGURA 7.5 - Diagrama de seqüência para o método <i>GetProcessDefinitions</i>	777
FIGURA 7.6 - Diagrama de seqüência para o método <i>StartProcess</i>	78
FIGURA 7.7 - Diagrama de seqüência para o método <i>GetWorkList</i>	79
FIGURA 7.8 - Diagrama de seqüência para o método <i>Start</i>	80
FIGURA 7.9 - Diagrama de seqüência para o método <i>CreateDocument</i>	822
FIGURA 7.10 - Diagrama de seqüência para o método <i>CreateDocumenteFromType</i>	833
FIGURA 7.11 - Diagrama de seqüência para o método <i>GetVersionList</i>	844
FIGURA 7.12 - Diagrama de seqüência para o método <i>Open</i>	855
FIGURA 7.13 - Diagrama de seqüência para o método <i>Save</i>	866
FIGURA 7.14 - Diagrama de seqüência para o método <i>CreateVersion</i>	877
FIGURA 7.15 - Diagrama de seqüência para o método <i>RemoveVersion</i>	889
FIGURA 7.16 Diagrama de seqüência para o método <i>Complete</i>	89

FIGURA 7.17 - Diagrama de seqüência para o método <i>Reassign</i> _____	900
FIGURA 7.18 - Tela principal do aplicativo cliente _____	91
FIGURA 7.19 - Tela principal com a lista de processos a disparar _____	93
FIGURA 7.20 - Lista de trabalho do participante _____	94
FIGURA 7.21 - Acesso aos documentos ligados a um item de trabalho _____	95

Lista de Tabelas

TABELA 3.1 - Visão geral das principais entidades do meta-modelo da WfMC	30
TABELA 3.2 - Relação entre as entidades do meta-modelo e as cláusulas da WPDL	31
TABELA 3.3 - Principais funções da interface 2 da WfMC	33
TABELA 4.1 - Principais componentes do padrão DMA	39
TABELA 4.2 - Componentes do modelo de conteúdo DMA	40
TABELA 4.3 - Componentes do modelo de versões DMA	42
TABELA 6.1 - Classes do modelo conceitual integrado	49
TABELA 6.2 - Mapeamento entre o modelo integrado e os padrões WAPI e DMA	52
TABELA 6.3 - Atributos adicionais para a classe <i>Doc Type</i>	54
TABELA 6.4 - Atributos adicionais para a classe <i>Document</i>	55
TABELA 6.5 - Atributos adicionais para instâncias de documentos	57
TABELA 6.6 - Atributos adicionais para as classes <i>Activity on Documents</i> e <i>Allowed Operations</i>	58
TABELA 6.7 - Exemplos de valores possíveis para <i>AllowedOperations</i>	59
TABELA 6.8 - Documentos e tipos de documentos para o processo exemplo	62
TABELA 6.9 - Representação gráfica dos conceitos introduzidos	63
TABELA 7.1 - Mecanismos de integração utilizados	71
TABELA 7.2 - Função das partes identificadas da interface	92
TABELA 7.3 - Descrição dos botões da barra de ferramentas	92

Resumo

A utilização de sistemas de gerência de documentos e de sistemas de gerência de *workflow* vem crescendo de forma expressiva nos últimos anos, motivados pela expectativa de obter-se benefícios organizacionais importantes, como o maior compartilhamento da informação e o aumento da eficiência dos processos. Pelo fato de a maioria dos processos organizacionais envolver a manipulação de documentos, é necessário, em grande parte das vezes, que esses sistemas trabalhem em conjunto. No entanto, a maioria das integrações entre tais produtos são proprietárias, causando prejuízos como a redução da liberdade de escolha dos usuários, a maior dificuldade de integração com sistemas legados e o aumento da dependência em relação aos fornecedores dos produtos.

Para solucionar esse problema, este trabalho propõe uma arquitetura aberta de integração entre sistemas de gerência de documentos e sistemas de gerência de *workflow*. O fundamento dessa arquitetura está na utilização de dois padrões emergentes da indústria: o padrão DMA, proposto pela AIIM, e o padrão WAPI, proposto pela WfMC. O padrão DMA consiste de uma API, a ser invocada pelos aplicativos clientes de documentos, de uma SPI, a ser implementada e oferecida pelos sistemas de gerência de documentos, e de um *middleware*, responsável pela conexão entre os clientes e servidores de documentos. O padrão WAPI consiste de uma linguagem padrão para a definição de processos e de uma API, a ser invocada pelo aplicativo cliente de *workflow* e implementada pelo sistema de gerência de *workflow*.

A arquitetura proposta compõe-se de três elementos. Primeiramente, um mecanismo integrado para a modelagem de documentos e *workflow*. Assim, é possível definir quais documentos serão manipulados pelo processo, e que operações cada participante poderá executar sobre ele. Esse mecanismo baseia-se exclusivamente nos padrões oferecidos pela WAPI e pela DMA. Em segundo lugar, é definido e implementado, na linguagem Java, um módulo de software denominado *camada de serviços*, oferecendo serviços integrados de gerência de documentos e gerência de *workflow*. Os métodos dessa camada invocam os métodos WAPI e DMA necessários à sua execução. Assim, qualquer DMS aderente à DMA pode ser integrado a qualquer WFMS aderente à WAPI, de forma completamente transparente. Por final, é desenvolvido um aplicativo cliente, também em linguagem Java, que acessa os métodos da camada de serviços, oferecendo uma interface gráfica para o usuário.

Os resultados demonstram a viabilidade de uma integração aberta entre sistemas de gerência de documentos e sistemas de gerência de *workflow*. Ainda, mostram a importância de uma modelagem integrada de documentos e *workflow*.

Palavras-chave: gerência de documentos, gerência de workflow, modelagem de workflow, arquitetura de software, interoperabilidade

Title: “An Open Architecture for Document Management Systems and Workflow Management Systems Integration”

Abstract

The document management systems and the workflow management systems areas have been facing an impressive growth in the last years, driven by the desire to achieve important organizational benefits, as an increased information sharing and more efficient processes. Since most business processes involve document manipulation, it is often needed that these systems work together. However, most products are integrated in a proprietary way, therefore reducing user’s freedom of choice, increasing the difficulty of legacy systems integration and rising the dependency from product vendors.

In order to overcome this problem, this work proposes an open architecture for document management systems and workflow management systems integration. The architecture’s fundamentals is the usage of two emerging industry standards: DMA, proposed by AIIM, and WAPI, proposed by WfMC. DMA consists of an API, to be invoked by client document applications, of an SPI, to be implemented and offered by document management systems, and of a *middleware*, responsible for connecting clients to document servers. WAPI consists of a standard process definition language and of an API, to be invoked by workflow client applications and implemented by workflow management systems.

The proposed architecture is composed by three elements. First, an integrated mechanism for document and workflow modeling is defined. Therefore, it is possible to define which documents will be used in the process, and which operations each workflow participant will be allowed to execute over them. This approach is based exclusively on DMA and WAPI standards. Second, it is defined an implemented a software module called *services layer*, offering integrated document management and workflow management services, using the Java language. This layer’s methods invoke the WAPI’s and DMA’s methods necessary for their execution. Therefore, any DMA-compliant DMS can be integrated to any WAPI-compliant WFMS, in a totally transparent way. Finally, a client application to access services layer’s methods and offer the user a graphical interface is developed, also using the Java language.

The results demonstrate the viability of an open integration between document management systems and workflow management systems. They also show the importance of an integrated document and workflow modeling.

Keywords: document management, workflow management, workflow modeling, software architecture, interoperability

1 Introdução

Um dos mais fortes movimentos atuais da indústria de sistemas de informação é, sem dúvida, o acelerado crescimento da utilização de sistemas de gerência de documentos e de sistemas de gerência de *workflow* [FRU 96][KHO 95][SIL 95]. Essas duas tecnologias vêm, cada vez mais, deixando de ser encaradas como ferramentas para nichos específicos de mercado, e passando a ser vistas como componentes indispensáveis para a concepção e desenvolvimento de modernos sistemas de informação [DAL 95][KOU 97][SIL 95].

Em linhas gerais, um sistema de gerência de *workflow* (designado também, neste trabalho, por WFMS - *workflow management system*) pode ser definido como um sistema de informação capaz de suportar processos de negócios, envolvendo conceitos como a ordenação entre atividades, participantes e ferramentas de software invocadas [WMC 96a]. A atual emergência dos sistemas de *workflow* pode ser explicada, principalmente, como uma resposta tecnológica a uma série de transformações econômicas que vêm afetando a economia mundial nos últimos anos [SIL 95] [THE 95]. Entre essas mudanças, podemos citar o aumento exacerbado da concorrência entre as empresas - de local para global - e a mudança de comportamento de grande parte dos clientes - de reativo para ativo [HAM 94][SIL 95]. Estas mudanças vêm exigindo das organizações maior produtividade, menores custos e, ainda, melhor atendimento aos clientes. Assim, criou-se a necessidade de novas formas de gestão das organizações, capazes de responder aos desafios colocados. A percepção comum a praticamente todas as técnicas propostas, principalmente a da reengenharia de processos (*business process reengineering* - BPR) [HAM 94][RAM 94], é que as estruturas organizacionais não devem mais orientar-se pela divisão tradicional em áreas específicas, como departamentos e seções. Ao contrário, a estrutura da organização deve ser baseada nos *processos* que ela realiza. Por esta visão, logo, o conceito de processo passa a ser central para a empresa. Consequentemente, tornam-se igualmente primordiais os sistemas de informação capazes de exprimir e suportar o conceito de processos. Dessa forma, sistemas de *workflow* começaram a despertar o interesse das organizações como a tecnologia capaz de efetivamente implementar esse conceito. Essa atenção especial pode ser demonstrada pela considerável quantidade de estudos de caso encontrados na literatura, como os expostos em [BEL 95], [DEN 95], [LIB 95] e [ROS 95], os quais ilustram o alcance de diversos benefícios organizacionais desejados - aumento de produtividade, redução de custos e melhoria no atendimento aos clientes.

Por sua vez, um sistema de gerência de documentos (designado também, neste trabalho, por DMS - *document management system*) é um sistema de informação capaz de armazenar, recuperar e manter a integridade de documentos, entre outras funcionalidades [AII 98a]. Diversas razões explicam a atual efervescência do mercado de gerência de documentos. A principal delas é a percepção da vital importância que os documentos possuem como repositório do conhecimento das organizações, uma vez que a maior parte de suas informações vitais estão contidas em documentos não-estruturados [SAD 97]. Logo, a facilidade em armazenar, recuperar e conservar a integridade deste verdadeiro patrimônio intelectual torna-se um imperativo para manter as organizações produtivas e competitivas nos dias atuais [AII 98a]. Acrescente-se a isso as novas

exigências, em termos de volume de documentos e necessidades de controle, ditadas por normas técnicas como as normas ISO 9000, e delineia-se um panorama altamente favorável ao crescimento da utilização desses sistemas.

1.1 Necessidades de Integração

Ainda que, conforme exposto anteriormente, sistemas de gerência de documentos e sistemas de gerência de *workflow* possuam finalidades bastante diversas - uma vez que o primeiro preocupa-se com a manipulação de documentos e o segundo com a manipulação de processos - há um elo fortíssimo entre essas duas classes de sistemas de informação, gerando assim uma grande necessidade por sua integração [AII 98a][DAL 95][KHO 95]. Com essa integração, pode-se obter os benefícios combinados da utilização de ambos sistemas - o armazenamento e recuperação confiáveis dos documentos proporcionados por sistemas de gerência de documentos, e o controle, integridade e agilidade dos processos de negócios proporcionados por sistemas de gerência de *workflow* [JOO 95].

A necessidade de integração é perceptível basicamente em duas situações:

- em primeiro lugar, os processos de negócios executados pelas organizações envolvem, tipicamente, a manipulação de diversos documentos [KHO 95] [SAD 97]. Mais precisamente, são os documentos, via de regra, os objetos de informação que registram o desenrolar e as saídas geradas pelo processo. Por exemplo, em um processo de compra de materiais, estão envolvidos documentos como solicitações de compra, cotações de preços dos fornecedores, ordens de compra e registros de recebimento dos materiais. Deve-se ressaltar, portanto, que a um determinado processo, ou mesmo a uma determinada atividade, podem estar ligados diversos documentos. Os documentos envolvidos podem tanto terem sido gerados eletronicamente como terem sido digitalizados a partir de originais em papel. Ainda, é importante ressaltar a importância da utilização de um DMS para a manipulação dos documentos. Quando um DMS não é utilizado, os documentos são armazenados, em geral, diretamente no sistema de arquivos do sistema operacional, o qual é desprovido de recursos suficientes para realizar a gerência de documentos [FRU 96]. Conseqüências diretas da não utilização de um DMS são a dificuldade de acesso aos documentos desejados e a dificuldade de manter a integridade dos documentos [AII 98a].
- em segundo lugar, a gerência de documentos pode envolver, além dos já citados aspectos de armazenamento, recuperação e manutenção da integridade do documento, a especificação e o controle do *ciclo de vida* do documento [AMA 97a][SAD 97]. No ciclo de vida, podem-se estabelecer atributos como as atividades possíveis sobre cada documento, as relações de dependências entre elas, e quem possui permissão para executá-las. Por exemplo, pode-se definir que o ciclo de vida de um documento do tipo 'ata de reunião' conterà as atividades de criação da ata, de revisão pelo coordenador e de aprovação final pelo grupo. É fácil perceber, então, que o ciclo de vida de um documento

constitui-se em um tipo de processo, pois incorpora os elementos básicos deste, como atividades, dependência entre atividades e a definição de participantes. Logo, é natural que a gerência do ciclo de vida de um documento seja feita por um WFMS. Observe-se, portanto, que a um determinado documento - ou tipo de documento - corresponde um único ciclo de vida. Via de regra, os documentos para os quais é definido um ciclo de vida são gerados e mantidos totalmente em meio eletrônico; apenas em casos excepcionais o documento é obtido a partir da digitalização de um original em papel [FRU 96] .

Dada essa estreita relação entre a utilização destas tecnologias, é compreensível o porquê da dificuldade, bastante comum, de dissociar a gerência de *workflow* da gerência de documentos [JOO 95][MAR 95]. Alguns dos primeiros recursos de *workflow*, incluindo funcionalidades básicas como a definição de atividades e dependências, foram inicialmente incorporados aos produtos de gerência de imagens da FileNet em 1985. A indústria de sistemas de *workflow* propriamente dita, no entanto, somente se estabeleceu em torno de 1992, quando empresas como Action Technologies, FileNet, Keyfile e Sigma posicionaram seus produtos como sistemas de gerência de *workflow* [SIL 95]. Em um curto período de tempo, logo, a indústria de sistemas de *workflow* adquiriu um perfil próprio. Atualmente, já existe grande diversidade de produtos e técnicas de modelagem específicos para *workflow* [GEO 95]. Ainda hoje, no entanto, o segmento mais consolidado da área de *workflow* e, conseqüentemente, onde encontram-se as aplicações mais bem-sucedidas, continua a ser o centrado em documentos, devido a sua grande abrangência de aplicação e a sua origem anterior. Ao mesmo tempo, a inclusão de recursos de *workflow* tornou-se um diferencial mercadológico crítico para sistemas de gerência de documentos, a ponto de aqueles que não os incorporaram serem considerados defasados tecnologicamente [SIL 95].

1.2 Objetivos

Em vista dos fatos expostos, a integração entre sistemas de gerência de documentos e sistemas de gerência de *workflow* revela-se uma solução altamente atraente para diversas áreas do mercado, devendo ser, portanto, facilitada e incentivada ao máximo. Atualmente, no entanto, a integração entre esses sistemas é realizada quase que somente através de suas respectivas interfaces proprietárias, via de regra firmada no bojo de acordos comerciais entre as empresas desenvolvedoras dos sistemas [MAR 97]. Isso faz com que um determinado WFMS consiga comunicar-se com apenas um pequeno conjunto de sistemas de gerência de documentos, para os quais uma integração específica foi realizada. De modo análogo, isso faz com que um determinado DMS consiga comunicar-se com apenas um conjunto restrito de sistemas de gerência de *workflow*, para os quais uma integração específica foi feita.

Evidentemente, tal situação é altamente indesejável. Mowbray, em [MOW 95], coloca diversos prejuízos decorrentes da dependência de software integrado de forma proprietária, tais como: i) inibição da extensibilidade do sistema, em função do formato proprietário de dados utilizado; ii) dependência extrema do fornecedor da solução para qualquer alteração e/ou evolução no sistema, inclusive no fator custo; iii) restrição das possibilidades de escolha de WFMS e DMS, pois apenas as combinações que possuem

integração entre si poderiam ser adquiridas. Mowbray aponta ainda que, neste último caso, é possível que a combinação de softwares mais adequados a uma dada organização - em termos de funcionalidades apresentadas, custos e suporte técnico, entre outros - não consiga ser utilizada, devido à inexistência de uma integração entre os produtos e à grande complexidade e elevado custo necessários para realizar essa integração, gerando assim frustração e descontentamento junto às organizações usuárias.

Felizmente, vem crescendo significativamente, nos últimos anos, a percepção de que essa situação é altamente prejudicial à consolidação dos mercados de gerência de documentos e de gerência de *workflow*. A inexistência de padrões é um fator que gera desconfiança e insegurança junto às organizações usuárias, e é usualmente considerada por elas uma prova da imaturidade da tecnologia [MOW 95]. De modo a reagir a essa situação, estabeleceram-se órgãos padronizadores para ambas áreas. Empresas desenvolvedoras de sistemas de *workflow* criaram, em 1993, um consórcio denominado WfMC - *Workflow Management Coalition*, para difundir e padronizar a área. Por sua vez, a *Association for Information & Image Management* (AIIM), tradicional consórcio de empresas de gerência de documentos, criou, em 1995, uma força-tarefa denominada DMA - *Document Management Alliance*, com o propósito de desenvolver padrões para a área de gerência de documentos.

É importante frisar que ambos padrões foram concebidos de forma absolutamente independente, a despeito dos expressivos laços que unem essas tecnologias, descritos na seção 1.1. Isto significa que os padrões concebidos pela WfMC não incluem nem prevêm qualquer ligação com sistemas de gerência de documentos, e tampouco os padrões ditados pela AIIM contemplam integração com sistemas de gerência de *workflow*.

Identifica-se, desta forma, uma lacuna de extrema importância na integração entre sistemas de gerência de documentos e sistemas de gerência de *workflow*. É justamente este o objetivo central desta dissertação de mestrado: propor uma arquitetura aberta capaz de integrar sistemas de gerência de documentos a sistemas de gerência de *workflow* através dos padrões definidos pela indústria: o padrão DMA (*Document Management Alliance*), definido pela AIIM, e o padrão WAPI (*Workflow API*), definido pela WfMC. Dessa forma, será possível integrar um sistema de gerência de documentos qualquer a um outro sistema de gerência de *workflow* qualquer, bastando apenas que ambos suportem, respectivamente, as interfaces DMA e WAPI. Com isso, espera-se romper com o *status quo*, facilitando a integração de produtos diversos por terceiros e elevando significativamente a liberdade de escolha das organizações, beneficiando diretamente as organizações usuárias.

É interessante ressaltar que o padrão definido pela força-tarefa DMA também é referenciado, na literatura da AIIM, por DMA [AII 98a]. Assim, para evitar confusões no decorrer do texto, doravante será utilizada a sigla DMA para designar os padrões desenvolvidos, e a sigla AIIM para designar o consórcio autor do padrão.

1.3 Estrutura do Trabalho

No intuito de tornar claro o processo evolutivo que culminou neste trabalho, serão apresentados, nos próximos capítulos, todos os embasamentos e passos percorridos. É importante ressaltar que, no decorrer deste trabalho, toda terminologia de *workflow* empregada estará de acordo com o significado estabelecido pela WfMC em [WMC 96a], sempre que o termo tiver sido por ela definido. De forma similar, toda a terminologia de documentos empregada estará de acordo com o significado estabelecido pela especificação da DMA [AII 97], sempre que o termo tiver sido por ela definido. Assim, pretende-se evitar quaisquer ambigüidades na leitura do texto.

Este texto é dividido da seguinte maneira. No capítulo 2, são descritas em detalhe as diversas formas atualmente utilizadas para a integração entre sistemas de gerência de documentos e sistemas de gerência de *workflow*. A seguir, no capítulo 3, são apresentados os padrões propostos pela WfMC. O capítulo 4 é destinado à apresentação dos padrões de gerência de documentos propostos pela AIIM. O capítulo 5 apresenta, em linhas gerais, a arquitetura proposta neste trabalho. Os aspectos de modelagem da arquitetura são apresentados no capítulo 6. O capítulo 7 apresenta o protótipo desenvolvido nesta dissertação, demonstrando a viabilidade da arquitetura proposta. Por final, o capítulo 8 apresenta as conclusões deste trabalho.

É importante observar que não é preocupação deste trabalho oferecer uma visão introdutória e genérica às áreas de gerência de *workflow* e gerência de documentos. Para tal, o leitor é convidado a buscar subsídios em [GEO 95] e [AMA 97b], para uma visão sobre sistemas de gerência de *workflow*, e em [SOA 95], para uma visão sobre sistemas de gerência de documentos.

2 Formas Atuais de Integração

Neste capítulo serão apresentadas as diversas formas pelas quais as funcionalidades de gerência de documentos e de gerência de *workflow* são atualmente integradas. Para cada uma das formas identificadas será descrita a arquitetura de software que ela representa, sendo feita, então, uma análise de vantagens e desvantagens, levando em consideração os recursos oferecidos e, principalmente, as facilidades de integração com outros sistemas de informação. Ao final do capítulo, será feita uma comparação entre as técnicas, de forma a definir e justificar os rumos deste trabalho. É importante ressaltar que a classificação aqui apresentada foi definida no escopo desta dissertação, constituindo-se, portanto, em material inédito.

2.1 DMS Estendidos para *Workflow*

É reconhecido que uma das principais origens dos atuais sistemas de gerência de *workflow* foram os sistemas de gerência de documentos, particularmente os sistemas de gerência de imagens [KHO 95][SIL 95]. Diversos desenvolvedores desses sistemas perceberam, em um dado momento, que a efetiva gerência de documentos requeria mais do que recursos como autoria, digitalização, armazenamento e recuperação. Percebeu-se, então, a necessidade de poder-se especificar o *ciclo de vida* dos documentos, estabelecendo, entre outros, atributos como as atividades possíveis sobre cada documento, as relações de dependência entre elas, e quem possui permissão para executá-las [KHO 95].

Dessa forma, diversos produtos de gerência de documentos passaram a incluir funcionalidades de *workflow*, como roteamento, distribuição automática e acompanhamento do trabalho, constituído por formulários e documentos a serem processados [SIL 95]. A adição desses recursos de *workflow* foi especialmente útil para a gerência de documentos complexos, como na elaboração de documentos compostos onde seja necessária a participação de diversos autores (autoria cooperativa). Problemas típicos da autoria cooperativa, como a dificuldade de conhecer o status corrente dos documentos e a dificuldade de saber qual participante está controlando qual componente em um dado momento, podem ser naturalmente resolvidos através dessas funcionalidades [REA 97a].

A primeira empresa a prover tais funções em seus produtos de gerência de documentos foi a FileNet, no ano de 1985, com um grande êxito em termos comerciais [SIL 95]. Como consequência, em um prazo de alguns anos a grande maioria dos sistemas de gerência de documentos incorporaram recursos de *workflow* a seus produtos. Atualmente, podem ser classificados nessa categoria produtos populares como o Docbase, da Documentum [REA 97a], o DOCS Enterprise Suite, da PC Docs [PCD 98b] e o LAVA, da LAVA Systems [LAV 97].

É preciso esclarecer que, mesmo incorporando funções de *workflow*, tais produtos continuam a ser considerados sistemas de gerência de documentos. Isto é

demonstrado tanto pelo posicionamento comercial das empresas - que rotulam esses produtos como DMS - quanto pelos próprios recursos de *workflow* oferecidos. Esses produtos incorporam tão-somente os recursos de *workflow* necessários à modelagem e ao suporte do ciclo de vida de documentos. Assim, muitos recursos fundamentais em WFMS modernos, como a invocação de aplicativos externos e a comunicação com outros sistemas, não costumam ser contemplados nesses sistemas [REA 97a]. Em razão disso, esses produtos foram, neste trabalho, classificados como "DMS estendidos para *workflow*".

Em termos de arquitetura de software, preocupação central desse trabalho, tais produtos não apresentam qualquer separação entre as funcionalidades de gerência de documentos e as funcionalidades de gerência de *workflow* que seja perceptível aos usuários. Isso significa que, ao adquirir um produto dessa categoria, não há possibilidade de trocar-se o módulo de gerência de *workflow* por um módulo de um outro fornecedor, pela simples razão de que não existe uma separação clara entre as funcionalidades. A arquitetura dessa categoria de produtos pode ser, assim, descrita como monolítica, onde tanto os recursos de gerência de documentos quanto de gerência de *workflow* encontram-se em um único componente de software.

A figura 2.1 mostra esquematicamente essa arquitetura. A sua figura central é o *DMS com recursos de workflow*, que oferece, através de uma API proprietária, serviços de gerência de documentos e de gerência de *workflow*. Esses serviços são utilizados por *aplicativos de documentos e workflow*, que oferecem uma interface de usuário para eles. Esses aplicativos, em muitos casos, são fornecidos pela mesma empresa desenvolvedora do DMS. Entre os produtos que operam dessa forma, pode-se citar o DocBase, da Documentum [REA 97a].

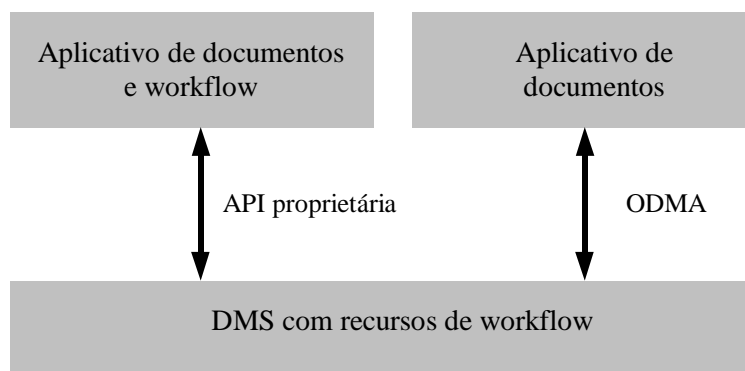


FIGURA 2.1 - Arquitetura dos DMS estendidos para workflow

A partir de 1996, com a definição, pela AIIM, do padrão ODMA (*Open Document Management API*) [AII 98b], alguns produtos passaram a permitir que aplicações cliente desenvolvidas por terceiros utilizassem seus serviços. Desta forma, um aplicativo cliente de documentos aderente ao ODMA, como o Microsoft Word 97 [ACT 98a], consegue interoperar com um DMS também aderente ao ODMA, como o Documentum [ACT 98b][DOC 97]. No entanto, como o padrão ODMA envolve apenas

funcionalidades de gerência de documentos, os recursos de *workflow* não conseguem ser utilizados neste caso.

2.1.1 Vantagens

Certamente, a maior vantagem desta forma de integração, do ponto de vista das organizações usuárias, reside na facilidade de utilização do produto. Uma vez que o produto já é concebido integrando funções de documentos e de *workflow*, não há necessidade de a organização usuária efetuar desenvolvimentos adicionais ao produto. De modo similar, os aplicativos de documentos e *workflow* fornecidos juntamente com esses produtos também apresentam uma interface integrada, facilitando a interação com os usuários.

De modo geral, pode-se afirmar que, quando o interesse da organização resume-se exclusivamente à manipulação do ciclo de vida de documentos, essa solução de integração pode ser bastante adequada.

2.1.2 Desvantagens

As desvantagens dessa forma de integração começam a aparecer no momento em que as organizações usuárias passam a desejar a automatização de processos de negócios mais complexos, que não estejam limitados unicamente à manipulação de documentos. Em um grande número de casos, por exemplo, o processo pode necessitar fazer acesso a uma base de dados externa, ou ser notificado de eventos externos, como a recepção de fax e de correio eletrônico e a ativação de periféricos, entre outros. Recursos como esses não estão, tipicamente, presentes em DMS estendidos para *workflow*. Esses produtos, logo, oferecem à organização recursos limitados para a gerência de *workflow*, o que pode constituir-se em um sério problema em futuras expansões do sistema.

Uma outra gama de problemas surge quando analisa-se o relacionamento de um produto desse tipo com outros sistemas já existentes na organização (legados) ou com sistemas de empresas parceiras com os quais ele precise se comunicar (por exemplo, para a constituição de uma empresa virtual). Se a organização ou uma parceira já possuir, por exemplo, um sistema de gerência de *workflow*, um DMS estendido para *workflow* dificilmente conseguiria comunicar-se com ele, uma vez que ele próprio implementa as funcionalidades de *workflow*. Desta forma, podem ocorrer cenários de difícil solução, como os seguintes:

- pode ser necessário relacionar um processo gerenciado pelo WFMS com o ciclo de vida de documentos gerenciado pelo DMS estendido para *workflow*;
- seria necessário descrever os elementos da organização (participantes, papéis e seus respectivos atributos) tanto no WFMS quanto no DMS estendido para *workflow*, assim como quaisquer modificações que sofram, multiplicando o esforço necessário para administrar o sistema;

- a administração de aspectos dinâmicos do sistema, como o balanceamento de carga entre os participantes, torna-se complexa, pois as informações são geradas por dois sistemas independentes;

Claramente, tais situações são altamente indesejáveis, e a sua solução, via de regra, exigirá expressivos investimentos por parte das organizações usuárias, como a mudança de produto, com a conseqüente migração de informações, e a construção de módulos de software que conectem um sistema ao outro [MOW 95].

2.2 Integração Proprietária de DMS e WFMS

A partir de 1992, diversos fabricantes de sistemas de gerência de documentos estendidos para *workflow* decidiram lançar produtos específicos para gerência de *workflow*. Esse posicionamento mercadológico ocorreu em função da grande demanda por sistemas de informação capazes de representar os processos de negócios das organizações, independentemente da manipulação de documentos [SIL 95]. Desde então, empresas como a FileNet, a IBM e a Oracle, entre diversas outras, vêm oferecendo tanto produtos de gerência de documentos quanto de gerência de *workflow*, separadamente. No entanto, todos esses fornecedores oferecem a opção de vender seus produtos já integrados através de suas respectivas interfaces proprietárias [MAR 97]. Assim, uma organização pode adquirir o DMS e o WFMS da FileNet já previamente integrados, mesmo tratando-se de produtos separados.

Além dessa situação, há outros casos de integrações prévias de produtos. De forma a tornarem-se mais competitivos, diversos fornecedores de DMS e fornecedores de WFMS aliaram-se, oferecendo seus respectivos produtos de forma pré-integrada. Tipicamente, tais integrações são realizadas no escopo de acordos comerciais mais extensos. Exemplos desses acordos são a integração entre o DMS Docbase, da Documentum, e o WFMS InConcert, da InConcert [INC 97]; a integração entre o DMS DOCS Open, da PC DOCS, e o WFMS WorkMAN, da Reach [REA 97b]; e a integração entre o DMS Saros, da FileNet, e o WFMS Action *Workflow*, da Action Technologies [MAR 97].

Uma vez que existem dois produtos independentes, a arquitetura de software resultante é consideravelmente mais complexa. Tipicamente, cada um dos produtos possui uma API proprietária, utilizada por aplicativos cliente. Para realizar essa integração faz-se necessário, primeiramente, adaptar os aplicativos cliente para que passem a utilizar as API proprietárias de *ambos* produtos, e não de apenas um deles. Por exemplo, um aplicativo para visualizar a lista de trabalho do participante - fornecido juntamente com o WFMS - pode, por exemplo, ser alterado para permitir que os documentos ligados ao item de trabalho sejam buscados no DMS. Para tal é necessário, evidentemente, incluir no código desse aplicativo chamadas à API proprietária do DMS. Similarmente, alterações podem ser necessárias nos aplicativos fornecidos juntamente com o DMS, permitindo assim que façam acesso à API proprietária do WFMS. Por exemplo, na integração realizada entre o DMS Docbase e o WFMS WorkMAN, todas as funcionalidades de *workflow* foram incluídas diretamente no aplicativo cliente oferecido junto com o DOCS Open, denominado DOCS Desktop [REA 97b].

A segunda ação necessária para essa integração é relacionar os serviços oferecidos pelo WFMS com os serviços oferecidos pelo DMS. Há uma variedade de situações onde um dos sistemas terá que conhecer o outro, seja para manter dados a seu respeito ou para invocar a sua respectiva API proprietária. Por exemplo, se um item de trabalho pode conter documentos anexados, é preciso que o WFMS conheça qual a forma de identificação de documentos utilizada pelo DMS a ele integrado, permitindo que esses documentos sejam recuperados e apresentados na lista de trabalho. Além disso, a integração pode permitir por exemplo, que campos do documento sejam utilizados como dados de decisão em regras do fluxo - neste caso, é preciso que o DMS envie dados ao WFMS. Um exemplo bastante ilustrativo deste aspecto da integração é oferecido pelos produtos DOCS Open e WorkMAN. Sendo um WFMS, o WorkMAN é responsável por manter o histórico das instâncias de processo, registrando dados sobre a execução de cada atividade. Na integração realizada, além de atualizar a sua própria base de dados, o WorkMAN também atualiza a base de dados proprietária do DOCS Open - a qual, evidentemente, é diferente da sua [REA 97b].

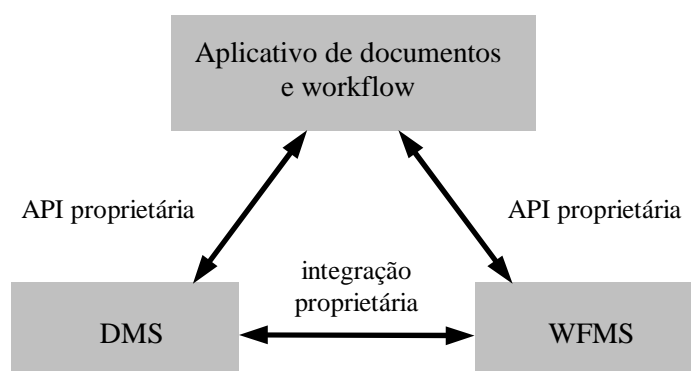


FIGURA 2.2 - Arquitetura de integração proprietária entre DMS e WFMS

A figura 2.2 mostra esquematicamente essa arquitetura. Como citado anteriormente, ela é composta por três elementos: o DMS, o WFMS e aplicativos de documentos e *workflow*. Os aplicativos utilizam os serviços oferecidos pelo DMS e pelo WFMS, através de suas respectivas interfaces proprietárias, o que é ilustrado pelas setas entre eles. Por sua vez, o DMS e o WFMS precisam trocar informações, como nos casos ilustrados no parágrafo anterior, o que é simbolizado pela seta entre o DMS e o WFMS, com o rótulo "integração proprietária".

2.2.1 Vantagens

A principal vantagem da utilização desta forma de integração é a riqueza de recursos oferecida, tanto de gerência de documentos quanto de gerência de *workflow*. Isto deve-se ao fato de que a integração é feita entre dois produtos completos, e o sistema resultante possuirá todas as funcionalidades dos dois produtos. Desta forma, não ocorre o risco existente com os DMS estendidos para *workflow* - a descoberta tardia que o produto é insuficiente para as necessidades da organização.

2.2.2 Desvantagens

Um dos grandes problemas dessa técnica são as poucas opções de integração que são oferecidas às organizações usuárias. Uma vez que tais integrações dependem de acordos comerciais, e que, devido a sua já comentada alta complexidade, levam um tempo relativamente longo para serem implementadas, cada DMS integra-se a um pequeno conjunto de WFMS, raramente em número maior que dois. Dessa forma, as organizações usuárias têm sua liberdade de escolha significativamente reduzida. Assim, é possível que sejam obrigadas a adquirir produtos que não sejam os mais adequados em termos de atendimento às necessidades da organização, preço, prazo de entrega, suporte, treinamento, ou qualquer outro fator decisório relevante, causando-lhes assim evidente insatisfação e prejuízo [MOW 95].

Além disso, uma vez adquirida a combinação de produtos, é extremamente difícil trocar algum dos sistemas por um outro similar. Por exemplo, mesmo que a organização deseje trocar o WFMS da solução integrada por um outro, considerado mais adequado às suas necessidades, ela dificilmente conseguirá fazer isso. Isto deve-se ao fato de que, como as integrações são *proprietárias*, ambos produtos são modificados para adaptarem-se ao outro. Logo, a troca de somente um componente não é trivial. Em verdade, a organização, ao optar por uma solução de integração como essa, compromete-se em demasia, ficando assim extremamente dependente das empresas fornecedoras dos produtos.

Assim como é difícil a troca de um produto da solução, é difícil, pelas mesmas razões, a comunicação com outros DMS e WFMS que porventura já existam na organização ou em empresas parceiras. [SWE 95] aponta para o alto risco da geração de "ilhas de informação" nas corporações, formadas por sistemas incapazes de interoperar de forma aberta com os demais. Sem dúvida, a integração de produtos dessa forma conduz a grandes possibilidades de criar-se ilhas de informação nas organizações.

2.3 Integração *Ad hoc* de DMS e WFMS

As duas formas anteriormente apresentadas de integração entre DMS e WFMS são desenvolvidas e oferecidas diretamente pelos próprios fabricantes dos sistemas. Pode ocorrer, no entanto, a necessidade que terceiros realizem a integração entre os produtos - geralmente empresas revendedoras dos produtos e fornecedores de soluções de integração. Esta forma de integração é denominada, neste trabalho, de *integração ad hoc*. Ela pode ser escolhida pela organização por diversas razões. Se os produtos a serem adquiridos não forem pré-integrados pelos respectivos fabricantes, ela será certamente a única opção disponível. Similarmente, quando deseja-se integrar um novo sistema a outro já existente na organização (legado) ou de uma organização parceira, é bastante provável que a integração *ad hoc* seja a única escolha possível.

De modo geral, a arquitetura resultante dessa forma de integração é bastante similar à da integração proprietária apresentada na seção 2.2. Assim como na integração proprietária, é preciso aqui, primeiramente, adaptar os aplicativos cliente para acessar as API dos produtos e, em seguida, relacionar os serviços oferecidos pelo WFMS com os oferecidos pelo DMS. No entanto, uma vez que, salvo casos excepcionais, os

integradores não têm acesso ao código-fonte dos produtos, os procedimentos reais de integração tendem a ser diferentes.

No caso da adaptação dos aplicativos cliente, por exemplo, se o código-fonte não estiver disponível, será, evidentemente, impossível integrá-los. Restam aos integradores, em geral, apenas duas opções: oferecer os aplicativos de *workflow* e de documentos desvinculados um do outro, ou desenvolver novos aplicativos cliente, utilizando então as APIs de ambos produtos. Na primeira opção, então, cada aplicativo utilizaria somente a API proprietária do produto com o qual é oferecido. Eventualmente, algum dos aplicativos cliente pode ter recursos extras que permitam a sua ligação com outros, como a possibilidade de parametrização e a criação de *scripts*.

Para relacionar os serviços oferecidos pelo WFMS com os serviços oferecidos pelo DMS será necessário, como na integração proprietária, permitir que um sistema conheça o outro. O fato de não se ter à disposição o código-fonte dos produtos pode aumentar a dificuldade de construir esses relacionamentos, obrigando os integradores a criar componentes de software intermediários para interligar os sistemas. Um exemplo típico de técnica para esse fim é a utilização de DDE (*Dynamic Data Exchange*), um recurso do sistema operacional Microsoft Windows para troca de dados entre aplicativos.

A figura 2.3 mostra esquematicamente essa arquitetura. Os segmentos de reta entre o DMS e o WFMS (com o rótulo "integração dos serviços") simbolizam a necessidade de relacionar ambos sistemas, ilustrada no parágrafo anterior, e os círculos entre eles denotam a construção de módulos intermediários para a comunicação entre os sistemas.

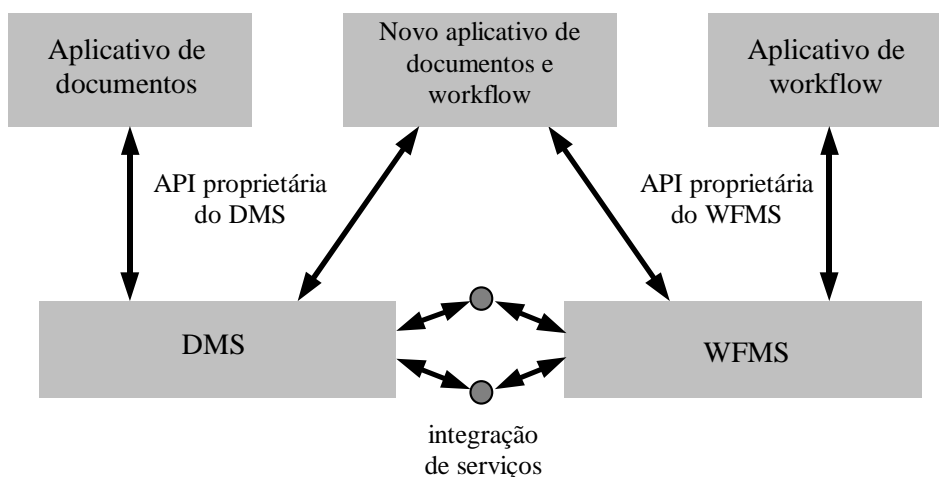


FIGURA 2.3 - Arquitetura de integração *ad hoc* entre DMS e WFMS

2.3.1 Vantagens

A principal vantagem da utilização desta forma de integração é o atendimento das necessidades exatas de integração entre os sistemas da organização, uma vez que o código é escrito especificamente para a sua realidade. Em teoria, qualquer par WFMS-

DMS pode ser integrado, bastando somente que ambos produtos possuam uma API publicada.

Além disso, o fato de ter-se dois produtos completos oferece uma grande riqueza de recursos, tanto de gerência de documentos quanto de gerência de *workflow*. Desta forma, não ocorre o risco existente com os DMS estendidos para *workflow* - a descoberta tardia que o produto é insuficiente para as necessidades da organização.

2.3.2 Desvantagens

Um dos grandes problemas dessa forma de integração é o seu elevado custo, tanto de criação como de manutenção. Nessa arquitetura, ao contrário das duas anteriormente vistas, é necessário que a organização usuária desenvolva módulos específicos para a integração (ou contrate o seu desenvolvimento). Isso, como já visto, pode incluir, por exemplo, o desenvolvimento de um novo aplicativo cliente para o sistema - um custo certamente muito alto e, para a grande maioria das organizações, proibitivo.

O alto custo de manutenção da solução pode ser observado em diversos aspectos. Por exemplo, a cada nova versão do DMS ou do WFMS utilizado, é possível que alterações tenham que ser feitas nos módulos de integração desenvolvidos. Além disso, como aponta [MOW 95], soluções de integração ad hoc costumam utilizar recursos avançados e específicos do sistema operacional (como o próprio DDE, anteriormente citado), os quais podem sofrer alterações em versões subseqüentes do sistema operacional, tornando obsoleta a solução e exigindo alterações nos módulos de integração. [MOW 95] ainda aponta o fato de que, via de regra, os artifícios utilizados para a integração não são adequadamente documentados, fazendo com que em torno de 50% do tempo de desenvolvimento seja consumido descobrindo como o sistema funciona (*discovery time*).

Além disso, uma vez realizada a integração entre os produtos, é extremamente difícil trocar algum dos sistemas por um outro similar. Por exemplo, mesmo que a organização deseje trocar o WFMS da solução integrada por um outro, considerado mais adequado às suas necessidades, ela dificilmente conseguirá fazer isso, pois os módulos de integração foram construídos utilizando as APIs proprietárias de cada produto. Neste caso, muito provavelmente, seria necessária a criação de novos módulos de integração para a interligação com o novo sistema, duplicando, assim, o volume de software a ser mantido em operação. Na prática, a organização tende a arcar com altos custos para realizar qualquer extensão no sistema.

Por fim, deve ser ressaltado que a integração ad hoc entre um DMS e um WFMS, por ser uma atividade de alta complexidade, pode tornar-se um elemento de risco no planejamento de um projeto de implantação desses sistemas. É perfeitamente possível que, por exemplo, sejam encontradas dificuldades inesperadas no processo de integração, gerando atrasos no cronograma e prejudicando o andamento do projeto como um todo.

2.4 Conclusões

Como demonstrado nas seções anteriores, as arquiteturas atuais para a integração de sistemas de gerência de documentos e sistemas de gerência de *workflow* apresentam diversos pontos negativos. A fusão de funcionalidades de gerência de documentos e de *workflow* em um só produto, como já colocado, assim como as formas proprietárias de integração entre DMS e WFMS, constituem-se em imposições indesejáveis às organizações usuárias desses sistemas, ora limitando a sua aplicação dentro da organização, ora dificultando ou mesmo impossibilitando a sua integração com outros sistemas já existentes.

A principal deficiência dessas arquiteturas de integração é o fato de serem *proprietárias*, isto é, a comunicação entre os softwares é feita através das APIs proprietárias dos produtos. Como ressalta [MOW 95], a adoção de uma arquitetura proprietária pode ocasionar, entre outros efeitos, o surgimento de ilhas de informação, a excessiva dependência das empresas fornecedores dos produtos e altos custos de desenvolvimento e manutenção das soluções de integração.

De modo a oferecer uma alternativa a essa situação, este trabalho propõe uma arquitetura de integração baseada em padrões da indústria. Na indústria de sistemas de gerência de *workflow*, a *Workflow Management Coalition* (WfMC) vem desenvolvendo o padrão WAPI (*Workflow API & Interchange Formats*), enquanto, na indústria de sistemas de gerência de documentos, a *Association for Information and Image Management* (AIIM) vem desenvolvendo o padrão DMA (*Document Management Alliance*). Logo, através dessas APIs, é possível construir uma arquitetura de integração *aberta*, uma vez que será independente de produtos. De fato, nesta arquitetura, basta que os produtos suportem a API padrão correspondente para que eles possam ser integrados à solução.

É importante observar que uma arquitetura baseada nesses padrões deve apresentar, forçosamente, o WFMS separado do DMS. Esse fato é reflexo do atual processo de especialização dos produtos de DMS e WFMS, isto é, diversos fabricantes de sistemas estão transformando produtos de DMS estendidos para *workflow* em DMS e WFMS. Os padrões WAPI e DMA seguem essa tendência, pelo fato de mostrarem-se independentes entre si. Dessa forma, o sistema resultante da arquitetura proposta poderá combinar o máximo da funcionalidade dos dois sistemas, evitando-se incorrer nas já citadas restrições existentes nos DMS estendidos para *workflow*.

A fim de tornar claros todos os fundamentos desse trabalho, os capítulos 3 e 4 a seguir detalharão, respectivamente, os padrões desenvolvidos pela WfMC e pela DMA. Após, nos capítulos 5, 6 e 7, será apresentada em detalhes a proposta de arquitetura aberta aqui introduzida.

3 Padrões para Sistemas de Gerência de *Workflow*

Neste capítulo serão apresentados os padrões para sistemas de gerência de *workflow*, definidos pela *Workflow Management Coalition* (WfMC). Primeiramente, serão apresentadas as motivações do esforço de padronização da WfMC. Logo após, será descrito o modelo de referência por ela desenvolvido, ilustrando a relação entre os padrões criados. Por final, serão detalhados dois desses padrões - o padrão para definição de processos e o padrão para acesso a serviços de gerência de *workflow* -, em virtude de sua posterior utilização nesse trabalho.

3.1 Motivações

A grande maioria dos primeiros sistemas de *workflow* eram sistemas *fechados*, isto é, seus componentes não possuíam interfaces publicadas, que pudessem ser utilizadas por sistemas de terceiros. Assim, todos os componentes de software - o sistema de gerência de *workflow*, a ferramenta de modelagem de processos e o ambiente para construção dos aplicativos e da interface de usuário - tinham de ser adquiridos de um único fornecedor. Além disso, não era possível transferir dados de um sistema de *workflow* para outro. Pode-se dizer que, para todos os fins práticos, os sistemas existentes eram inteiramente incompatíveis [WfMC 94].

Nesse contexto, dois fatores surgiram como propulsores da interoperabilidade de sistemas de *workflow*: a necessidade de flexibilidade operacional e a especialização de produtos.

A flexibilidade operacional, atualmente um aspecto extremamente crítico para as organizações [HAM 94], pode ficar comprometida na existência de sistemas de *workflow* incompatíveis. Por exemplo, a integração entre dois departamentos de uma organização que já utilizassem sistemas de gerência de *workflow* diferentes tornaria-se inviável. Da mesma forma, duas organizações que desejassem compartilhar um mesmo processo (por exemplo, um processo que relacionasse um determinado fornecedor a um cliente seu) seriam obrigadas a ter sistemas de *workflow* do mesmo fornecedor, o que dificilmente ocorreria na prática. É importante ressaltar que um grande número de projeções de mercado, como [KOU 97] e [SIL 95], aponta um crescimento explosivo da utilização de sistemas de *workflow* durante os próximos 5 a 10 anos, onde essas incompatibilidades podem levar a sérios problemas. Esta visão é claramente expressa pela WfMC; inclusive, sua expectativa é de que a disponibilização de padrões para a área de *workflow* aumente a confiança do mercado, estimulando-o ainda mais a adotar esta tecnologia [WfMC 98a].

Como segundo fator, a padronização da área de *workflow* permitirá a especialização dos produtos existentes. Em uma arquitetura fechada, cada fornecedor precisa desenvolver todos seus componentes. Com a existência de padrões, um sistema de *workflow* poderá ser criado selecionando componentes específicos de cada fornecedor. Assim, pode-se escolher a ferramenta de modelagem de processos de um

fornecedor, o sistema de gerência de *workflow* de outro, e um ambiente para a construção de aplicativos de um terceiro. A expectativa da WfMC é que, dessa forma, um número maior de empresas passe a desenvolver componentes de *workflow*, expandindo e dinamizando este mercado [WfMC 98a].

3.2 Modelo de Referência

De forma a estabelecer padrões para a área de *workflow*, a WfMC desenvolveu um *modelo de referência*. Esse modelo de referência identifica os componentes que comunicam-se entre si, juntamente com as respectivas interfaces e formatos de intercâmbio necessários para atingir-se a interoperabilidade desejada.

O modelo de referência, apresentado na figura 3.1, define cinco interfaces entre componentes, além de uma interface sobre o sistema de gerência de *workflow*, denominada WAPI (*Workflow API and Interchange Formats*). Esta interface consiste em uma série de construções pelas quais os serviços de gerência de *workflow* podem ser utilizados. Desta forma, os serviços de *workflow* podem ser implementados de diferentes formas, contanto que sejam oferecidas interfaces que traduzam a implementação particular de cada produto de *workflow* para a interface padronizada pela WfMC.

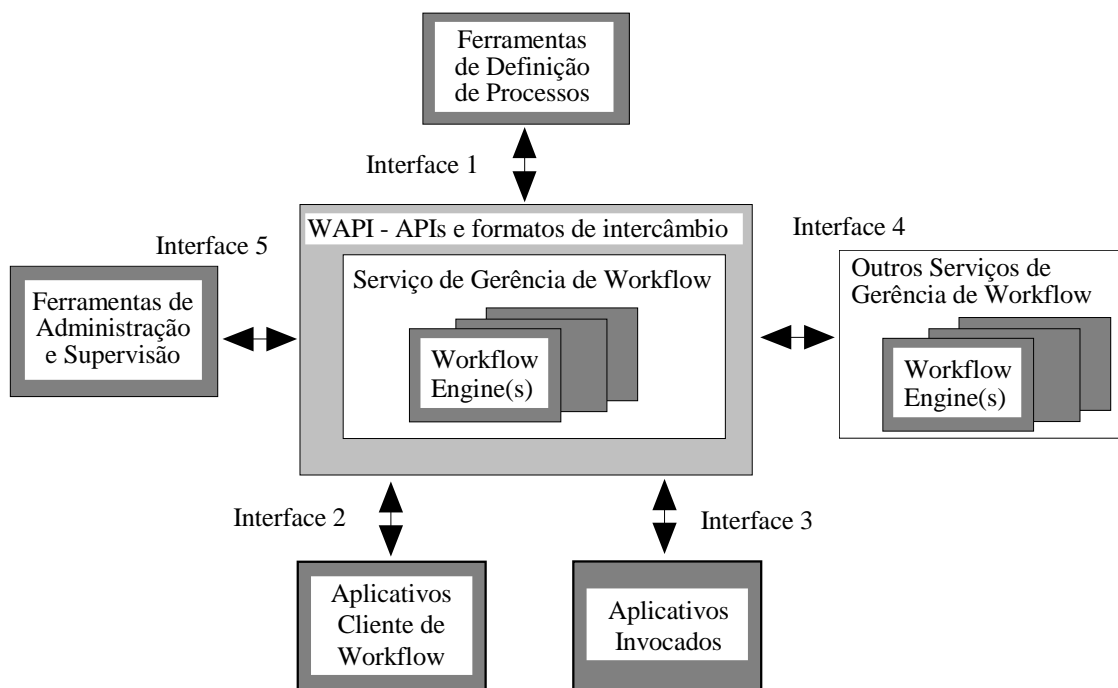


FIGURA 3.1 - Modelo de referência de *workflow* - Componentes e Interfaces

A interface 1 envolve a definição de processos, tendo sido proposta uma linguagem padrão para a definição de processos, a WPDL (*Workflow Process Definition Language*). Com essa linguagem, torna-se possível exportar uma definição de processo, criada em uma certa ferramenta de modelagem, para qualquer WFMS que suporte este padrão [WfMC 98b].

A interface 2 define uma API para que aplicativos cliente de *workflow* possam invocar serviços de gerência de *workflow* de forma padrão. Assim, um aplicativo construído por um terceiro pode utilizar os serviços de gerência de *workflow* de qualquer WFMS que suporte este padrão. A interface 2 inclui, atualmente, chamadas a serviços como recuperação de itens de trabalho e declaração de início e término de atividades. A WfMC publicou uma API em linguagem C cobrindo estas classes de chamadas, a qual encontra-se em [WMC 97].

A interface 3 trata da comunicação com aplicativos externos, invocados pelo serviço de gerência de *workflow* para a realização de atividades específicas [WMC 94]. Por exemplo, uma atividade pode necessitar fazer acesso a um sistema de gerência de banco de dados, ou a um sistema de correio eletrônico. Essa interface padronizaria a interface com outros sistemas, envolvendo aspectos como a passagem e o retorno de parâmetros, além de serviços de notificação. É importante observar que, até a conclusão desta dissertação de mestrado, nenhuma versão (nem sequer preliminar) dessa interface havia sido disponibilizada ao público.

A interface 4 trata da comunicação entre diversos sistemas de gerência de *workflow*, quando envolvidos na administração de partes de um mesmo processo. Com esta interface, torna-se possível a execução de um processo através de vários WFMS diferentes. Por exemplo, várias organizações poderiam fazer parte de um único processo, aumentando ainda mais o grau de coordenação entre si, e levando os benefícios da tecnologia de *workflow* para o âmbito inter-organizacional. A WfMC publicou dois documentos a respeito desta interface: [WMC 96b], que define abstratamente os mecanismos de integração, e [WMC 96c], que propõe mecanismos de integração baseados no Internet MIME.

Por final, a interface 5 envolve ferramentas de administração e monitoramento dos processos executados. Essa interface especifica que dados devem ser gerados pelo WFMS durante a execução dos processos, para que ferramentas possam consultá-los e, possivelmente, realizar análises [WMC 96d].

3.3 Padrões para Definição de Processos

A interface 1 foi concebida para permitir que diferentes ferramentas de modelagem de *workflow* e sistemas de gerência de *workflow* pudessem trocar informações entre si. Desta forma, torna-se possível modelar um processo em uma ferramenta qualquer que atenda a este padrão, e logo após implementá-lo em qualquer WFMS que também atenda ao padrão. Assim, um dos maiores problemas atualmente enfrentados pela área de sistemas de *workflow* - ressaltado por [AMA 97b] e [BAI 93] -, a incompatibilidade entre os produtos de modelagem e os produtos de gerência de *workflow*, poderia ser solucionado.

Os padrões da WfMC para definição de processos compõem-se de dois elementos, definidos em [WMC 98b], os quais serão detalhados nas subseções a seguir.

- um meta-modelo, contendo os conceitos que devem ser suportados por um sistema de *workflow*;
- uma linguagem padrão para definição de processos, a WPDL (*Workflow Process Definition Language*).

É fundamental ressaltar, no entanto, que a interface 1 ainda não teve sua versão final aprovada pela WfMC, permanecendo sua especificação, ainda, como “em trabalho”. Desta forma, ainda que a interface 1 já venha sendo definida desde 1995, e tenha avançado consideravelmente, é possível que alguns de seus aspectos abordados neste trabalho venham a sofrer modificações, levando, possivelmente, a alterações na proposta de integração aqui apresentada.

3.3.1 Meta-modelo

O meta-modelo definido pela WfMC, ilustrado na figura 3.2, na notação UML, apresenta os conceitos de um sistema de *workflow* e seus respectivos relacionamentos. Além disso, a WfMC definiu, em [WfMC 98b], um conjunto de atributos para cada entidade do meta-modelo. De modo geral, este meta-modelo apresenta conceitos já consagrados da área de *workflow*, como definição de processo (classe *Process Definition*), atividade (classe *Activity*), transição - também denominada por outros autores como *gatilho* ou *rota* - (classe *Transition*) e participante - também denominado por outros autores como *ator* ou *agente* - (classe *Participant*). Os aspectos mais interessantes desse meta-modelo estão nos conceitos de *definição de aplicativo invocado* (classe *Application Definition*) e de *dados relevantes do processo* (classe *Workflow Relevant Data*). Um aplicativo invocado define uma determinada ferramenta de software que será invocada, em tempo de execução, para a realização de uma determinada atividade. Os dados relevantes do processo são, por sua vez, variáveis definidas em tempo de modelagem, que podem, em tempo de execução, ser consultadas e/ou receberem valores, tanto através do próprio WFMS como através de aplicativos cliente. Para esses últimos, a interface 2 oferece funções para consulta e atribuição de valores aos dados relevantes do processo.

Dados relevantes do processo podem ser utilizados em regras, definindo o roteamento de um processo. Por exemplo, uma atividade de revisão de um documento pode possuir uma variável do tipo lógico *Aprovado*. Caso a variável receba um valor *verdadeiro*, o processo seria direcionado, digamos, para uma atividade de aprovação final; caso contrário, o processo seria direcionado novamente para a atividade anterior, de criação do documento.

Além disso, dados relevantes do processo podem ser utilizados para adicionar atributos específicos a um determinado processo ou atividade. Por exemplo, pode-se definir um atributo *prioridade* para um processo, e, posteriormente, em tempo de execução, consultar e/ou atribuir valor a esse atributo.

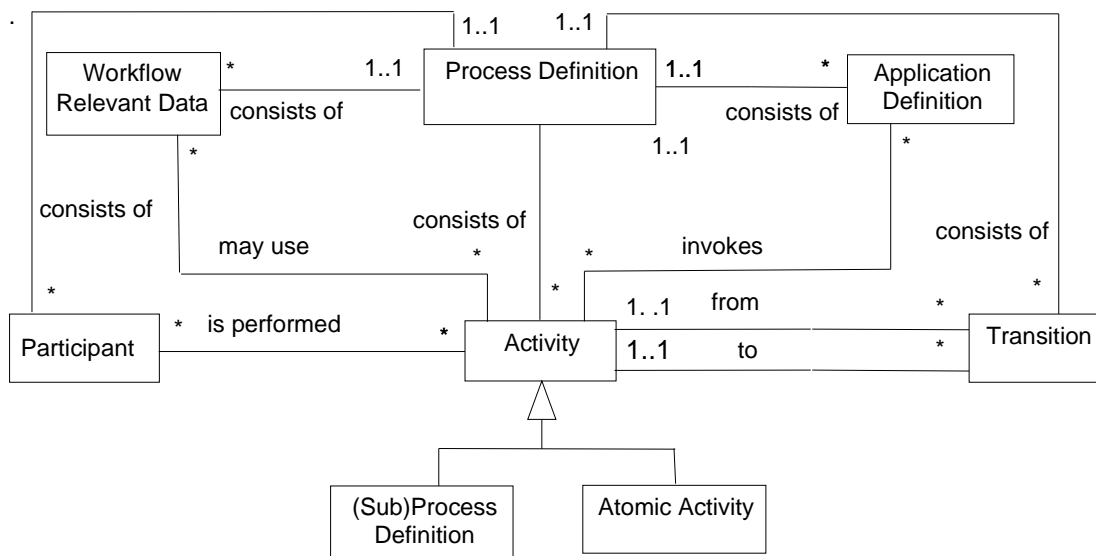


FIGURA 3.2 - Meta-modelo proposto pela WfMC

A tabela 3.1 relaciona as principais entidades do meta-modelo, com uma descrição de sua função e alguns de seus principais atributos.

TABELA 3.1 - Visão geral das principais entidades do meta-modelo da WfMC

Classe	Função	Atributos Principais
Process Definition	Identifica o processo	<ul style="list-style-type: none"> nome do processo descrição
Activity	Identifica as diversas atividades que compõem o processo	<ul style="list-style-type: none"> participantes que a executam aplicativos invocados pré-condições/pós-condições
Participant	Define as pessoas e/ou os papéis que executam o processo	<ul style="list-style-type: none"> nome estratégia de alocação de trabalho
Transition	Representa os relacionamentos de dependência entre as atividades	<ul style="list-style-type: none"> atividades predecessoras atividades sucessoras regras para a transição.
Application Definition	Descreve as ferramentas disponíveis para serem invocadas pelo WFMS	<ul style="list-style-type: none"> aplicativo parâmetros de entrada e saída
Workflow Relevant Data	Atributo do processo, podendo ser utilizado por atividades, transições ou aplicativos invocados	<ul style="list-style-type: none"> tipo de dado e valor

3.3.2 WPDL

As entidades do meta-modelo, juntamente com seus respectivos atributos, podem ser definidas através de um conjunto de comandos da WPDL. A WfMC definiu uma

gramática para essa linguagem, descrita em [WMC 98b]. Cada uma das entidades é representada por uma cláusula da linguagem. A tabela 3.2 ilustra a relação entre as entidades e as cláusulas da linguagem. As reticências (...) simbolizam os atributos de cada entidade, aqui omitidos.

TABELA 3.2 - Relação entre as entidades do meta-modelo e as cláusulas da WPD

Entidade	Cláusula WPD
Process Definition	WORKFLOW <nome do processo> (...) END_WORKFLOW
Activity	ACTIVITY <nome da atividade> (...) END_ACTIVITY
Participant	PARTICIPANT <nome do participante> (...) END_PARTICIPANT
Transition	TRANSITION FROM <atividade origem> TO <atividade destino> (...) END_TRANSITION
Invoked Application	APPLICATION <nome do aplicativo> (...) END_APPLICATION
<i>Workflow</i> Relevant Data	DATA <nome do dado> (...) END_DATA

A WfMC define, em [WMC 98b], os requisitos e passos para a utilização da WPD. Para atender ao padrão, cada ferramenta de modelagem de *workflow* precisa oferecer uma opção de exportação dos modelos nela gerados para a WPD. Essa operação deve gerar um arquivo ASCII (*script*), contendo as cláusulas WPD. Esse arquivo, então, deve ser submetido ao WFMS desejado. É necessário, evidentemente, que o WFMS ofereça alguma forma para a importação do arquivo. Ao ser importado pelo WFMS, o arquivo é, internamente, convertido para a representação interna proprietária do produto. A figura 3.3 apresenta esquematicamente essa relação de passos.

É importante ressaltar que a WPD foi criada para ser um formato de intercâmbio entre diversas linguagens de definição de *workflow* distintas. Apesar de ser possível especificar-se sistemas de *workflow* utilizando-a diretamente, esta não é a diretiva traçada pela WfMC e, aparentemente, também não pela indústria, ao menos a curto e médio prazo. Com a crescente assimilação dos padrões da WfMC pelos desenvolvedores de WFMS e de ferramentas de modelagem, é possível que, a longo prazo, a WPD (muito provavelmente acompanhada de uma representação visual)

torne-se um modelo conceitual padrão *de facto* para a indústria de *workflow* [AMA 97b].

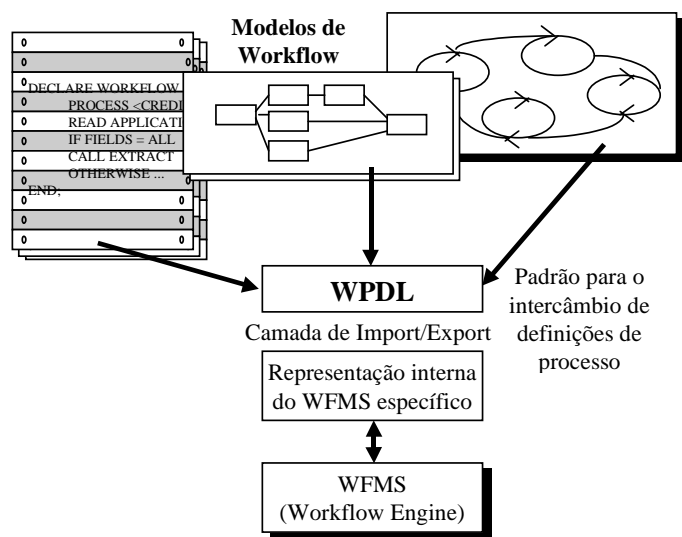


FIGURA 3.3 - Processo de exportação/importação de *scripts* WPDL

3.4 Padrões para Acesso aos Serviços de *Workflow*

A interface 2 constitui-se de uma API para que aplicativos cliente possam utilizar serviços de gerência de *workflow* de forma padronizada. Atualmente, esta API está disponível em linguagem C, sendo que a WfMC vem preparando versões para IDL (Interface Definition Language), da OMG, e para OLE, da Microsoft. É importante observar que a interface 2, ao contrário da interface 1, já teve sua especificação final aprovada pela WfMC, garantindo, dessa forma, a estabilidade da proposta de integração apresentada nesse trabalho, nos aspectos que envolvem a utilização desta interface.

Os recursos oferecidos pela interface 2 podem ser agrupados em funções de [WMC 97]:

- conexão ao sistema, responsáveis pela conexão e pela desconexão ao WFMS;
- controle de processo, responsáveis pelo disparo de processos e pela manutenção de seus atributos;
- controle de atividade, responsáveis pela manutenção de seus atributos;
- status de processo, responsáveis pela consulta ao status dos processos atualmente em execução;
- status de atividade, responsáveis pela consulta ao status das atividades atualmente em execução;
- lista de trabalho, responsáveis pela obtenção da lista de trabalho, e pelo disparo, encerramento e delegação de itens de trabalho;
- administração, como o encerramento coletivo de processos.

Evidentemente, não cabe aqui descrever esta API por completo, dada sua significativa extensão (56 chamadas atualmente definidas). No entanto, a fim de tornar mais clara a implementação apresentada no capítulo 7, são listadas, na tabela 3.3, algumas das principais chamadas, agrupadas por suas respectivas áreas.

TABELA 3.3 - Principais funções da interface 2 da WfMC

Grupo	Funções Definidas	Finalidade
Conexão ao Sistema	WMConnect	conecta ao WFMS
	WMDisconnect	desconecta do WFMS
Controle do Processo	WMOpenProcessDefinitionList	recuperam as definições de processo que podem ser iniciadas pelo participante
	WMFetchProcessDefinition	
	WMCloseProcessDefinitionList	
	WMCreateProcessInstance	disparam uma nova instância de processo
	WMStartProcess	
	WMGetProcessInstanceAttributeValue	consulta um atributo do processo
WMAssignProcessInstanceAttribute	define o valor de um atributo do processo	
Lista de Trabalho	WMOpenWorkList	recuperam os itens de trabalho designados para o participante
	WMFetchWorkItem	
	WMCloseWorkList	
	WMGetWorkItem	inicia um item de trabalho
	WMCompleteWorkItem	declara o término de um item de trabalho
	WMReassignWorkItem	delega um item de trabalho
	WMGetWorkItemAttributeValue	consulta um atributo do item de trabalho
	WMAssignWorkItemAttribute	define o valor de um atributo do item de trabalho

É interessante ressaltar que as funções *WMGetProcessInstanceAttributeValue* e *WMAssignProcessInstanceAttribute* tratam de atributos dos processos. Esses atributos devem ter sido, em tempo de modelagem, definidos como *dados relevantes do processo*, através da cláusula DATA, como mostrado na tabela 3.2. Similarmente, as funções *WMGetWorkItemAttributeValue* e *WMAssignWorkItemAttribute* tratam dos atributos de itens de trabalho, os quais também devem ter sido definidos através de cláusulas DATA.

É importante, ainda, ressaltar que a interface 2 não define, atualmente, qualquer forma de comunicação entre a aplicação cliente e o sistema de gerência de *workflow*. Não são especificados, por exemplo, formas de localizar o WFMS, tampouco a tecnologia de comunicação, como *sockets* ou RPC. Dessa forma, as implementações

atualmente disponíveis da interface 2 realizam esta comunicação de forma proprietária, mantendo *drivers* específicos do WFMS junto à aplicação cliente. Esta forma é, por exemplo, similar à utilizada na implementação do padrão ODBC, da Microsoft, para conexão a bancos de dados.

4 Padrões para Sistemas de Gerência de Documentos

Neste capítulo será apresentado o padrão para sistemas de gerência de documentos, denominado DMA, definido pela *Association for Information & Image Management* (AIIM). Inicialmente, serão apresentadas as motivações e objetivos desse esforço de padronização. Após, serão relacionados os conceitos fundamentais utilizados pelo padrão. A seguir, serão apresentados a arquitetura de componentes e o modelo de integração definidos pela AIIM. Por final, serão detalhados dois aspectos essenciais deste padrão: o seu modelo de documentos e o seu modelo de versões, com a apresentação de seus modelos de objetos e principais métodos. É importante destacar que não serão abordados aqui todos os aspectos deste padrão, uma vez que trata-se de uma especificação bastante volumosa (atualmente em torno de 500 páginas). Logo, serão focados especialmente os aspectos do padrão utilizados na proposta de integração, apresentada no capítulo 5.

4.1 Motivações e Objetivos

É reconhecido que os documentos são um elemento fundamental em todas as organizações, por registrarem informações, decisões e atividades por elas executadas [SAD 97]. Devido a essa importância, a preservação, consulta e controle de acesso aos documentos são atividades essenciais a praticamente qualquer organização. Conforme [DAL 95] e [KOU 97], a agilidade atualmente exigida das organizações as obriga a ações e a tomadas de decisão rápidas e de alta qualidade, para as quais é imprescindível o acesso à maior quantidade de informações possível, as quais estão, via de regra, sob a forma de documentos.

Encontra-se, atualmente, um vasto número de aplicações relacionadas a documentos, envolvendo aspectos como sua criação, armazenamento e recuperação. Entre estas, incluem-se processadores de textos, softwares de digitalização e sistemas de gerência de documentos. Essa diversidade, no entanto, ocasiona uma grande incompatibilidade entre os sistemas atuais. Por exemplo, um processador de textos não é capaz de buscar documentos em um sistema de gerência de documentos, a menos que tenha sido programado para utilizar a API específica daquele produto. Similarmente, não é possível realizar pesquisas simultaneamente através de sistemas de gerência de documentos desenvolvidos por empresas diferentes. No contexto atual, onde proliferam a diversidade de plataformas, a interconectividade e os sistemas distribuídos, tais restrições são extremamente graves [AII 98a].

As empresas fornecedoras de aplicativos para gerência de documentos vêm, de forma crescente, conscientizando-se que, desta forma, os ganhos de eficiência possíveis com a utilização de sistemas de gerência de documentos ficam bastante prejudicados. De modo a apresentar uma resposta a essas exigências, a AIIM, consórcio que reúne um expressivo número destas empresas, criou uma força-tarefa para desenvolver um padrão para sistemas de gerência de documentos. Este padrão possui os seguintes objetivos [AII 98a]:

- permitir um acesso uniforme a documentos, armazenados em sistemas de gerência de documentos, independentemente de plataforma, tecnologia de rede ou formato dos documentos;
- manter a integridade dos documentos armazenados, permitindo que regras sejam definidas, e controlando a mídia e os softwares relacionados a cada documento;
- enquadrar as coleções de documentos dentro dos propósitos/interesses da organização, permitindo a definição de políticas de acesso específicas para cada uma delas;
- expandir o uso colaborativo de documentos, através do oferecimento de serviços de compartilhamento dos documentos, inclusive em tempo real.

A AIIM espera que todos esses pontos sejam cobertos pela especificação DMA ao longo do tempo. A versão 1.0 da especificação, divulgada em dezembro de 1997, foca principalmente a interoperabilidade entre os serviços e aplicativos.

4.2 Conceitos Fundamentais

O modelo DMA define diversos conceitos a respeito de como os documentos são armazenados, pesquisados e recuperados. Um conceito fundamental é o de *espaço de documentos (document space)*. Um espaço de documentos é, no modelo DMA, uma coleção de documentos, mantida sob a responsabilidade de um sistema de gerência de documentos, o qual emprega uma determinada tecnologia e define políticas de utilização. Uma organização pode, por exemplo, possuir um espaço de documentos onde são armazenadas imagens de todas as faturas recebidas, em um sistema de gerência de documentos baseado em discos ópticos, assim como possuir um outro espaço de documentos que armazene documentos confidenciais da alta gerência, como planos estratégicos e projeções de mercado, em um sistema de gerência de documentos baseado em bancos de dados relacionais. Essas duas bases distintas podem corresponder a espaços de documentos distintos, pois diferem em tecnologia e em controles de acesso.

Outro conceito importante é o de *sistema de documentos (document system)*. Um sistema de documentos representa um agrupamento de diversos espaços de documentos, possivelmente heterogêneos em suas respectivas tecnologias e políticas. A única condição existente sobre os espaços de documentos agrupados é que eles possam ser acessados a partir de um sistema de documentos comum. O sistema de documentos pode ser visto, assim, como uma porta de entrada a diversos espaços de documentos. Tipicamente, uma organização possuirá um único sistema de documentos, o qual englobará todos os possíveis espaços de documentos nela existentes, que podem diferir significativamente em tecnologia, propósito e políticas de utilização. Pode-se afirmar, então, que o modelo DMA encara um sistema de documentos como uma *federação* de sistemas de gerência de documentos, permitindo um elevado grau de independência entre eles.

Os documentos são armazenados, como já afirmado, dentro dos espaços de documentos. O conceito de *documento*, no modelo DMA, abrange qualquer agrupamento de informações relacionadas, independente de seu formato ou de seu grau de estruturação [AII 97]; textos, gráficos, imagens digitalizadas, arquivos de vídeo e som são exemplos de documentos DMA.

Um conceito vastamente utilizado no modelo DMA é o conceito de *propriedade* (*property*). Uma propriedade representa um atributo de um objeto DMA. O modelo DMA oferece diversos métodos para a manipulação de propriedades. O conteúdo de um documento, por exemplo, está armazenado em uma propriedade do objeto documento.

A figura 4.1 ilustra o modelo conceitual da DMA, conforme [AII 97]. Foram utilizados os termos originais DMA, em inglês.

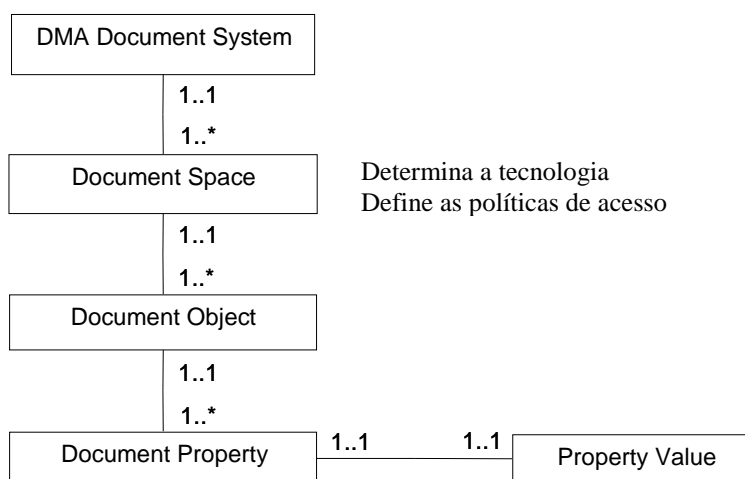


FIGURA 4.1 - Modelo conceitual da DMA

Uma das inspirações deste modelo conceitual é a analogia com sistemas de bibliotecas. Conforme [AII 97], essa semelhança pretende tornar a compreensão e operação de sistemas DMA a mais fácil e intuitiva possível. Assim, um sistema de documentos representa um conjunto de bibliotecas; cada espaço de documentos corresponde a uma única biblioteca. Os documentos são elementos que podem ser emprestados por uma biblioteca, como livros e revistas. As propriedades de cada documento, como data de criação e formato, equivalem aos atributos dos elementos da biblioteca, como data de publicação e editora.

Os passos para obter-se um determinado documento são, também, inspirados nos mecanismos de uma biblioteca. Assim como em um sistema de bibliotecas é preciso inicialmente decidir em qual das bibliotecas buscar o documento, no modelo DMA é preciso primeiramente decidir em qual espaço de documentos é mais provável que exista o documento. Para facilitar essa decisão, pesquisas entre diversos espaços de documentos são possíveis. Uma vez localizado o documento desejado, a aplicação cliente de gerência de documentos pode requisitar o próprio documento, ou informações adicionais sobre ele, ao espaço de documentos, de forma similar ao usuário de uma

biblioteca que solicita um empréstimo de um livro, ou maiores informações sobre ele, a um bibliotecário. Cabe ao espaço de documentos definir e exercer as políticas de acesso e segurança aos documentos, assim como cabe ao bibliotecário definir as normas da biblioteca.

4.3 Arquitetura DMA

O padrão DMA define uma arquitetura genérica para a operação dos sistemas computacionais a ele aderentes. Esta arquitetura é baseada no paradigma cliente/servidor [REN 94], envolvendo três componentes básicos de software: os *aplicativos cliente DMA*, a *camada de coordenação DMA* e os *elementos de serviço* – sistemas de gerência de documentos, que efetivamente armazenam os documentos. A figura 4.2 mostra esquematicamente esta relação.

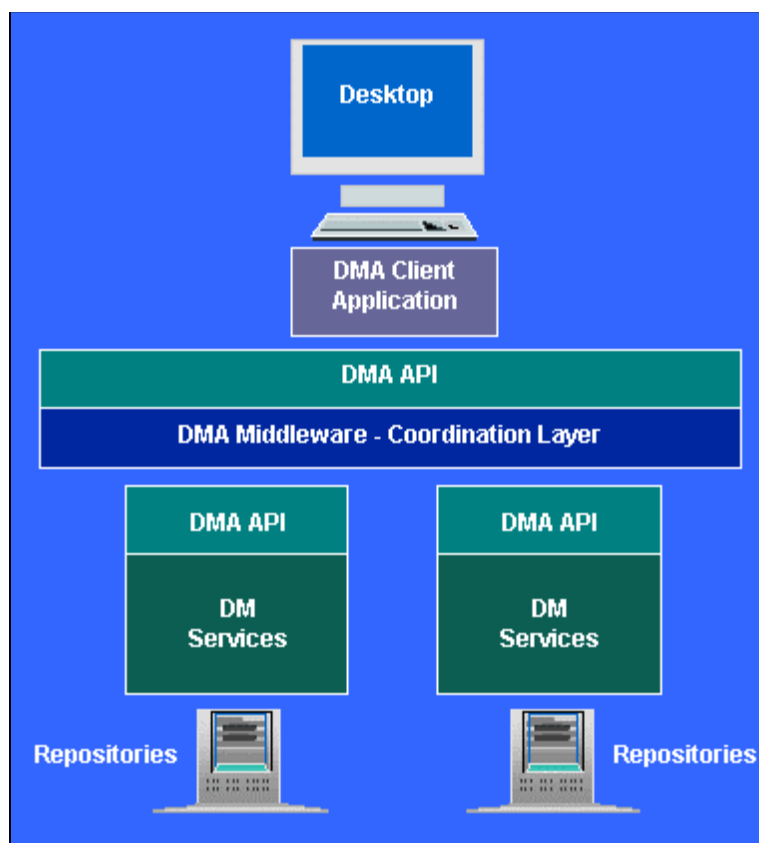


FIGURA 4.2 - Arquitetura DMA

A comunicação entre os aplicativos cliente e a camada DMA é realizada através de uma API, composta por um conjunto de objetos COM. A AIIM utiliza a expressão *ponto de acesso* para designar nodos onde esta API esteja disponível. Similarmente, a comunicação entre os elementos de serviço e a camada DMA é realizada através de uma SPI, também composta por um conjunto de objetos COM. A AIIM utiliza a expressão *ponto de serviço* para designar os nodos onde esta SPI esteja disponível. O conjunto de pontos de acesso e pontos de presença forma os *pontos de presença DMA*.

A camada DMA, juntamente com os pontos de acesso e serviço, compõe o *middleware* DMA. Esse *middleware* interliga aplicativos cliente aos DMS, permitindo que estes softwares possam residir em quaisquer nodos de uma rede, utilizando a mesma plataforma ou plataformas diferentes. O aplicativo cliente submete todas suas requisições ao *middleware*, o qual, por sua vez, comunica-se com o DMS correspondente, obtendo a resposta e repassando-a ao aplicativo. Assim, do ponto de vista do aplicativo cliente, a arquitetura DMA oferece um acesso uniforme a sistemas de gerência de documentos, independentemente de fornecedor, plataforma ou rede. É importante observar que essa arquitetura é significativamente mais sofisticada que a definida pela WfMC, a qual não define um *middleware* para a comunicação entre clientes e servidores.

Deve-se fazer, no entanto, uma ressalva quanto a esta arquitetura. Pode-se perceber facilmente que a comunicação entre os clientes e a camada DMA, assim como entre a camada DMA e os servidores, exige a utilização de alguma tecnologia de comunicação cliente/servidor, como RPC [ORF 96], DCOM [ORF 96] ou CORBA [SIE 96]. No entanto, a especificação DMA 1.0 é omissa a respeito da definição da tecnologia a ser empregada, deixando esta decisão a cargo dos desenvolvedores de software aderente à DMA. No presente momento, por exemplo, diversas implementações, como a da FileNet, mantém o *middleware* DMA residente no cliente, realizando a comunicação através de suas próprias interfaces proprietárias [FAY 98].

4.4 Modelo de Integração

O padrão DMA define um modelo de integração, que determina como os componentes de software (objetos COM) que implementam os conceitos apresentados na seção 4.2 são construídos e acessados, e como interagem entre si. A tabela 4.1 enumera esses componentes e suas respectivas atribuições.

TABELA 4.1 - Principais componentes do padrão DMA

Classe	Função
System Manager	objeto inicial do sistema; permite que objetos System sejam acessados a partir dele
System	representa um sistema de documentos; permite que objetos DocSpace sejam acessados a partir dele
DocSpace	representa um espaço de documentos; permite que objetos DocVersion sejam acessados a partir dele
DocVersion	representa uma determinada instância de um documento

O objeto inicial *System Manager* está implementado na própria camada de coordenação DMA. Para obtê-lo, é feita uma chamada à função *C dmaConnectSystemManager*. Essa constitui-se na única chamada não-COM da API DMA. Nesse momento, pode-se estabelecer uma conexão com um determinado sistema de documentos (objeto *System*) através de uma chamada ao método *ConnectSystem* da interface *IdmaSystemManager*. Então, pode-se conectar com um determinado espaço de

documentos (objeto *DocSpace*) com uma chamada ao método *ConnectDocSpace* da interface *IdmaSystem*. Pode-se, então, obter um determinado documento (objeto *DocVersion*) através de uma invocação do método *ConnectObject* da interface *IdmaDocSpace*. [AII 97] oferece maiores detalhes sobre estas interações, enumerando detalhadamente todos os parâmetros utilizados nas chamadas.

No modelo DMA, todos os objetos persistentes, como documentos (objetos *DocVersion*) possuem um identificador, denominado *Object Instance Identifier* (OIID). A sintaxe de um OIID é bastante similar à de uma URL da Internet. Conhecendo o OIID de um objeto persistente DMA, é possível recuperá-lo através do método *ConnectObject* da interface *IdmaDocSpace*. A estrutura geral de um OIID é mostrada na figura 4.3.

dma://[<dma pop>]/<system id>/<docspace id>/<object id>, onde:

<dma pop> é o nome do host do ponto de presença DMA;
 <system id> é o identificador do objeto System;
 <docspace id> é o identificador do objeto DocSpace;
 <object id> é o identificador (interno) do objeto persistente

FIGURA 4.3 - Estrutura geral de um OIID

4.5 Modelo de Conteúdo

O modelo de conteúdo do padrão DMA define objetos, interfaces e propriedades relacionados à criação, consulta, modificação e remoção de conteúdo dos documentos gerenciados. Este modelo é consideravelmente poderoso, permitindo a representação de documentos complexos, compostos por diversas apresentações, podendo cada uma destas ser composta por diversos componentes. A tabela 4.2 relaciona as principais classes envolvidas e suas respectivas atribuições.

TABELA 4.2 - Componentes do modelo de conteúdo DMA

Classe	Função
DocVersion	representa uma versão de um documento, armazenando suas propriedades descritivas e referenciando suas apresentações (objetos Rendition).
Rendition	representa uma determinada <i>apresentação</i> de uma versão de um documento. As apresentações diferem em propósito e em formato de dados, ainda que possuam o mesmo conteúdo. Assim, um documento pode ter uma apresentação como arquivo HTML, para consulta, outra como arquivo PostScript, para impressão, e uma terceira como arquivo Microsoft Word, para edição.
Content Element	representa um determinado componente do documento, possuindo métodos para recuperar o conteúdo efetivo do documento. O padrão DMA define duas subclasses de Content Element: <i>ContentTransfer</i> , utilizada quando o DMS realmente armazena o conteúdo do documento, e <i>ContentReference</i> , quando o DMS apenas referencia um arquivo externo que possui o conteúdo do documento.

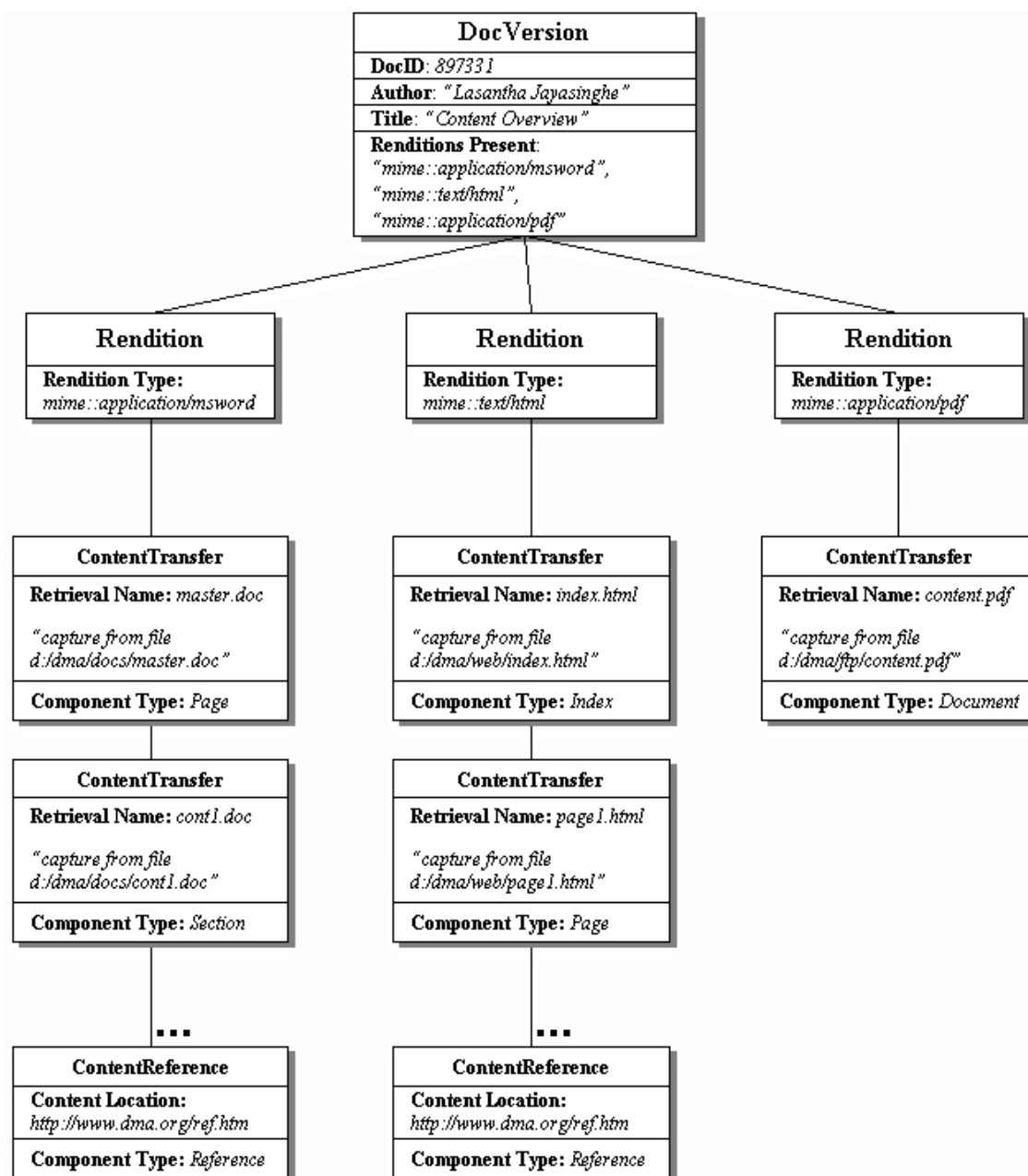


FIGURA 4.4 - Exemplo de modelo de documento segundo o padrão DMA

A figura 4.4 mostra um exemplo, extraído de [AII 97], de utilização deste conjunto de classes. Neste exemplo, o documento é composto por três apresentações: uma em Microsoft Word, outra em HTML e uma terceira em PDF. A apresentação em Word é composta por três arquivos, assim como a em HTML, enquanto a em PDF possui um único arquivo. É importante, ainda, observar que o conteúdo do documento só é acessível através da classe *Content Element* (ou uma de suas subclasses).

4.6 Modelo de Versionamento

O modelo de versões do padrão DMA define objetos, interfaces e propriedades relacionados à gerência de versões de documentos, permitindo que a evolução do

documento ao longo do tempo seja registrada e consultada. O modelo de versões suportado pela DMA 1.0 é um modelo de versões *linear*. Neste modelo, as versões de um documento são mantidas como uma série ordenada de objetos de versão de documento (objetos *DocVersion*), havendo somente uma versão corrente para o documento em um determinado instante do tempo. Outros modelos de versões, como o modelo com alternativas (*branch versioning*), devem ser completamente suportados em um futuro próximo, uma vez que o padrão já define algumas classes, como a classes *Configuration History*, especificamente para estes modelos.

A tabela 4.3 relaciona as principais classes envolvidas e suas respectivas atribuições.

TABELA 4.3 - Componentes do modelo de versões DMA

Classe	Função
Configuration History	representa a entidade que está sendo versionada; é o ponto de entrada para todas as possíveis configurações do objeto versionado
Version Series	representa uma determinada seqüência de versões que reflete a evolução do documento; em um modelo linear, há somente um objeto Version Series para cada documento versionado.
Versionable	representa uma determinada versão de um documento versionado. A classe DocVersion, apresentada na seção 4.5, é uma subclasse da classe Versionable.
Version Description	representa a associação entre uma determinada seqüência de versões (objeto Version Series) e uma determinada versão de um documento (objeto Versionable), indicando que este objeto DocVersion está presente naquele objeto Version Series; em um modelo linear, essa associação é sempre 1:n, mas em outros modelos ela pode diferir.
Reservation	representa a existência de uma <i>reserva</i> de um objeto Versionable para tornar-se a próxima versão em uma determinada Version Series; isto indica que foi invocado, previamente, o método de <i>check-out</i> sobre essa Version Series, reservando este direito ao aplicativo que o invocou.

Na figura 4.5 é apresentado o modelo de objetos do modelo de versões DMA, utilizando a notação UML. Observe-se que esse modelo é mais abrangente do que o necessário para representar um modelo de versões linear. Em um modelo linear, há, para um determinado documento versionado, somente um objeto *Configuration History* e um objeto *Version Series*. Similarmente, há apenas um objeto *Version Description* para cada objeto *Versionable*.

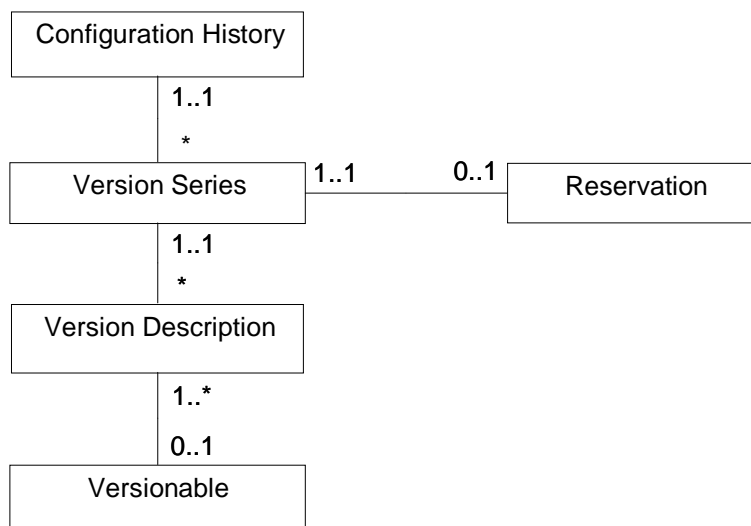


FIGURA 4.5 - Modelo de versões DMA

O modelo DMA define, ainda, as operações possíveis para a gerência de versões. Os métodos *SetReserveNext* e *SetCheckOutNext* permitem fazer uma operação de *check-out* sobre o documento, reservando ao aplicativo que os chamou o direito de criar a próxima versão deste documento. Esta nova versão pode ser criada, posteriormente, através da invocação do método *SetCheckIn*, ou pode-se abdicar deste direito através da invocação do método *SetRevoke*. No modelo DMA, a operação de *check-out* bloqueia o acesso ao documento para modificação, o qual só é liberado após o *check-in* ou o cancelamento da reserva.

5 Uma Proposta de Arquitetura Aberta

Neste capítulo será apresentada uma proposta de uma arquitetura aberta para a integração entre sistemas de gerência de documentos e sistemas de gerência de *workflow*. Inicialmente, será exposta a visão de sistemas de informação que norteia esta proposta. Em seqüência, serão expostos os requisitos necessários identificados para efetivar essa integração, e as ações a serem tomadas para tal. Os dois capítulos seguintes as apresentarão em detalhes.

5.1 Visão

Como exposto nos capítulos anteriores, o principal problema das técnicas atualmente empregadas para a integração entre sistemas de gerência de documentos e sistemas de gerência de *workflow* é o de esta integração ser *proprietária*, isto é, baseada nas particularidades de um determinado produto de software. [MOW 95] enumera diversas desvantagens desta estratégia, como a dificuldade em estender o sistema, a grande dependência dos fornecedores e as escassas opções de escolha. [SWE 95] aponta o alto risco, decorrente dessa estratégia, de a organização usuária produzir diversas 'ilhas de informação' com tais sistemas, isto é, sistemas incapazes de comunicar-se e de interoperar entre si, gerando assim altos custos de manutenção à organização.

Nesse contexto, tornam-se claras as vantagens da adoção de padrões para tais sistemas. [MOW 95] relaciona vários benefícios do desenvolvimento de software baseado em padrões, como o aumento da interoperabilidade, a possibilidade de substituir determinados sistemas aderentes ao padrão por outros igualmente aderentes, consequentemente com um maior número de opções. Assim, através da utilização de padrões, é possível construir aplicativos independentes da implementação proprietária de um determinado fabricante de software.

A visão que norteia esse trabalho é justamente a de promover a integração entre sistemas de gerência de documentos e sistemas de gerência de *workflow* através da *utilização conjunta* de padrões específicos destas áreas, respectivamente o padrão DMA e o padrão WAPI, apresentados nos capítulos anteriores. Em linhas gerais, deseja-se conceber uma *arquitetura* para esta integração que seja baseada unicamente nesses padrões, sendo que cada produto - de gerência de documentos ou gerência de *workflow* - é então visto como uma "caixa-preta", sendo utilizado tão-somente através das interfaces padrão estabelecidas pelos consórcios AIIM e WfMC. Procedendo-se desta forma, esta arquitetura permitirá a integração entre qualquer produto de gerência de documentos e qualquer produto de gerência de *workflow*, desde que estes suportem, respectivamente, as interfaces padrão DMA e WAPI. Trata-se, portanto, de uma evolução bastante expressiva em relação às estratégias atuais de integração descritas nesse trabalho.

5.2 Estrutura da Solução

Uma vez definida a utilização dos padrões DMA e WAPI, é necessário especificar como esta integração será suportada por uma arquitetura de sistemas de informação. A nosso ver, emergem três aspectos necessários para uma integração efetiva:

- a existência de um modelo integrado de documentos e *workflow*;
- a integração entre os serviços de gerência de documentos e de gerência de *workflow*;
- a existência de aplicações cliente integradas.

As seções a seguir introduzirão cada um desses aspectos.

5.2.1 Modelo integrado de documentos e *workflow*

O primeiro aspecto - a existência de um modelo integrado entre documentos e *workflow* - é indispensável para representar adequadamente as conexões existentes entre a gerência de documentos e a gerência de *workflow*. Para tal, é preciso claramente especificar os pontos de inter-relacionamento entre a gerência de documentos e a gerência de *workflow*. Essa relação é descrita através de uma *integração conceitual* entre os modelos, discutida em profundidade na seção 6.1. Nessa integração, em particular, duas necessidades se sobressaem. A primeira é a necessidade de descrever quais documentos serão gerados ao longo do processo de *workflow*. Desta forma, poder-se-á, em tempo de execução, garantir-se esse aspecto de integridade do processo, ao mesmo tempo automatizando a criação dos documentos e evitando que documentos que não os desejados sejam gerados.

Uma segunda necessidade relaciona-se com as operações a serem realizadas sobre os documentos, como alterações, criação de versões e eliminação. É essencial que essas operações possam ser especificadas com um certo grau de detalhamento. Por exemplo, pode-se desejar que o participante que está realizando a atividade "comentar orçamento" não tenha permissão de alterar o documento de orçamento, podendo apenas adicionar comentários. Para tal, faz-se necessária a existência de recursos poderosos de modelagem, capazes de definir precisamente as atribuições de uma determinada atividade do processo em relação a um determinado documento.

No entanto, a grande maioria dos modelos e linguagens de especificação de *workflow*, como o modelo de Casati/Ceri [CAS 95], o modelo *Action Workflow* [MED 92] e o *Trigger Modeling* [JOO 94], não dão suporte a uma definição precisa das atividades modeladas, e tampouco a WPDL - a linguagem de especificação de *workflows* proposta pela WfMC [WMC 98b] - o oferece. A definição da atividade resume-se, na maior parte dos casos, a uma simples descrição textual da atividade, a qual não é analisada nem interpretada pelo sistema de gerência de *workflow*. Cabe integralmente ao participante que recebê-la, nesses casos, a responsabilidade de interpretar a tarefa e realizar ações de acordo com essa interpretação. Desta forma, tais linguagens e modelos apresentam-se insuficientes para a efetivação dos propósitos delineados.

Por sua vez, abordagens que buscam modelar o comportamento dinâmico de documentos, como o modelo Inquiry Cycle [TAK 95], e abordagens baseadas em statecharts, como [WOD 96], também não suprem as necessidades levantadas, pois não contemplam aspectos essenciais da modelagem de *workflow*, como a definição de papéis e a especificação de regras de roteamento [AMA 97b].

Dessa forma, observa-se que nenhum dos dois aspectos necessários para a conexão entre a gerência de documentos e a gerência de *workflow* é adequadamente suportado por modelos e linguagens de definição de *workflows* ou de documentos puros. Daí reside, então, a necessidade de definir novas linguagens, ou extensões a linguagens existentes, que incluam os tratamentos desejados. Para tal, foi estabelecido um mapeamento dos tratamentos desejados para os padrões DMA e WAPI. Assim, concebeu-se a linguagem WPD-L-DOC - uma extensão da WPD-L acrescida de relacionamentos para com documentos, e um conjunto de classes e métodos DMA a serem invocados. Este mapeamento será apresentado em detalhes na seção 6.2.

5.2.2 Integração de serviços

O segundo aspecto envolve a integração dos serviços de gerência de documentos oferecidos pelo padrão DMA com os serviços de gerência de *workflow* oferecidos pelo padrão WAPI. Uma vez que estes dois padrões são absolutamente ortogonais entre si, isto é, não possuem nenhum elemento em sua interseção, esta integração deve ser realizada através da construção de um módulo de software sobre esses padrões. Esse módulo de integração será denominado, neste trabalho, de *camada de serviços*. Com essa abordagem, o acesso aos produtos de DMS e WFMS, realizado através dos respectivos padrões, é unificado, e a própria existência de produtos diferentes é ocultada dos aplicativos cliente. A figura 5.1 mostra, esquematicamente, como os serviços se integrarão.

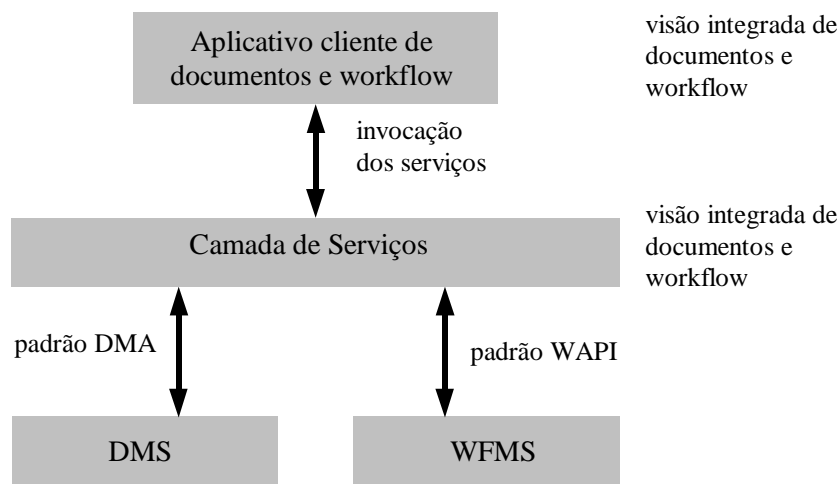


FIGURA 5.1 - Integração entre os serviços

Conforme ilustrado na figura 5.1, a camada de serviços oferece um conjunto de serviços aos aplicativos cliente. Esses serviços, por sua vez, serão implementados

através de chamadas aos serviços definidos pelos padrões DMA e WAPI. Assim, o serviço pode ser desempenhado por qualquer DMS aderente ao padrão DMA ou por qualquer WFMS aderente ao padrão WAPI.

A camada de serviços aqui apresentada, de acordo com a classificação proposta por [BER 96], constitui-se em um *middleware* do tipo *framework* sobre outros dois *serviços de middleware* - os padrões DMA e WAPI. A camada de serviços será explorada em detalhes na seção 7.1.

5.2.3 Integração dos aplicativos cliente

Por final, deve ser observado o aspecto da existência de aplicativos cliente integrados. Por aplicativo integrado entende-se, nesse contexto, um aplicativo que ofereça acesso aos serviços de gerência de documentos e de gerência de *workflow* de forma absolutamente transparente, com uma interface unificada e sem necessidade de, por exemplo, *logins* em sistemas separados. Conforme discutido no capítulo 2, este é um dos principais problemas das integrações proprietárias realizadas entre produtos de gerência de *workflow* e de gerência de documentos. Nesse trabalho, serão propostos e implementados aplicativos cliente integrados. A seção 7.2 apresentará em detalhes esses aplicativos.

6 Aspectos de Modelagem

Neste capítulo serão tratados aspectos no modelo integrado de documentos e *workflow*. Inicialmente, será promovida uma integração conceitual entre documentos e *workflow*, de acordo com os requisitos levantados no capítulo anterior. A seguir, serão apresentadas as estruturas utilizadas dos dois padrões - DMA e WAPI - a fim de suportar o modelo integrado definido, e como será a dinâmica deste mapeamento. Por final, com o propósito de bem ilustrar o mapeamento estabelecido, será apresentado um exemplo detalhado de um processo de *workflow* ligado a documentos, mostrando as respectivas estruturas DMA e WAPI geradas.

6.1 Integração Conceitual

Como passo inicial, em função das motivações apresentadas na seção 5.2.1, será estabelecida uma *integração de modelos conceituais*. Isto é, será definida uma relação entre os conceitos existentes em um sistema de gerência de documentos (tipo de documento e documento, entre outros) e os conceitos existentes em um sistema de gerência de *workflow* (processo, atividade e participante, entre outros).

A fim de conceber esta integração conceitual, foi realizado um estudo em diversas frentes. A primeira foi o estudo dos conceitos suportados pelos padrões WAPI (mais especificamente, pela linguagem WPDLE) e DMA. Esta atividade foi de vital importância, uma vez que o princípio dessa integração conceitual é de que todos os conceitos e relacionamentos identificados possam ser suportados dentro desses padrões. Ressalte-se que, no entanto, como comentado na seção 1.2, os padrões WAPI e DMA são omisso um em relação ao outro. Assim, somente um estudo desses padrões não é suficiente para estabelecer o relacionamento entre os conceitos.

Uma segunda frente de trabalho constituiu-se na avaliação dos conceitos utilizados por diversas ferramentas comerciais, tanto por sistemas de gerência de documentos quanto por sistemas de gerência de *workflow*. Nesses produtos, foram avaliados principalmente os recursos de integração entre documentos e *workflow* de que dispunham. Utilizando a classificação apresentada no capítulo 2, foram analisados, como produtos de DMS estendidos para *workflow*, o produto da Documentum (DocBase) [REA 97a] e o produto da Keyfile (Keyfile) [KEY 98]. Como integrações proprietárias entre produtos de gerência de documentos e produtos de gerência de *workflow*, foram avaliados, respectivamente, os DMS e WFMS da PC DOCS (DOCS Open) e Reach Software (WorkMAN); da PC DOCS (DOCS Open) e Action Technologies (Action Workflow). Ainda, foi avaliado o produto de gerência de *workflow* da Oracle (Oracle Workflow Cartridge) e o produto de gerência de documentos da FileNet (Panagon Mezzanine).

O modelo conceitual resultante é apresentado na figura 6.1, utilizando a notação UML [FOW 97]. A tabela 6.1 sumariza as principais características desse modelo

integrado, definindo cada uma das classes do modelo e indicando o produto ou padrão que tipicamente define tal conceito.

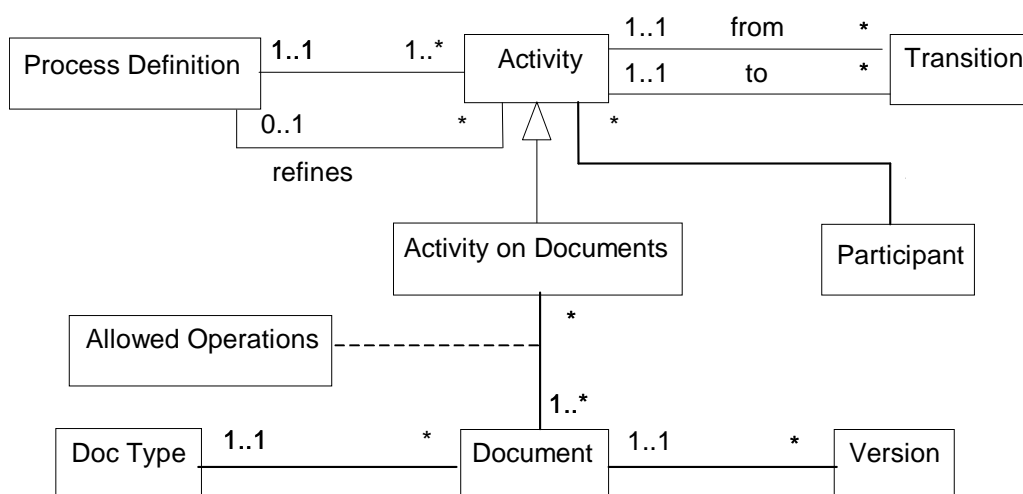


FIGURA 6.1 - Modelo conceitual integrado de documentos e *workflow*

TABELA 6.1 - Classes do modelo conceitual integrado

Classe	Função	Presente em
Process Definition	Representa um determinado processo de <i>workflow</i>	<ul style="list-style-type: none"> • WPDL • todos WFMS analisados
Activity	Representa uma atividade dentro de um processo de <i>workflow</i>	
Transition	Representa os relacionamentos de dependência entre as atividades	
Participant	Representa participantes e papéis que executam o processo	
Document	Representa um determinado documento	<ul style="list-style-type: none"> • DMA • todos DMS analisados
Version	Representa uma determinada versão de um documento	
Doc Type	Classifica os documentos de acordo com seu propósito e com o <i>template</i> utilizado	<ul style="list-style-type: none"> • maioria dos DMS analisados
Activity on Documents	Tipo de atividade que envolve a manipulação de um ou mais documentos	<ul style="list-style-type: none"> • DMS estendidos para <i>workflow</i>
Allowed Operations	Descreve as permissões da atividade sobre cada documento por ela manipulado	<ul style="list-style-type: none"> • Integrações proprietárias entre DMS e WFMS

Diversas observações devem ser feitas sobre o modelo de integração aqui apresentado. As seções a seguir detalharão cada uma dessas observações.

6.1.1 Classes de *Workflow*

Primeiramente, deve-se ressaltar que as quatro classes que descrevem aspectos puros de *workflow* (*Process Definition*, *Activity*, *Transition* e *Participant*) foram, assim como seus respectivos inter-relacionamentos, extraídas, sem modificações, do meta-modelo proposto pela WfMC [WMC 98b]. Desta forma, tem-se a garantia que estes conceitos poderão ser adequadamente suportados através dos padrões WfMC.

6.1.2 Classes *Document* e *Version*

Outro ponto importante a ressaltar é a natureza dos documentos manipulados. Os documentos, como concebidos nesse modelo, são documentos *não-estruturados*. Isto é, ao contrário de propostas como [FER 93] e [GRA 97], não é feita nenhuma definição, suposição ou controle sobre a estrutura lógica do documento, tampouco sobre sua estrutura de apresentação. A opção por documentos não-estruturados deveu-se a dois fatores: primeiramente, o padrão DMA não suporta atualmente documentos estruturados; em segundo lugar, poucos DMS comerciais o implementam, restringindo assim consideravelmente a sua aplicabilidade.

6.1.3 Classe *Doc Type*

Outro aspecto essencial deste modelo é a presença do conceito de tipo de documento, representado pela classe *Doc Type*. Esse conceito, ainda que não esteja incorporado ao padrão DMA, é utilizado por diversos produtos de gerência de documentos, como o Panagon Mezzanine [FIL 98b]. Além disso, um grande número de trabalhos acadêmicos, como [FER 93] e [GRA 97], também fazem uso desse conceito. No entanto, propostas como essas duas últimas versam sobre documentos estruturados – o que não é a finalidade da presente proposta – e, nelas, o tipo de documento serve basicamente para definir a estrutura lógica e de apresentação dos documentos daquele tipo. Nessa proposta, o tipo de documento não terá, portanto, tal responsabilidade, sendo utilizado para designar, basicamente, duas informações:

- designar a finalidade do documento, o que pode ser útil para fins de pesquisas entre documentos;
- definir um *template* para os documentos de cada tipo. Por *template* entende-se uma seqüência de bytes que representa uma pré-formatação aconselhada ou sugerida para os documentos desse tipo. *Templates* são amplamente utilizados em diversos sistemas e aplicativos ligados a documentos, como o DMS DOCS Open [PCD 98a] e o processador de texto Microsoft Word [KIN 97].

Assim, o tipo de documento ‘orçamento de projeto’, por exemplo, define uma categoria de documentos ‘orçamento de projeto’ – permitindo pesquisas por esse atributo - e, opcionalmente, um ‘template’ para todos os documentos do tipo ‘orçamento de projeto’ - por exemplo, especificando tópicos que devem constar do documento e uma formatação.

6.1.4 Classes *Activity on Documents* e *Allowed Operations*

O conceito de atividade sobre documentos, representado pela classe *Activity on Documents*, é uma especialização do conceito genérico de atividade, onipresente em técnicas de modelagem de *workflow* [BAI 93]. Uma atividade sobre documento, neste modelo, é uma atividade que *manipula* um ou mais documentos. Dois recursos importantes são viabilizados com a introdução desse conceito. A primeira é permitir que, em tempo de execução, o usuário possa ter acesso direto aos documentos manipulados pela atividade, uma vez que estes já foram definidos em tempo de modelagem. A segunda inovação aqui apresentada é permitir definir precisamente, através da classe associativa *Allowed Operations*, quais as permissões que cada atividade possui sobre cada documento que ela manipula, como, por exemplo, se a atividade pode alterar o documento, ou criar uma nova versão dele. Essas capacidades de modelagem não são suportadas pela linguagem WPDL.

6.1.5 Conceito de ciclo de vida de um documento

Uma questão importante a ser discutida é o papel do conceito de ciclo de vida de documento nesta integração conceitual. Conforme exposto no capítulo 1, o ciclo de vida de um documento define as atividades que serão realizadas sobre esse documento, incluindo quem pode executá-las [SAD 97]. Esse conceito encontra-se presente em diversos sistemas de gerência de documentos, particularmente naqueles produtos classificados neste trabalho como DMS estendidos para *workflow*.

Na integração conceitual apresentada na figura 6.1 percebe-se que esse conceito não está explicitamente representado. Na realidade, o conceito de ciclo de vida de documento é um subconjunto do conceito, mais genérico, de processo de *workflow* (classe *Process Definition*) relacionado na figura 6.1. De fato, o ciclo de vida de documento envolve a definição de atividades referentes a um único documento, enquanto a classe *Process Definition* permite a definição de atividades envolvendo diversos documentos. É importante observar que, em uma modelagem de *workflow* para uma determinada aplicação, os diversos documentos existentes podem até mesmo seguir rotas distintas, configurando-se assim uma superposição de ciclos de vida de documentos em um único processo de *workflow*.

6.2 Mapeamento para os padrões DMA e WAPI

Uma vez definida a integração conceitual entre documentos e *workflow*, o próximo passo é estabelecer como esses conceitos serão mapeados para conceitos do padrão DMA e do padrão WAPI, de acordo com a visão exposta no capítulo 5. De modo geral, este mapeamento gerará duas saídas distintas:

- uma saída constituída por um conjunto de comandos WPDL (a linguagem padrão para definição de *workflow* da WfMC);
- uma saída constituída por definições de objetos DMA e invocações de métodos sobre eles.

A figura 6.2 mostra esquematicamente esse mapeamento.

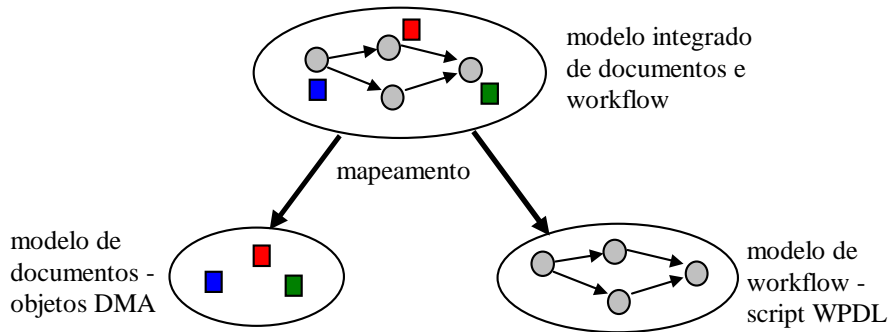


FIGURA 6.2 - Visão geral do mapeamento

Sem dúvida, diversos dos conceitos do modelo integrado não apresentam maiores dificuldades para serem mapeados. Por exemplo, as classes de *workflow*, como já comentado, foram extraídas sem alteração do meta-modelo da WfMC e, conseqüentemente, a WPDL oferece cláusulas específicas para cada uma delas. De forma similar, os conceitos de *Document* e *Version* podem ser diretamente transpostos, respectivamente, para as classes DMA *VersionSeries* e *DocVersion*.

As decisões vitais desse mapeamento concentram-se, pois, nas classes do modelo integrado que contemplam simultaneamente aspectos de documentos e de *workflow*. São essas as classes *Activity on Documents* e *Allowed Operations*. Além dessas, é importante mapear a classe *Doc Type*, visto que ela não possui correspondente direto no padrão DMA.

É de fundamental importância observar um aspecto que diferencia os padrões WAPI e DMA. A WfMC definiu uma linguagem para especificação de *workflows* - a WPDL -, ao passo que a AIIM não criou nenhuma estrutura correspondente para a modelagem de documentos. Desta forma, a capacidade de modelagem permitida pelo padrão DMA é muito restrita - em verdade, pode-se apenas utilizar as classes e os métodos oferecidos. Com o uso da WPDL, ao contrário, possui-se uma grande flexibilidade para incluir dados e atributos adicionais às entidades que ela representa. Desta forma, a opção natural para este mapeamento é expressar a maioria dos conceitos do modelo através da WPDL, deixando para o padrão DMA apenas os conceitos exclusivos da gerência de documentos. A tabela 6.2 mostra o mapeamento realizado.

TABELA 6.2 - Mapeamento entre o modelo integrado e os padrões WAPI e DMA

Classe do modelo integrado	Mapeada para
Process Definition	cláusula WORKFLOW da WPDL
Activity	cláusula ACTIVITY da WPDL
Transition	cláusula TRANSITION da WPDL
Participant	cláusula PARTICIPANT da WPDL

Classe do modelo integrado	Mapeada para
Document	classe <i>VersionSeries</i> da DMA, acrescida de cláusulas DATA da WPD L para representar os relacionamentos com as classes <i>Activity on Documents</i> e <i>Doc Type</i> , assim como dados descritivos adicionais não presentes nas classes <i>VersionSeries</i> e <i>DocVersion</i>
Version	classe <i>DocVersion</i> da DMA
Doc Type	classe <i>DocVersion</i> da DMA, acrescida de cláusulas DATA da WPD L para representar o relacionamento com a classe <i>Document</i> , assim como dados descritivos adicionais não presentes na classe <i>DocVersion</i>
Activity on Documents	cláusula ACTIVITY da WPD L, acrescida de cláusulas DATA para representar o relacionamento com a classe <i>Document</i>
Allowed Operations	cláusula DATA da WPD L, representando os atributos do relacionamento entre as classes <i>Activity on Documents</i> e <i>Document</i>

Observe-se que as classes *Process Definition*, *Activity*, *Transition*, *Participant* são completamente suportadas pelo padrão WAPI, assim como a classe *Version* o é pelo DMA. No entanto, as classes *Document*, *Doc Type*, *Activity on Documents* e *Allowed Operations* são apenas parcialmente suportadas pelos padrões. Essas últimas exigem, assim, a utilização de recursos adicionais da WPD L para seu completo mapeamento. Desta forma, será proposta uma forma organizada de utilizar a WPD L para representar os conceitos identificados. Essa forma, denominada WPD L-DOC (WPD L-Document), será apresentada na próxima seção.

6.2.1 A WPD L-DOC

A WPD L-DOC é uma extensão à WPD L no aspecto da gerência de documentos, enriquecendo assim os recursos de modelagem de *workflow* atualmente por ela oferecidos. Conforme o exposto na tabela 6.2, são três as extensões propostas pela WPD L-DOC. A primeira envolve a classe *Doc Type*, a segunda a classe *Document* e a terceira as classes *Activity on Documents* e *Allowed Operations*. As subseções a seguir tratarão de cada uma dessas extensões.

Antes, porém, é importante delinear o mecanismo geral que será utilizado para definir as extensões necessárias. A linguagem WPD L provê diversos recursos para estendê-la. Um desses recursos é a cláusula EXTENDED_ATTRIBUTE, que permite definir novos atributos para qualquer uma das entidades da linguagem [WMC 98b]. No entanto, nenhum WFMS é obrigado, pelas normas da WfMC, a interpretar um determinado EXTENDED_ATTRIBUTE - em verdade, cada WFMS deve definir *quais* EXTENDED_ATTRIBUTES ele suportará [KRE 97]. A utilização desse recurso, logo, depende de definições estritamente proprietárias e, conseqüentemente, reduzirá a

portabilidade dos comandos WPDL gerados. Logo, o uso deste recurso não é adequado para a situação colocada.

Uma solução mais adequada é a utilização da entidade *Workflow Relevant Data* do meta-modelo da WfMC, apresentada na seção 3.3.1. A WPDL permite a definição de dados que poderão ser utilizados, em tempo de execução, pelo processo de *workflow* e/ou pelas atividades, através da cláusula DATA. Assim, por exemplo, pode-se definir cláusulas DATA com as extensões desejadas, e referenciar esses dados em tempo de execução, através das funções WAPI *WMGetProcessInstanceAttributeValue* e *WMGetWorkItemAttributeValue*, conforme ilustrado na seção 3.4. A utilização desse recurso é significativamente mais atraente, visto que os comandos WPDL permanecem estritamente dentro do padrão, podendo ser interpretados por qualquer WFMS a ele aderente.

É importante destacar que a WPDL-DOC não adiciona nenhuma construção sintática à gramática da WPDL. Ao contrário, ela utiliza-se unicamente dos elementos sintáticos já existentes para definir novas estruturas de dados capazes de expressar os conceitos do modelo integrado definido.

6.2.1.1 Extensão para a classe *Doc Type*

Conforme exposto na seção 6.1.3, a classe *Doc Type* deve desempenhar duas funções: classificar o documento, de acordo com sua finalidade, e oferecer um *template* para o documento. Conforme apresentado na tabela 6.2, este *template* deve ser mapeado para um objeto da classe *DocVersion* da DMA. No entanto, uma vez que a classe *DocVersion* não oferece nenhum atributo descritivo, não é possível, com ela, descrever o documento. Assim, são necessários atributos adicionais, que podem ser definidos através de cláusulas DATA da WPDL.

A tabela 6.3 descreve os atributos adicionados necessários para o mapeamento da classe *Doc Type*.

TABELA 6.3 - Atributos adicionais para a classe *Doc Type*

Atributo	Tipo WPDL	Função
DocTypeId	Integer	identificador do tipo de documento no modelo integrado
DocTypeOID	String	OIID do objeto <i>DocVersion</i> que representa o <i>template</i> do tipo de documento
DocTypeName	String	nome do tipo de documento; utilizado para descrever o tipo de documento

Uma vez estabelecidos os atributos necessários, é preciso definir a estrutura WPDL que irá representá-los. Uma vez que um único objeto *Process Definition* pode estar relacionado a diversos objetos *Doc Type*, cada um possuindo os atributos acima citados, faz-se necessária a construção de uma estrutura de dados complexa. A WPDL

tem esta capacidade, possuindo recursos para a definição de estruturas como listas, vetores, enumerações e registros, dentro de cláusulas DATA [WMC 98b].

A figura 6.3 apresenta a estrutura de dados *DocTypeList*, definida para relacionar os tipos de documentos envolvidos em um determinado processo de *workflow*.

```

DATA      DocTypeList
TYPE      LIST OF RECORD
           INTEGER   DocTypeId
           STRING    DocTypeName
           STRING    DocTypeOIID
           END
END_DATA

```

FIGURA 6.3 - Extensão WPDL-DOC para a classe *Doc Type*

A atribuição de valores à variável *DocTypeList* pode ser feita através da cláusula *DEFAULT_DATA* da WPDL. É importante ressaltar, ainda, que esses valores podem ser consultados ou modificados, em tempo de execução, através de primitivas da interface 2 da WAPI. A figura 6.4 ilustra a definição de dois tipos de documentos em WPDL-DOC, “Relatório Técnico” e “Parecer de Avaliação”. Os valores são atribuídos aos campos do registro na mesma ordem de sua definição.

```

DATA      DocTypeList
TYPE      LIST OF RECORD
           INTEGER   DocTypeId
           STRING    DocTypeName
           STRING    DocTypeOIID
           END
DEFAULT_VALUE
           ( (1  “Relatório Técnico”      “dma:///123456789”)
           (2  “Parecer de Avaliação”     “dma:///124163264”) )
END_DATA

```

FIGURA 6.4 - Exemplo de dados para a extensão *DocTypeList*

6.2.1.2 Extensões para a classe *Document*

De acordo com o apresentado na tabela 6.2, a classe *Document* é mapeada para objetos da classe *VersionSeries* da DMA. No entanto, são necessários atributos adicionais para representar os relacionamentos com as classes *Activity on Documents* e *Doc Type*. Além disso, é interessante adicionar dados descritivos, como o nome do documento, não disponíveis no padrão DMA.

A tabela 6.4 descreve os atributos adicionados necessários para o mapeamento da classe *Document*.

TABELA 6.4 - Atributos adicionais para a classe *Document*

Atributo	Tipo WPDL	Função
DocId	Integer	identificador do documento no modelo integrado
DocTypeId	Integer	identificador do objeto <i>Doc Type</i> ao qual o documento está relacionado
DocName	String	nome do documento; utilizado para propósitos descritivos

Assim como no mapeamento da classe *Doc Type*, faz-se necessária a definição de uma estrutura de dados WPDL capaz de representar diversos documentos, cada um possuindo os atributos acima citados. A figura 6.5 apresenta a estrutura de dados *DocList*, definida para enumerar os documentos envolvidos em um determinado processo de *workflow*.

```

DATA      DocList
TYPE      LIST OF RECORD
           INTEGER   DocId
           INTEGER   DocTypeId
           STRING    DocName
           END
END_DATA

```

FIGURA 6.5 - Extensão WPDL-DOC para a classe *Document*

A atribuição de valores à variável *DocList* pode ser feita através da cláusula *DEFAULT_DATA* da WPDL. A figura 6.6 ilustra a definição de dois documentos em WPDL-DOC, “Relatório Técnico de Poluição Atmosférica” e “Parecer de Avaliação Ambiental”. Os valores são atribuídos aos campos do registro na mesma ordem de sua definição.

```

DATA      DocList
TYPE      LIST OF RECORD
           INTEGER   DocId
           INTEGER   DocTypeId
           STRING    DocName
           END
DEFAULT_VALUE
  (( 1 1 “Relatório Técnico de Poluição Atmosférica” )
   ( 2 2 “Parecer de Avaliação Ambiental” ))
END_DATA

```

FIGURA 6.6 - Exemplo de dados para a extensão *DocList*

Um aspecto adicional precisa ser considerado, no entanto, ao definir extensões para a representação da classe *Document*. A variável *DocList* serve para representar quais são os documentos que são utilizados por uma determinada definição de processo.

É armazenado, portanto, apenas o relacionamento que essas entidades possuem em termos de modelagem. É necessário, além disso, poder relacioná-las em tempo de execução, isto é, poder associar uma determinada *instância de documento* à *instância de processo* que a gerou. A tabela 6.5 apresenta os atributos necessários a esse relacionamento.

TABELA 6.5 - Atributos adicionais para instâncias de documentos

Atributo	Tipo WPDL	Função
DocId	Integer	identificador do documento no modelo integrado
ProcessInstance	Integer	identificador da instância de processo armazenada no WFMS
DocOIID	String	OIID do objeto DMA VersionSeries armazenado no DMS

Para registrar esse relacionamento, é necessário o acréscimo de uma nova estrutura de dados WPDL. Assim, foi incorporada a variável de processo DocInstanceList, apresentada na figura 6.7.

DATA	DocInstanceList
TYPE	LIST OF RECORD
	INTEGER DocId
	INTEGER ProcessInstance
	STRING DocOIID
	END
END_DATA	

FIGURA 6.7 - Extensão WPDL-DOC para instâncias de documentos

É importante ressaltar que essa variável, quando da geração do modelo integrado, não possui, via de regra, nenhum elemento. A inclusão de valores na variável DocInstanceList se dará quando da criação de um novo documento, através da invocação do método da camada de serviços *Document.CreateDocument*, apresentado na seção 7.2.2.7.

6.2.1.3 Extensão para as classes *Activity on Document* e *Allowed Operations*

Conforme apresentado na tabela 6.2, a classe *Activity on Document* é mapeada para a cláusula ACTIVITY da WPDL. No entanto, são necessários atributos adicionais para representar o relacionamento de cada atividade com os documentos (classe *Document*) utilizados no processo de *workflow*. A classe *Allowed Operations*, por sua vez, não é suportada nem pela WPDL nem pela DMA. Dessa forma, é preciso definir novos atributos de forma a representá-la.

A tabela 6.6 descreve os atributos adicionados necessários para o mapeamento das classes *Activity on Documents* e *Allowed Operations*.

TABELA 6.6 - Atributos adicionais para as classes *Activity on Documents* e *Allowed Operations*

Atributo	Tipo WPDL	Função
ActivityName	String	identificador da atividade
DocId	Integer	identificador do objeto <i>Document</i> ao qual a atividade está relacionada
AllowedOperations	List of String	lista de operações que a atividade <ActivityName> pode executar sobre o documento <DocId>

Assim como no mapeamento da classe *Document*, faz-se necessária a definição de uma estrutura de dados WPDL capaz de representar diversos documentos, cada um possuindo os atributos acima citados. A figura 6.8 apresenta a estrutura de dados *ActivityDocs*. Essa variável relaciona os documentos definidos em um *workflow* através da variável *DocList* com cada atividade definida nesse *workflow*, definindo, ainda, quais operações cada atividade pode executar sobre cada documento.

DATA	ActivityDocs
TYPE	LIST OF RECORD
	STRING ActivityName
	INTEGER DocId
	LIST OF STRING AllowedOperations
	END
END_DATA	

FIGURA 6.8 - Extensão WPDL-DOC para as classes *Activity on Documents* e *Allowed Operations*

Um aspecto importante a ser comentado é o atributo *AllowedOperations*, definido na figura 6.8 como uma lista de *strings*. É importante observar que a WPDL-DOC não define qual é o conjunto de operações possíveis sobre documentos. Ao contrário, existe uma total liberdade de entrada de valores no campo. Essa decisão foi tomada de forma que o modelo possa ser facilmente expandido para incorporar novas operações possíveis. A semântica dos valores entrados no atributo *AllowedOperations* será dada pelas comparações realizadas pelos métodos da camada de serviços, apresentada no capítulo 7.

De forma análoga ao visto nas seções anteriores, a atribuição de valores à variável *ActivityDocs* pode ser feita através da cláusula *DEFAULT_DATA* da WPDL. É importante observar que esses valores podem ser consultados ou modificados, em tempo de execução, através de primitivas da interface 2 da WAPI. A figura 6.9 ilustra a definição de três relacionamentos entre atividades e documentos em WPDL-DOC: entre a atividade “Cria Relatório de Poluição” e o documento “Relatório Técnico de Poluição Atmosférica”; entre a atividade “Revisa Relatório Técnico” e o documento “Relatório Técnico de Poluição Atmosférica”; e entre a atividade “Cria Parecer” e o documento

“Parecer de Avaliação Ambiental”. Os valores são atribuídos aos campos do registro na mesma ordem de sua definição.

DATA	ActivityDocs		
TYPE	LIST OF RECORD		
	STRING		ActivityName
	INTEGER		DocId
	LIST OF STRING		AllowedOperations
	END		
DEFAULT_VALUE			
	((“Cria Relatório de Poluição”	1	(“Modify” “Remove”))
	(“Revisa Relatório Técnico”	1	(“Version”))
	(“Avalia Parecer”	2	(“ViewOnly”))
END_DATA			

FIGURA 6.9 - Exemplo de dados para a extensão ActivityDocs

A versão da camada de serviços apresentada neste trabalho suporta quatro operações básicas da gerência de documentos. Essas operações são um subconjunto de operações típicas de sistemas de gerência de documentos, como as descritas em [FER 93], [GRA 97] e [NOR 98]. São elas:

- modificação e salvamento de um documento;
- criação de uma nova versão de um documento;
- visualização de um documento;
- remoção de um documento da base de documentos.

A tabela 6.7 relaciona essas operações suportadas com o valor correspondente necessário do atributo AllowedOperations.

TABELA 6.7 - Exemplos de valores possíveis para AllowedOperations

Valor	Semântica
Modify	indica que o participante pode, no contexto daquela atividade, visualizar o documento e modificar seu conteúdo
Version	indica que o participante pode, no contexto daquela atividade, visualizar o documento e criar uma nova versão dele
ViewOnly	indica que o participante pode, no contexto daquela atividade, somente visualizar o documento
Remove	indica que o participante pode, no contexto daquela atividade, visualizar o documento e removê-lo da base de documentos

É importante ressaltar que o leque de operações modeladas pode ser expandido, sem qualquer prejuízo de portabilidade do arquivo WPDL-DOC gerado. Para tal, a

única exigência seria adaptar a camada de serviços, para que essa possa reconhecer os novos valores e manipulá-los adequadamente.

6.2.2 Dinâmica do mapeamento

Na integração proposta, o único modelo efetivamente percebido pelo usuário é o modelo integrado de documentos e *workflow*. O usuário poderá, assim, representar os diversos conceitos apresentados, como processos, atividades, tipos de documentos e documentos. No momento desejado, o usuário poderá implantar os modelos desenvolvidos. Como já citado, o ato de implantar o modelo envolve tanto a geração de um arquivo WPDL-DOC, quanto a invocação de métodos DMA para a criação de objetos DMA.

Esta implantação do modelo necessita seguir uma determinada ordem de passos. Como já visto, o arquivo em WPDL-DOC possui, em suas cláusulas, referências aos OIID dos objetos *Doc Type*. Para tal, evidentemente, é preciso que, já de antemão, esses objetos tenham sido criados no DMS. Assim se define, então, a ordem das etapas:

- definição do *script* WPDL-DOC, descrevendo todos os processos, atividades, participantes, transições, tipos de documentos e documentos. No entanto, não é atribuído, ainda, valor ao atributo *DocTypeOIID* da variável *DocTypeList* (etapa 1 da figura 6.10);
- geração dos objetos DMA *DocVersion* que representem os objetos da classe *Doc Type*. Essa geração é feita por um pequeno programa que invoca os métodos DMA *ConnectDocSpace* e *CreateObject*. Esse mesmo programa deve obter os valores gerados como OIID para os objetos *DocVersion*. Esses valores são criados pelo próprio DMS, e podem ser recuperados através de chamadas ao método *GetPropValOIIDById* (etapa 2 da figura 6.10);
- de posse destes valores, o atributo *DocTypeOIID* da variável *DocTypeList* do *script* WPDL-DOC pode receber seus valores. Assim, o *script* está completo (etapa 3 da figura 6.10).
- uma vez completo, o arquivo pode ser submetido ao WFMS. O procedimento exato desta geração não é definido pela WfMC, sendo, portanto, específico para cada produto de gerência de *workflow* (etapa 4 da figura 6.10)

É importante ressaltar que não está no escopo deste trabalho o desenvolvimento de ferramentas para automatizar essa dinâmica de mapeamento. Assim, os passos acima listados devem ser executados manualmente pelo profissional responsável pela modelagem do sistema de *workflow* e documentos.

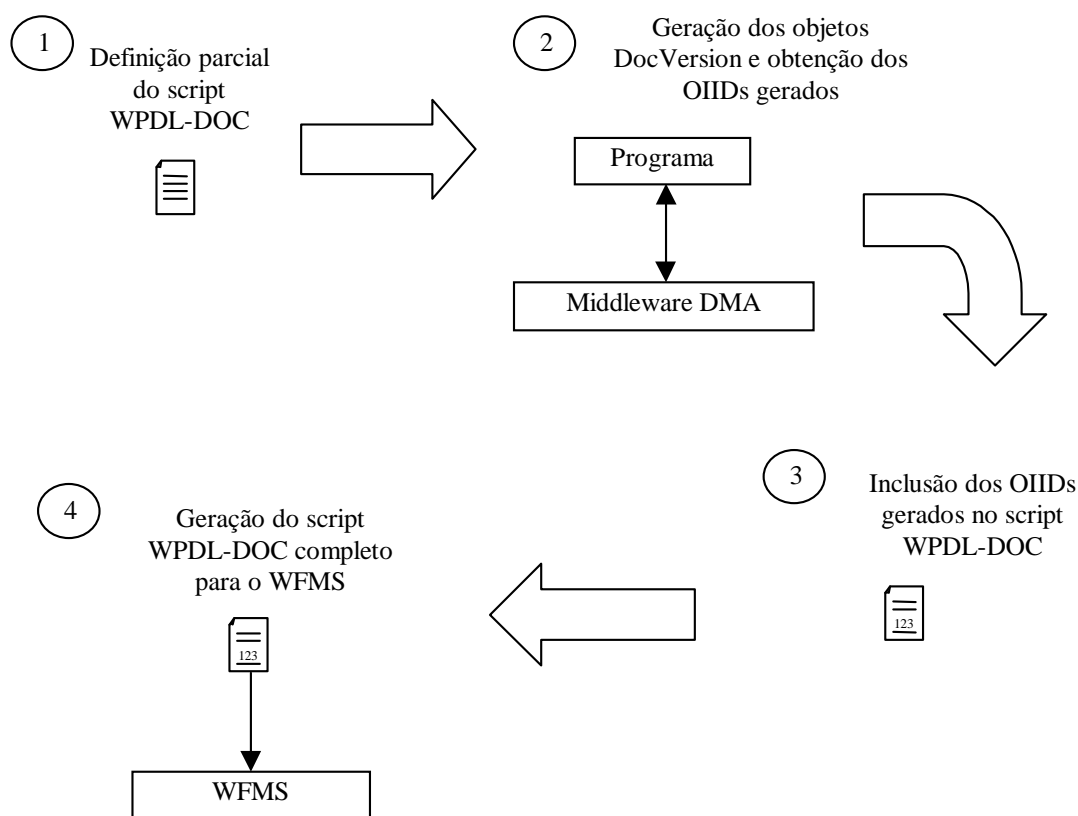


FIGURA 6.10 - Processo de modelagem de documentos e *workflow*

6.3 Exemplo de mapeamento

A fim de bem ilustrar o mapeamento proposto neste capítulo, esta seção apresenta um exemplo detalhado de um processo de *workflow* ligado a documentos. Assim, será possível observar em detalhes as respectivas estruturas DMA e WAPI geradas. Inicialmente, na seção 6.3.1, será apresentado o processo tomado como exemplo e, posteriormente, na seção 6.3.2, será apresentado o mapeamento desse processo para a WPDL-DOC e para os objetos DMA correspondentes.

6.3.1 Definição do processo exemplo

O processo aqui apresentado é uma parte de um processo da realidade, o processo de licenciamento ambiental da Fundação Estadual de Proteção Ambiental do Estado do Rio Grande do Sul (FEPAM-RS). O processo de licenciamento ambiental tem por função avaliar o impacto de empreendimentos (industriais, agrícolas, de mineração e urbanísticos) sobre o meio ambiente, concedendo, se for julgado conveniente, uma licença ambiental de operação. De acordo com a legislação, esse documento é obrigatório a qualquer empreendimento, devendo ser renovado, na maioria dos casos, anualmente.

A emissão da licença ambiental é o resultado de um processo de análise de diversos aspectos da poluição produzidos por cada empreendimento. Essas análises são registradas em documentos denominados pareceres técnicos. A confecção desses

pareceres técnicos é a parte mais crítica do processo de licenciamento ambiental. Por ser uma parte do processo de licenciamento ambiental bastante complexa e com elevada utilização de documentos, ela foi escolhida para exemplificar a aplicação das técnicas apresentadas nesse capítulo.

No processo selecionado, inicialmente, o setor de protocolo recebe a solicitação de emissão de licença da empresa. Então, os técnicos responsáveis pela produção dos pareceres podem começar a atuar. Diversos aspectos da poluição devem ser analisados, dependendo da natureza do empreendimento. No processo apresentado, são tratadas apenas a poluição atmosférica e a emissão de efluentes líquidos, sendo que há técnicos especializados em cada uma dessas áreas. Uma vez confeccionados esses pareceres, eles são enviados a um coordenador do processo, que os avalia, podendo solicitar alterações nos pareceres. Essas solicitações são registradas em um documento de comentários. Nesse caso, os pareceres retornam aos técnicos que os escreveram, que podem corrigi-los, criando assim novas versões dos pareceres. Após, os documentos são reenviados ao coordenador, que pode aceitá-los ou solicitar novas alterações, reiniciando o ciclo. Por final, quando todos os pareceres técnicos forem aprovados pelo coordenador, eles são reunidos em um único documento, denominado parecer final. Esse parecer é o núcleo da licença ambiental emitida pela FEPAM.

A descrição desse processo ilustra a íntima relação que há entre a gerência de *workflow* e a gerência de documentos. Praticamente todas as atividades não só envolvem a manipulação de documentos, mas também essa manipulação é rigorosamente definida. O coordenador do processo, por exemplo, não pode alterar o conteúdo dos pareceres por ele analisados, podendo apenas tecer comentários em um documento à parte. Similarmente, os técnicos não podem alterar o documento de comentários, podendo apenas consultá-lo. Não podem, ainda, modificar o parecer que escreveram, após ele ter sido analisado pelo coordenador do processo, podendo apenas criar novas versões do parecer. Com esse recurso é possível, por exemplo, ter o controle exato de quais alterações foram realizadas, e em que momento.

É interessante listar os documentos e tipos de documentos identificados nesse processo. A tabela 6.8 os relaciona.

TABELA 6.8 - Documentos e tipos de documentos para o processo exemplo

Tipos de documento	Documentos
Parecer	<ul style="list-style-type: none"> • Parecer de Condições Atmosféricas • Parecer de Efluentes Líquidos • Parecer Final
Comentário	<ul style="list-style-type: none"> • Comentário sobre o Parecer de Condições Atmosféricas • Comentário sobre o Parecer de Efluente Líquidos

Para facilitar a compreensão do processo modelado, será apresentada uma representação gráfica do processo modelado. No entanto, é importante ressaltar que, de acordo com a proposta desta dissertação de mestrado, não é necessária uma

representação gráfica para a definição de processos e de documentos. A representação gráfica é, antes de tudo, um mecanismo auxiliar para facilitar a definição do *script* WPDL e dos objetos DMA necessários.

Conforme comentado anteriormente, há, na literatura, uma ausência de notações gráficas que representem simultaneamente os aspectos de modelagem de *workflow* e de modelagem de documentos. Assim, o diagrama apresentado utiliza uma adaptação da notação utilizada pela técnica de modelagem de *workflow* denominada *Trigger Modeling* [JOO 94]. A técnica *Trigger Modeling* incorpora alguns elementos básicos da modelagem de *workflow*, como atividades, papéis e transições. Faltam a essa técnica, no entanto, diversos dos conceitos do modelo conceitual integrado apresentado na figura 6.1, como os de regras de transição, documentos (classe *Document*), tipos de documentos (classe *Doc Type*), atividades sobre documentos (classe *Activity on Documents*) e operações permitidas sobre os documentos (classe *Allowed Operations*).

Faz-se necessário, portanto, estender a notação para representar esses conceitos. Assim, foram adicionados à técnica *Trigger Modeling* os conceitos de regras de transição, de documentos, de atividade sobre documentos e de operações permitidas sobre documentos. O conceito de tipo de documentos não foi incorporado à representação gráfica, sob o risco de torná-la demasiado confusa, haja vista que coexistiria com a representação gráfica do conceito de documento. Uma vez que a representação gráfica tem papel apenas auxiliar, essa lacuna não traz prejuízo algum para a proposta.

A tabela 6.9 apresenta a representação gráfica dos conceitos introduzidos.

TABELA 6.9 - Representação gráfica dos conceitos introduzidos

Conceito	Representação Gráfica	Explicação
Regra de transição	[condição] →	A transição indicada ocorrerá se [condição] for verdadeira
Documento	■ “Parecer Final”	Define o documento “Parecer Final”
Atividade sobre documentos e operações permitidas	<div style="border: 1px solid black; padding: 2px; display: inline-block;">Atividade A</div> ■ Modify “Parecer Final”	A atividade “A” manipula o documento “Parecer Final” e pode executar a operação “Modify” sobre ele

A figura 6.11 ilustra graficamente a modelagem do processo exemplo.

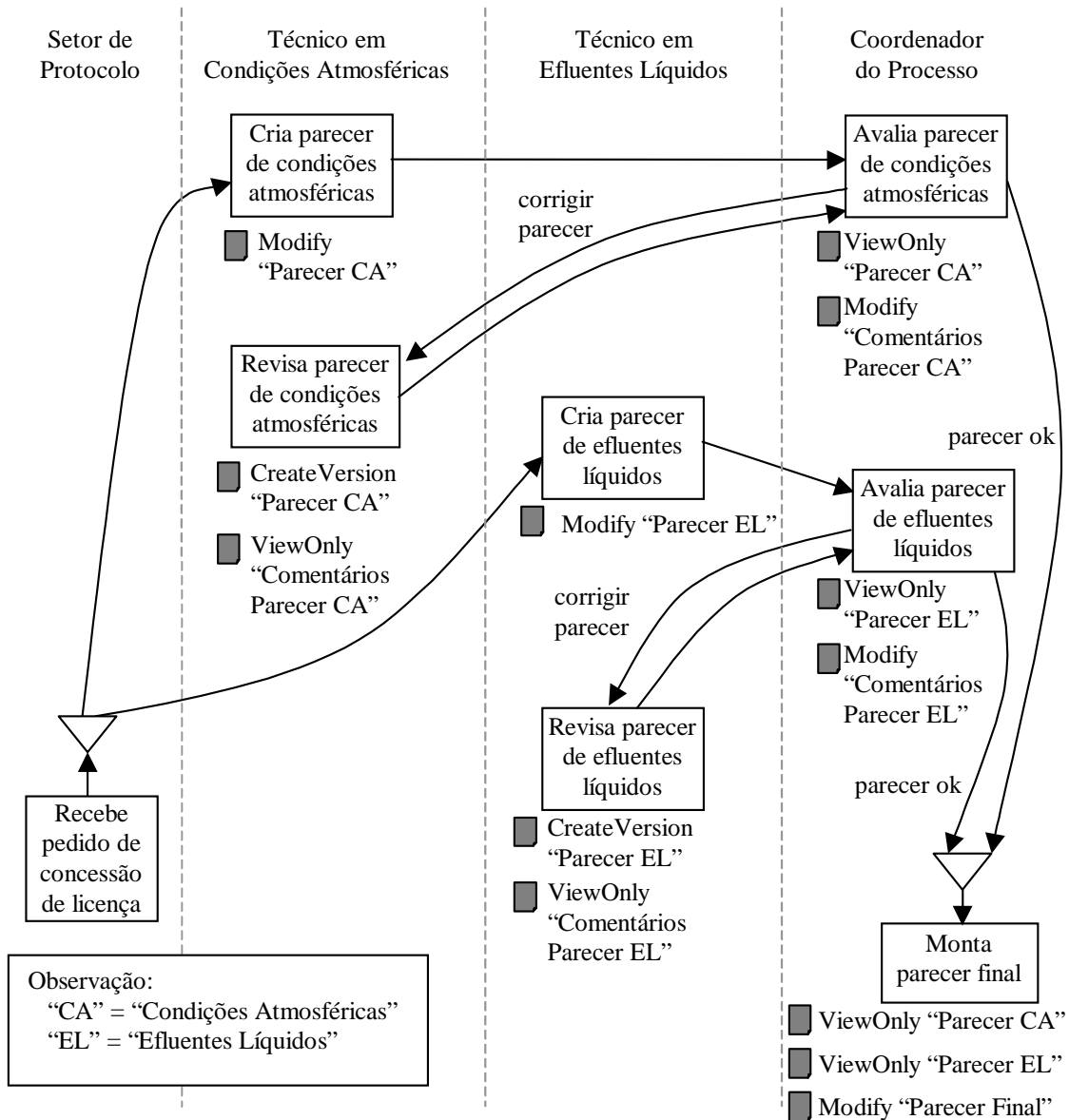


FIGURA 6.11 - Representação gráfica para o processo de licenciamento ambiental

6.3.2 Mapeamento para a WPD-L-DOC e DMA

Esta seção apresenta as estruturas WPD-L-DOC e DMA necessárias para representar o processo de licenciamento ambiental ilustrado na seção 6.3.1. Conforme a dinâmica de mapeamento apresentada na seção 6.2, o primeiro passo a ser executado deve ser a definição do *script* WPD-L-DOC parcial, contendo todas as definições de documentos e *workflow* necessárias, exceto os OIID dos objetos *Doc Type*. Após isso, os objetos *Doc Type* seriam criados no DMS, podendo então ser obtidos seus OIID. Esses OIID devem, então, ser colocados no *script* WPD-L-DOC, o qual pode, então, ser gerado no WFMS.

A fim de simplificar a apresentação do exemplo, o *script* WPD-L-DOC será mostrado já completo, ou seja, com valores para o OIID dos objetos *Doc Type*. Isso significa que, previamente, os objetos *Doc Type* já devem ter sido gerados. É necessário

gerar dois objetos *Doc Type*, já que, como ilustrado na tabela 6.7, foram identificados dois tipos de documentos no processo exemplo: um objeto “Parecer” e outro objeto “Comentários”. Esses objetos servirão como *templates* para os documentos pertencentes ao tipo, conforme explicado na seção 6.1.3.

O *script* WPDL-DOC é apresentado na figura 6.12. Para facilitar a compreensão do *script*, foram colocados comentários em diversos pontos, especialmente início das seções de definições gerais, de definição dos processos, de definição das atividades, de definição das transições de definição dos participantes e de definição dos dados relevantes do processo.

```
// definições gerais do modelo
MODEL                `FEPAM`
  WPD_L_VERSION      "1.0"
  VENDOR             "CPGCC:1.0"
  CREATED            1998-08-01

// definição do processo de licenciamento ambiental

WORKFLOW             `wf_1`
  NAME               "Processo de licenciamento ambiental"
  DESCRIPTION        "Processo exemplo do mapeamento"
  AUTHOR             "Vinicius Amaral"

// definição das atividades do processo

ACTIVITY             `a_1`
  NAME               "Recebe pedido de concessão de licença"
  IMPLEMENTATION     NO
  CHARACTERISTIC     BEGIN
  PERFORMER          `p_1`
END_ACTIVITY
ACTIVITY             `a_2`
  NAME               "Cria parecer de condições atmosféricas"
  IMPLEMENTATION     NO
  PERFORMER          `p_2`
END_ACTIVITY
ACTIVITY             `a_3`
  NAME               "Cria parecer de efluentes líquidos"
  IMPLEMENTATION     NO
  PERFORMER          `p_3`
END_ACTIVITY
ACTIVITY             `a_4`
  NAME               "Avalia parecer de condições atmosféricas"
  IMPLEMENTATION     NO
  PERFORMER          `p_4`
END_ACTIVITY
ACTIVITY             `a_5`
  NAME               "Avalia parecer de efluentes líquidos"
  IMPLEMENTATION     NO
  PERFORMER          `p_4`
END_ACTIVITY
ACTIVITY             `a_6`
```

```

NAME                "Revisa parecer de condições atmosféricas"
IMPLEMENTATION      NO
PERFORMER           'p_2'
END_ACTIVITY
ACTIVITY           'a_7'
NAME                "Revisa parecer de efluentes líquidos"
IMPLEMENTATION      NO
PERFORMER           'p_3'
END_ACTIVITY
ACTIVITY           'a_8'
NAME                "Monta parecer final"
IMPLEMENTATION      NO
CHARACTERISTIC      END
PERFORMER           'p_4'
END_ACTIVITY

```

// definição das transições entre as atividades

```

TRANSITION         't_1'
FROM                'a_1'
TO                  'a_2'
END_TRANSITION
TRANSITION         't_2'
FROM                'a_1'
TO                  'a_3'
END_TRANSITION
TRANSITION         't_3'
FROM                'a_2'
TO                  'a_4'
END_TRANSITION
TRANSITION         't_4'
FROM                'a_3'
TO                  'a_5'
END_TRANSITION
TRANSITION         't_5'
FROM                'a_4'
TO                  'a_6'
CONDITION           aval_parecer_ca = 'corrigir parecer'
END_TRANSITION
TRANSITION         't_6'
FROM                'a_5'
TO                  'a_7'
CONDITION           aval_parecer_el = 'corrigir parecer'
END_TRANSITION
TRANSITION         't_7'
FROM                'a_6'
TO                  'a_4'
END_TRANSITION
TRANSITION         't_8'
FROM                'a_7'
TO                  'a_5'
END_TRANSITION
TRANSITION         't_9'
FROM                'a_4'
TO                  'a_8'

```

```

CONDITION          aval_parecer_ca = 'parecer ok'
END_TRANSITION
TRANSITION      't_10'
FROM               'a_5'
TO                 'a_8'
CONDITION          aval_parecer_el = 'parecer ok'
END_TRANSITION

// definição dos participantes do processo

PARTICIPANT     'p_1'
NAME               "Setor de Protocolo"
TYPE               ORGANISATIONAL_UNIT
DESCRIPTION        "Inicia o processo de licenciamento ambiental"
END_PARTICIPANT
PARTICIPANT     'p_2'
NAME               "Técnico em Condições Atmosféricas"
TYPE               ROLE
DESCRIPTION        "Produz o parecer técnico de condições atmosféricas"
END_PARTICIPANT
PARTICIPANT     'p_3'
NAME               "Técnico em Efluentes Líquidos"
TYPE               ROLE
DESCRIPTION        "Produz o parecer técnico de efluentes líquidos"
END_PARTICIPANT
PARTICIPANT     'p_4'
NAME               "Coordenador do Processo"
TYPE               ROLE
DESCRIPTION        "Avalia e aprova os pareceres técnicos"
END_PARTICIPANT

// definição dos dados relevantes do processo

// atributo para o roteamento de pareceres de condições atmosféricas
DATA            'aval_parecer_ca'
TYPE               STRING
DEFAULT_VALUE      "Parecer ok"
DESCRIPTION        "Avaliação do parecer de condições atmosféricas"
END_DATA

// atributo para o roteamento de pareceres de efluentes líquidos
DATA            'aval_parecer_el'
TYPE               STRING
DEFAULT_VALUE      "Parecer ok"
DESCRIPTION        "Avaliação do parecer de efluentes líquidos"
END_DATA

// lista dos tipos de documento: dados conforme tabela 6.7
DATA            'DocTypeList'
TYPE               LIST OF RECORD
                   INTEGER      DocTypeId
                   STRING       DocTypeName
                   STRING       DocTypeOIID
                   END
DEFAULT_VALUE
( (1 "Parecer"      "dma:///4356743455")

```

```

(2 "Comentários" "dma:///1834585902" )
DESCRIPTION "Lista de tipos de documento do processo"
END_DATA

// lista dos documentos do processo: dados conforme tabela 6.7
DATA 'DocList'
TYPE LIST OF RECORD
INTEGER DocId
INTEGER DocTypeId
STRING DocName
END
DEFAULT_VALUE
((1 1 "Parecer de Condições Atmosféricas" )
(2 1 "Parecer de Efluentes Líquidos" )
(3 1 "Parecer Final" )
(4 2 "Comentário sobre o Parecer de Condições Atmosféricas")
(5 2 "Comentário sobre o Parecer de Efluentes Líquidos" )
)
DESCRIPTION "Lista de documentos do processo"
END_DATA

// lista das instâncias de documentos de cada instância de processo
DATA 'DocInstanceList'
TYPE LIST OF RECORD
INTEGER DocId
INTEGER ProcessInstance
STRING DocOIID
END
END_DATA

// lista das permissões de cada atividade do processo sobre cada
// documento definido
DATA 'ActivityDocs'
TYPE LIST OF RECORD
STRING ActivityName
INTEGER DocId
LIST OF STRING AllowedOperations
END
DEFAULT_VALUE
(("Cria parecer de condições atmosféricas" 1 ("Modify"))
"Cria parecer de efluentes líquidos" 2 ("Modify"))
"Avalia parecer de condições atmosféricas" 1 ("ViewOnly"))
"Avalia parecer de condições atmosféricas" 4 ("Modify"))
"Avalia parecer de efluentes líquidos" 2 ("ViewOnly"))
"Avalia parecer de efluentes líquidos" 5 ("Modify"))
"Revisa parecer de condições atmosféricas" 1 ("CreateVersion"))
"Revisa parecer de condições atmosféricas" 4 ("ViewOnly"))
"Revisa parecer de efluentes líquidos" 2 ("CreateVersion"))
"Revisa parecer de efluentes líquidos" 5 ("ViewOnly"))
"Monta parecer final" 1 ("ViewOnly"))
"Monta parecer final" 2 ("ViewOnly"))
"Monta parecer final" 5 ("Modify")) )
DESCRIPTION "Operações permitidas sobre cada documentos"
END_DATA
END_WORFKLOW
END_MODEL

```

FIGURA 6.12 - Script WPD-L-DOC referente ao processo exemplo

7 Descrição do Protótipo

Neste capítulo serão tratados aspectos da implementação do protótipo desenvolvido. Inicialmente, é feito um detalhamento da arquitetura de integração definida na seção 5.2.2. Neste detalhamento, são apresentados todos os módulos de software que compõem a arquitetura, assim como os procedimentos utilizados para integrá-los. A seguir, são descritos em detalhes os dois módulos de software construídos neste trabalho, conforme a proposta do capítulo 5: a camada de serviços e o aplicativo cliente. A camada de serviços oferece um conjunto de classes integrando funcionalidades de gerência de *workflow* (através da WAPI) e funcionalidades de gerência de documentos (através da DMA). A aplicação cliente fará uso desses recursos, invocando os métodos da camada de serviços e gerenciando a interação com o usuário. É importante observar que, adotando tal arquitetura, todos os aspectos de utilização dos padrões DMA e WAPI são ocultados da aplicação cliente, que, assim, possui conhecimento somente dos métodos da camada de serviços.

7.1 Detalhamento da Arquitetura

Esta seção apresenta um detalhamento da arquitetura de integração proposta na seção 5.2.2. De modo geral, os módulos que compõem a arquitetura podem ser divididos em três categorias: os módulos de gerência de documentos, os módulos de gerência de *workflow* e os módulos de integração. A figura 7.1 ilustra a arquitetura com a classificação apresentada.

Entre os módulos de gerência de documentos, estão o *middleware* DMA, um determinado DMS e um *driver* que realize o mapeamento entre a SPI DMA e a API proprietária desse DMS. O *middleware* DMA é formado por um conjunto de objetos COM, podendo assim ser acessado por programas escritos em linguagens que suportem integração com esse modelo de objetos, como C e C++ [ORF 96]. Ainda, o *middleware* DMA foi desenvolvido e é fornecido gratuitamente pela AIIM, enquanto os DMS e seus respectivos *drivers* DMA são desenvolvidos tipicamente por companhias comerciais.

Entre os módulos de gerência de *workflow*, estão um determinado WFMS e um *driver* que realize o mapeamento entre a WAPI e a API proprietária desse WFMS. A WAPI é formada por um conjunto de funções em linguagem C [WMC 97], podendo assim ser invocada por programas escritos em diversas linguagens, como C, C++ e Java. O WFMS e o driver WAPI são desenvolvidos tipicamente por companhias comerciais.

Entre os módulos de integração, estão a camada de serviços e o aplicativo cliente de documentos e *workflow*. Esses dois módulos foram desenvolvidos nesta dissertação de mestrado, e serão detalhados, respectivamente, nas seções 7.2 e 7.3. A linguagem de programação utilizada para desenvolvê-los foi a linguagem Java [COA 98][WEB 97]. Diversos fatores foram determinantes para a sua escolha, como a facilidade para a criação de aplicações distribuídas, a independência de plataforma e a maturidade de seu modelo de objetos [WEB 97]. No desenvolvimento desses módulos, foi utilizada a ferramenta de desenvolvimento Borland JBuilder [INP 98].

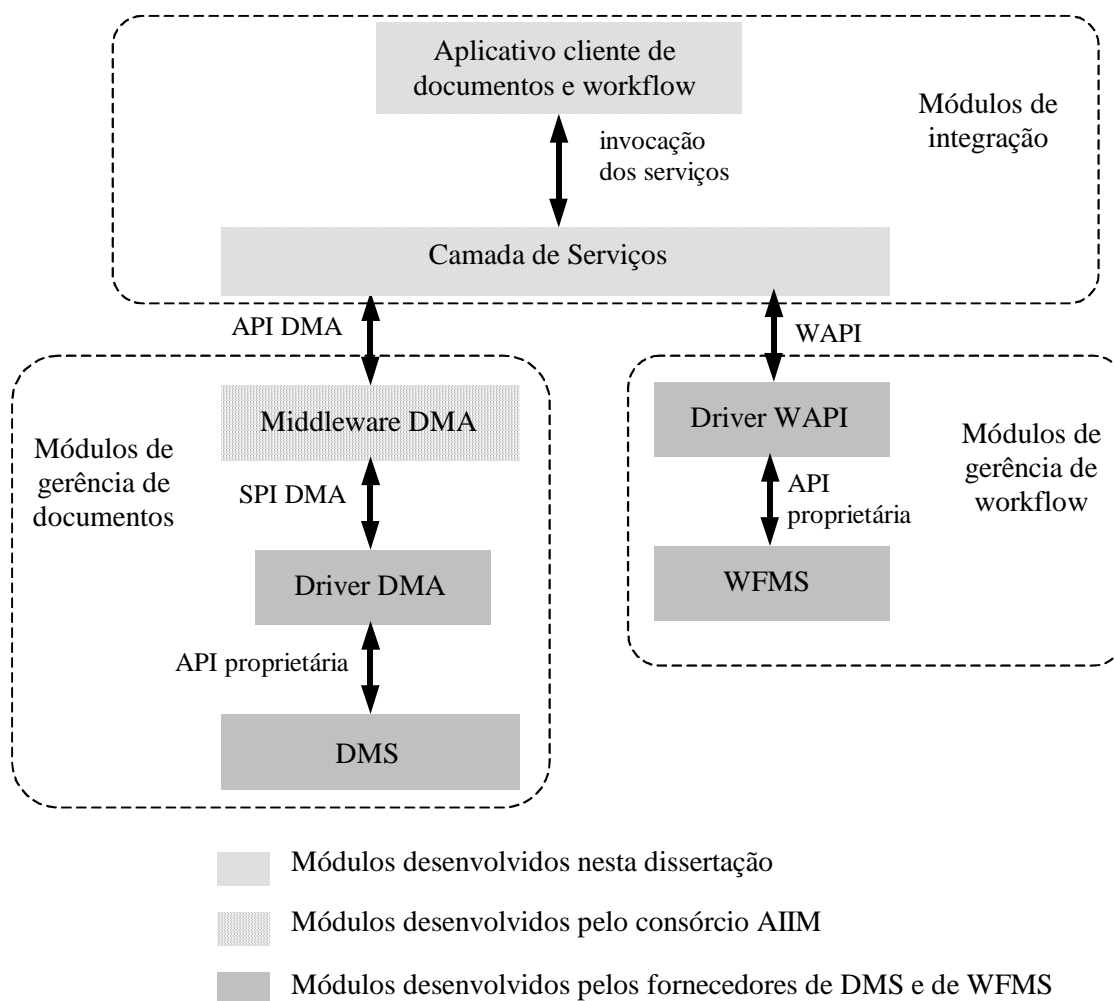


FIGURA 7.1 - Arquitetura detalhada da proposta de integração

Ainda, é importante analisar como as diversas integração entre os módulos citados foram realizadas. Evidentemente, faz sentido apenas citar as integrações que envolvem os módulos construídos nesta dissertação, pois os pormenores da integração de módulos de terceiros não é relevantes para a proposta de arquitetura integrada. Primeiramente, o aplicativo cliente integra-se com a camada de serviços através da RMI (*Remote Method Invocation*), um mecanismo de Java que permite a comunicação cliente/servidor entre dois aplicativos [WEB 97]. A integração entre a camada de serviços e o *middleware* DMA é mais complexa. Uma vez que a integração entre Java e COM não faz atualmente parte do corpo comum da linguagem, optou-se por construir um módulo C++ adicional que contivesse todas as chamadas a COM. Os métodos desse módulo C++ são, por sua vez, invocados pela camada de serviços a partir da interface C++ nativa de Java [WEB 97]. Por sua vez, a integração entre a camada de serviços e o *driver* WAPI é realizada através da interface C nativa de Java, que permite a um módulo Java invocar transparentemente uma função C localizada em outro aplicativo [WEB 97]. A tabela 7.1 resume esses mecanismos.

TABELA 7.1 - Mecanismos de integração utilizados

Integração		Mecanismo
Aplicativo cliente de documentos e <i>workflow</i>	Camada de Serviços	RMI
Camada de Serviços	<i>Middleware</i> DMA	Invocação indireta das funções do <i>middleware</i> através de um módulo C++, invocado pela interface 'C++' nativa de Java
Camada de Serviços	Driver WAPI	Invocação direta das funções do <i>driver</i> pela interface 'C' nativa de Java

É importante, ainda, ressaltar o esforço necessário para a compreensão precisa e completa dos padrões DMA e WAPI, passo indispensável para a definição da integração. Particularmente quanto ao padrão DMA, esse esforço foi consideravelmente alto, já que sua arquitetura é bastante sofisticada e seu modelo de objetos é o COM, cuja complexidade é reconhecidamente elevada [ORF 96].

7.2 A Camada de Serviços

A camada de serviços é formada por um conjunto de classes que espelham o domínio do problema, representando a funcionalidade implementada pelo sistema como um todo. Essa funcionalidade é um subconjunto dos serviços providos pelos padrões WAPI [WMC 97] e DMA [AII 97]. As funções de gerência de *workflow* e de gerência de documentos selecionadas representam as atividades mais comuns na manipulação de processos e documentos, podendo ser encontradas em diversos WFMS, como o Oracle Workflow Cartridge e o FileNet Visual WorkFlo [FIL 98a], assim como diversos DMS, como o Panagon Mezzanine [FIL 98b] e o GDOC [GRA 97].

A lista abaixo enumera as funcionalidades providas pela camada de serviços:

- conexão/desconexão ao sistema;
- recuperação das definições de processos que o participante pode disparar;
- disparo (início) de um processo;
- recuperação da lista de trabalho do participante;
- disparo (início) de um determinado item de trabalho;
- criação de um novo documento vinculado a um determinado item de trabalho;
- recuperação da lista de versões de um documento vinculado a um determinado item de trabalho;
- recuperação de uma versão de um documento;
- salvamento (gravação) de uma versão de um documento;
- criação de uma nova versão de um documento;

- remoção (exclusão) de uma versão de um documento;
- conclusão (término) de um item de trabalho;
- delegação de um item de trabalho a outro participante;

Cada uma dessas funcionalidades poderá exigir, em sua implementação, a utilização de serviços de gerência de *workflow*, a utilização de serviços de gerência de documentos, ou ambos, simultaneamente. É importante ressaltar que a camada de serviços não possui qualquer preocupação com a interface com o usuário; esta tarefa deve ser desempenhada por outro aplicativo, o qual fará uso dos serviços aqui definidos. Dessa forma, pode-se afirmar que trata-se de uma arquitetura de software em três camadas (*three-tier*) [ORF 96].

A seção a seguir definirá as classes que constituem a camada de serviços, com seus respectivos métodos. Logo após, cada método terá sua estrutura interna detalhada, ilustrando assim a utilização dos padrões DMA e WAPI.

7.2.1 Diagrama de classes

Uma vez levantados os requisitos para a camada de serviços, o próximo passo é desenvolver um *diagrama de classes*. Esse diagrama ilustra quais as classes farão parte da implementação do sistema, juntamente com seus respectivos métodos e atributos. A notação utilizada é a da UML 1.0 [FOW 97].

É importante observar que esse diagrama de classes, apresentado na figura 7.2, foi construído sob uma *perspectiva de implementação*. Dessa forma, ele difere significativamente no modelo conceitual apresentado na seção 6.1, apesar de os conceitos expressos serem basicamente os mesmos. Um diagrama de implementação de classes representa efetivamente o software que foi desenvolvido, ao passo que um diagrama conceitual representa os elementos da realidade identificados [FOW 97].

Nesta seção, serão abordados aspectos gerais do diagrama, particularmente ressaltando as diferenças em relação ao diagrama conceitual apresentado na figura 6.1. A seção 7.3 discutirá cada um dos métodos em detalhes.

A principal diferença deste diagrama de implementação em relação ao diagrama conceitual anteriormente apresentado é, sem dúvida, a introdução da classe *Server*. Essa classe desempenha um papel fundamental na camada de serviços, sendo responsável por aspectos como a conexão ao WFMS e ao DMS e pela recuperação de objetos junto a esses sistemas. Ela ocupa, por exemplo, o papel de fábrica (*object factory* [GAM 95]) dos objetos *Process Definition* e *Work Item*.

Outro aspecto relevante nesse diagrama é a mudança da cardinalidade do relacionamento entre as classes *Work Item* e *Document*. A classe *Work Item* representa a classe *Activity* do modelo conceitual, e essa alteração na nomenclatura justifica-se pela distinção feita pela WfMC entre os conceitos de atividade e item de trabalho. Em termos conceituais, é simples perceber que a relação entre essas classes é de cardinalidade n:n, uma vez que um certo item de trabalho pode possuir diversos documentos a ele

relacionados, e um certo documento pode ser manipulado por diversos itens de trabalho. No entanto, neste diagrama de classes de implementação, um determinado objeto da classe *Document* está relacionado a apenas um objeto da classe *Work Item*. Esta decisão deve-se ao fato de que, mesmo existindo dois itens de trabalho (objetos *Work Item*) ligados a um certo documento (objeto *Document*), esses itens de trabalho ou serão executados simultaneamente, por participantes diferentes, ou serão executados em momentos diferentes, pelo mesmo participante ou participantes diferentes. Assim, não há possibilidade de o mesmo documento estar ligado, no mesmo *espaço de endereçamento*, a itens de trabalho diferentes. Logo, em termos de implementação, é perfeitamente aceitável tratar o relacionamento entre as classes *Work Item* e *Document* como 1:n. Dessa forma, passam a fazer parte da classe *Document*, igualmente, os atributos indicando as operações possíveis sobre cada documento, que pertenciam à classe associativa *Allowed Operations* no diagrama de classes conceitual da figura 6.1. E, por essa razão, a classe *Allowed Operations* não está mais presente neste diagrama de classes de implementação.

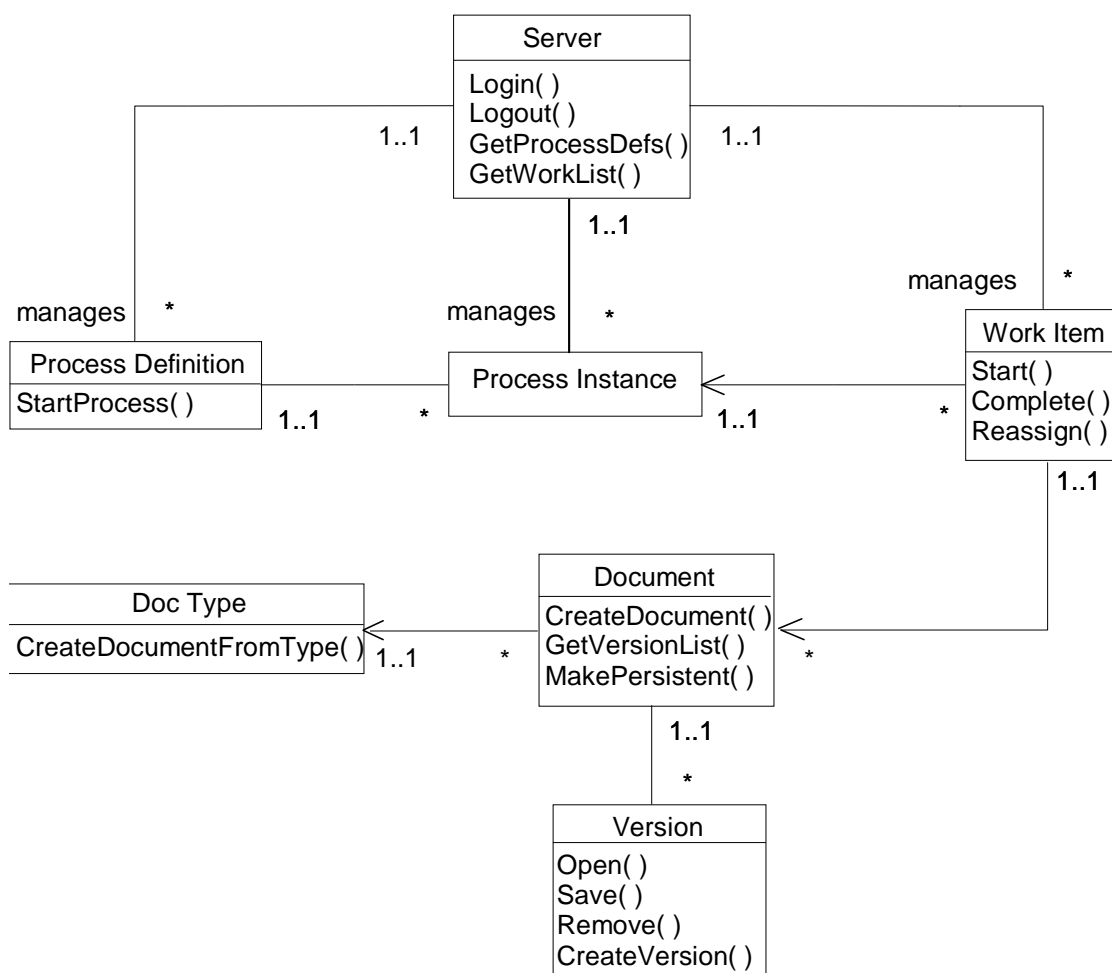


FIGURA 7.2 - Diagrama de classes para a camada de serviços

Por final, é importante ressaltar o aspecto de *navegabilidade* entre os objetos. De acordo com [FOW 97], existe navegabilidade de uma classe para outra, em um diagrama de classes em perspectiva de implementação, quando uma classe (origem)

possui atributos que são apontadores para objetos da outra classe (destino). Isso significa que a classe origem tem meios para acessar objetos da classe destino, permitindo, efetivamente, a navegação de um objeto a outro. A navegabilidade é denotada pela presença ou ausência de setas nas extremidades das linhas das associações. A seta em uma extremidade indica que a classe de onde parte a seta possui como atributo um apontador para objetos da outra classe. A ausência de setas pode possuir dois sentidos: ou existe navegabilidade em ambas direções (ambas classes possuem apontadores para a outra), ou a navegabilidade entre as classes é indeterminada [FOW 97].

No diagrama de classes apresentado, pode-se observar a existência de navegabilidade unidirecional da classe *Work Item* para a classe *Process Instance*, da classe *Document* para a classe *Doc Type* e da classe *Work Item* para a classe *Document*. Isso indica, respectivamente, que cada item de trabalho sabe a qual processo ele pertence, que cada documento sabe qual o seu tipo e que cada item de trabalho sabe quais documentos trata. Além desses, existe navegabilidade bidirecional entre as classes *Document* e *Version*. Isso indica que cada documento sabe quais são suas versões, e que cada versão sabe a que documento pertence.

7.2.2 Implementação dos serviços

Uma vez definidos as classes e os métodos que fazem parte da camada de serviços, é necessário, agora, ilustrar como essas funcionalidades podem ser implementadas através do uso dos padrões DMA e WAPI. Esse ponto constitui-se, de fato, em um dos mais importantes deste trabalho de mestrado, pois mostrará a viabilidade da integração de qualquer sistema de gerência de documentos aderente ao padrão DMA a qualquer sistema de gerência de *workflow* aderente ao padrão WAPI. É importante ressaltar que essa integração é, no momento, inédita [AII 98a], representando, assim, um avanço significativo na atual realidade destas áreas.

A forma escolhida para representar a implementação realizada é o *diagrama de seqüência*. Esse tipo de diagrama, introduzido pela UML 1.0, permite mostrar a seqüência de troca de mensagens entre objetos, para atender a uma determinada funcionalidade identificada [FOW 97]. Dessa forma, fica explícita a interação entre os objetos, permitindo ilustrar claramente, neste caso, o papel desempenhado por cada elemento DMA ou WAPI para atender cada uma das funcionalidades da camada de serviços.

É importante ressaltar que um diagrama de seqüência não permite representar em detalhes o fluxo de controle de cada método, como decisões ou repetições. Desta forma, nas situações onde o diagrama não conseguir expressar algum detalhe relevante, ele será explicado no texto que acompanha cada diagrama. Além disso, para manter a legibilidade do diagrama, somente os parâmetros mais relevantes são ilustrados nas invocações dos métodos. Pela mesma razão, algumas invocações de métodos DMA são omitidas, como as chamadas *QueryInterface* e *Release*, utilizadas para a obtenção e liberação de interfaces COM, assim como são omitidas as classes de listas e enumerações de objetos DMA e WAPI, cuja inclusão nos diagramas pouco contribuiria para a compreensão do mecanismo de integração entre os padrões.

É importante observar, ainda, algumas convenções especiais assumidas nesta representação dos métodos. Como exposto no capítulo 3, a WAPI não é definida através de um conjunto de classes; a definição atual constitui-se de um conjunto de funções em linguagem C. Assim, não é possível representar objetos WfMC nos diagramas, pois tais, a rigor, não existem. De modo a resolver essa questão, foi introduzido nos diagramas um objeto denominado *WfMC*. Esse objeto recebe todas as chamadas de função WAPI. Ele, no entanto, não existe em termos de implementação, e sim somente como artifício para a representação de uma chamada de função procedural em um diagrama de objetos.

Uma segunda observação refere-se ao padrão DMA. Conforme explicado no capítulo 4, o modelo de objetos DMA baseia-se no COM, da Microsoft. Nesse modelo, os métodos não são invocados sobre objetos de uma determinada classe, mas sim sobre determinadas *interfaces* (agrupamentos de métodos) que aquela classe suporta [ORF 96]. A utilização de objetos COM consiste então, na prática, na utilização das interfaces definidas, o que torna os próprios conceitos de classe e objeto de difícil percepção dentro do software desenvolvido. Para tornar o mais claro possível esse mecanismo, optou-se por referenciar tanto o nome da classe do objeto quanto a interface à qual pertence cada método ilustrado. Assim, os diagramas a seguir incluem o nome da classe, enquanto a interface à qual cada método pertence é definida no texto que acompanha o diagrama.

Cada uma das próximas subseções mostrará a implementação de um método da camada de serviços, indicando qual das funcionalidades levantadas no início da seção 7.2 está sendo atendida.

7.2.2.1 Conexão ao Sistema

O serviço de conexão ao sistema é realizado pelo método *Login* da classe *Server*. A figura 7.3 mostra a implementação do método *Login*.

A invocação do método *Login* deve conter, como parâmetros, o nome do sistema de documentos (objeto *System*) e do espaço de documentos (objeto *DocSpace*) a serem conectados, assim como o nome do sistema de gerência de *workflow*, além de um nome de usuário e de uma senha. Esses valores podem ser entradas diretamente pelo usuário, através da interface de uma aplicação cliente, ou obtidos a partir de algum arquivo de configuração da máquina cliente. É assumido que um mesmo conjunto {nome de usuário, senha} é válido tanto para a conexão com o DMS quanto para a conexão com o WFMS. Uma vez que os padrões WAPI e DMA não oferecem, atualmente, recursos para a definição de usuários e senhas, assume-se aqui que uma convenção interna da organização mantém esses valores idênticos.

A primeira chamada realizada por esse método é à função DMA *dmaConnectSystemManager*. Essa função é a única chamada DMA que não se baseia em COM, constituindo-se em uma função em linguagem C. Ela representa o ponto de entrada no sistema DMA, retornando uma interface COM para o objeto *System Manager*. Obtendo-se um objeto *System Manager*, é possível conectar-se a um objeto *System* através do método *ConnectSystem* da interface *IdmaSystemManager*. Com um objeto *System*, pode-se invocar o método *AuthenticateUser* da interface

IdmaAuthenticate para passar o nome do usuário e sua senha (a disponibilização dessa interface é opcional, podendo variar entre as diferentes implementações). Então, é possível conectar-se a um objeto *DocSpace* através do método *ConnectDocSpace* da interface *IdmaSystem*. Similarmente ao objeto *System*, com um objeto *DocSpace* pode-se invocar o método *AuthenticateUser* da interface *IdmaAuthenticate* para passar o nome do usuário e sua senha (a disponibilização dessa interface é igualmente opcional).

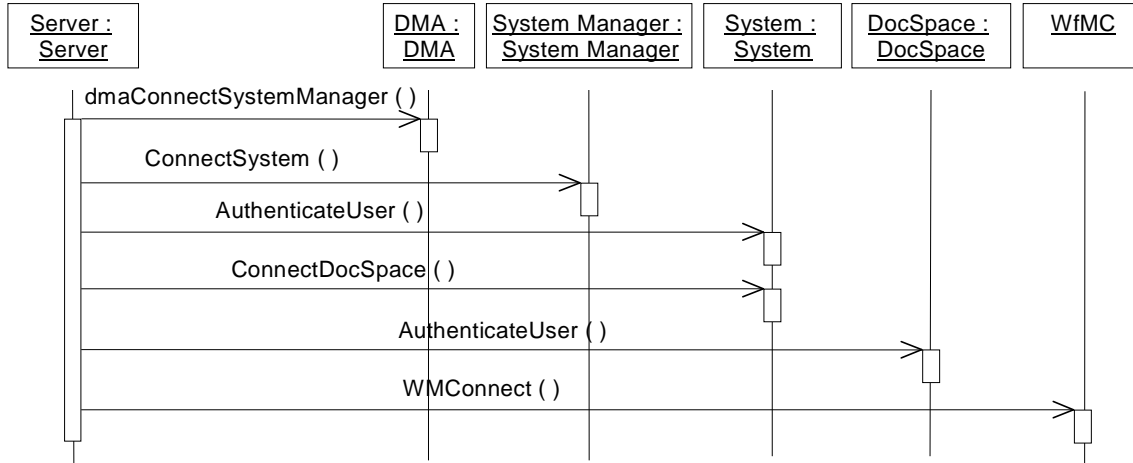


FIGURA 7.3 - Diagrama de seqüência para o método *Login*

A conexão com o sistema de gerência de *workflow* é feita através da chamada da função WAPI *WfMCConnect*. O nome do usuário e sua senha são passados nessa chamada.

Como resultados dessas chamadas, são obtidos um apontador COM para um objeto *System Manager*, um apontador COM para um objeto *System*, um apontador COM para um objeto *DocSpace* e um *handle* (inteiro) para o WFMS. Esses valores são, então, armazenados em atributos da classe *Server*, e podem ser acessados pelos demais objetos através dos métodos *GetDocSpacePointer* e *GetWFMSHandle*. Esses métodos serão utilizados, posteriormente, por todos os métodos que precisarem fazer conexão ao DMS ou ao WFMS, respectivamente. Contudo, por razões de legibilidade, a invocação desses métodos será omitida dos diagramas de seqüência correspondentes.

7.2.2.2 Desconexão do Sistema

O serviço de desconexão do sistema é realizado pelo método *Logout* da classe *Server*. A figura 7.4 mostra a implementação do método *Logout*.

A invocação do método *Logout* não necessita de parâmetro algum. São chamados, em seqüência, para desconectar-se do DMS, o método *Release* da interface *IdmaDocSpace*, o método *Release* da interface *IdmaSystem* e o método *Release* da interface *IdmaSystemManager*.

A desconexão com o sistema de gerência de *workflow* é feita através da chamada da função WAPI *WMDisconnect*.

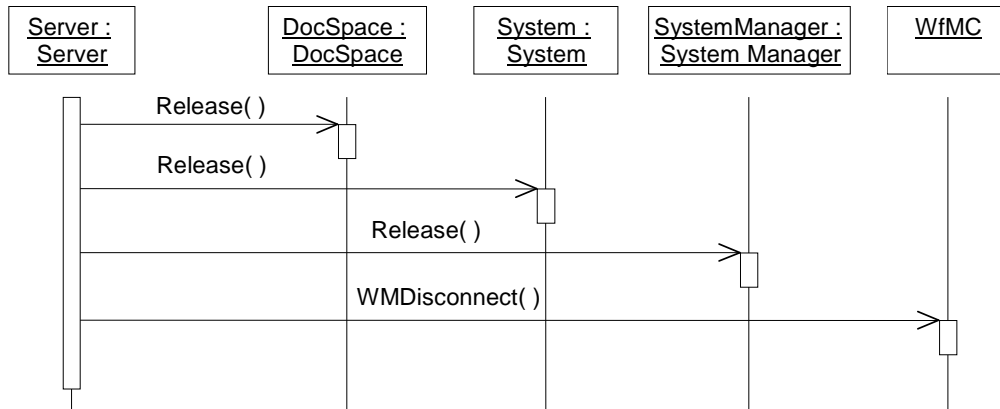


FIGURA 7.4 - Diagrama de seqüência para o método *Logout*

7.2.2.3 Obtenção dos processos a disparar

Um serviço importante é permitir que o participante recupere todas as definições de processo que ele seja capaz de disparar. Isso significa, via de regra, que este participante pertence ao papel encarregado de executar a primeira atividade desta definição de processo. Esse serviço é realizado pelo método *GetProcessDefinitions* da classe *Server*. A figura 7.5 mostra a implementação deste método.

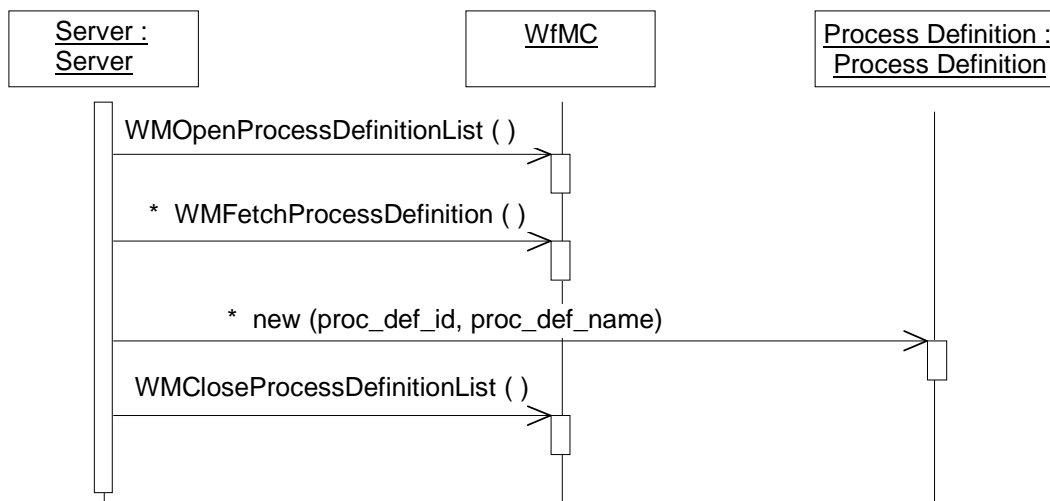


FIGURA 7.5 - Diagrama de seqüência para o método *GetProcessDefinitions*

A invocação do método *GetProcessDefinitions* não necessita de parâmetros. Inicialmente, é chamada a função WAPI *WMOpenProcessDefinitionList*. Essa função retorna um *handle* de consulta, que deve ser utilizado para percorrer a lista de definições de processo encontradas. Para cada elemento da lista, deve ser chamada a função *WMFetchProcessDefinition*, que efetivamente recupera a definição de processo.

Para cada definição de processo recuperada, é criado um objeto correspondente na camada de serviços. Isso é ilustrado pela invocação do método *new* sobre o objeto *Process Definition*. Entre os parâmetros passados na criação deste objeto estão o identificador da definição de processo (parâmetro *proc_def_id*) e nome da definição de processo (parâmetro *proc_def_name*), conforme armazenados no WFMS. Os objetos criados podem, a partir de então, ser mostrados em uma interface para o usuário, que assim poderá, em um momento posterior, disparar algum desses processos – provocando a invocação, então, do método *StartProcess*.

Por final, é chamada a função WAPI *WMCloseProcessDefinitionList*, encerrando a pesquisa junto ao WFMS.

7.2.2.4 Disparo de um processo

Outro serviço identificado é o disparo (início) de um processo. Esse serviço é realizado pelo método *StartProcess* da classe *Process Definition*. A figura 7.6 mostra a implementação deste método.

O método *StartProcess* recebe, como parâmetro, um nome para a instância de processo que será criada. Inicialmente, é chamada a função WAPI *WMCreateProcessInstance*, tendo como parâmetros o identificador da definição de processo (parâmetro *proc_def_id*, atributo do objeto *Process Definition*) e o nome para a instância a ser criada (parâmetro *proc_inst_name*, recebido na chamada do método *StartProcess*). Essa função retorna um identificador de instância de processo temporário. Em seqüência, é chamada a função *WMStartProcess*, tendo como parâmetro o identificador temporário obtido (parâmetro *proc_inst_id*) e retornando um identificador de instância de processo definitivo (*new_proc_inst_id*).

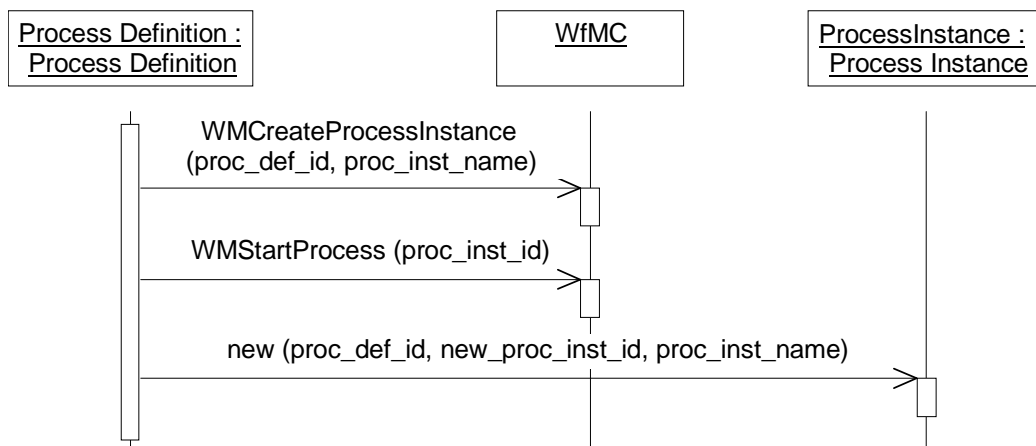


FIGURA 7.6 - Diagrama de seqüência para o método *StartProcess*

É, então, criado um objeto na camada de serviços correspondente à instância de processo criada. Isso é ilustrado pela invocação do método *new* sobre o objeto *Process Instance*. Entre os parâmetros passados na criação do objeto estão o identificador da

definição de processo (parâmetro *proc_def_id*) e o nome (parâmetro *proc_inst_name*) e o identificador definitivo da instância de processo (parâmetro *new_proc_inst_id*).

7.2.2.5 Obtenção da lista de trabalho

Outro serviço identificado é a recuperação da lista de trabalho do participante. A lista de trabalho, conforme [WMC 97], é uma estrutura que agrega todos os itens de trabalho atribuídos a um participante. Esse serviço é realizado pelo método *GetWorkList* da classe *Server*. A figura 7.7 mostra a implementação deste método.

O método *GetWorkList* não recebe parâmetros. Inicialmente, é chamada a função WAPI *WMOpenWorkList*. Essa função retorna um *handle* de consulta, que deve ser utilizado para percorrer a lista de itens de trabalho encontrados. Para cada elemento da lista, deve ser chamada a função *WMFetchWorkItem*, que efetivamente recupera o item de trabalho.

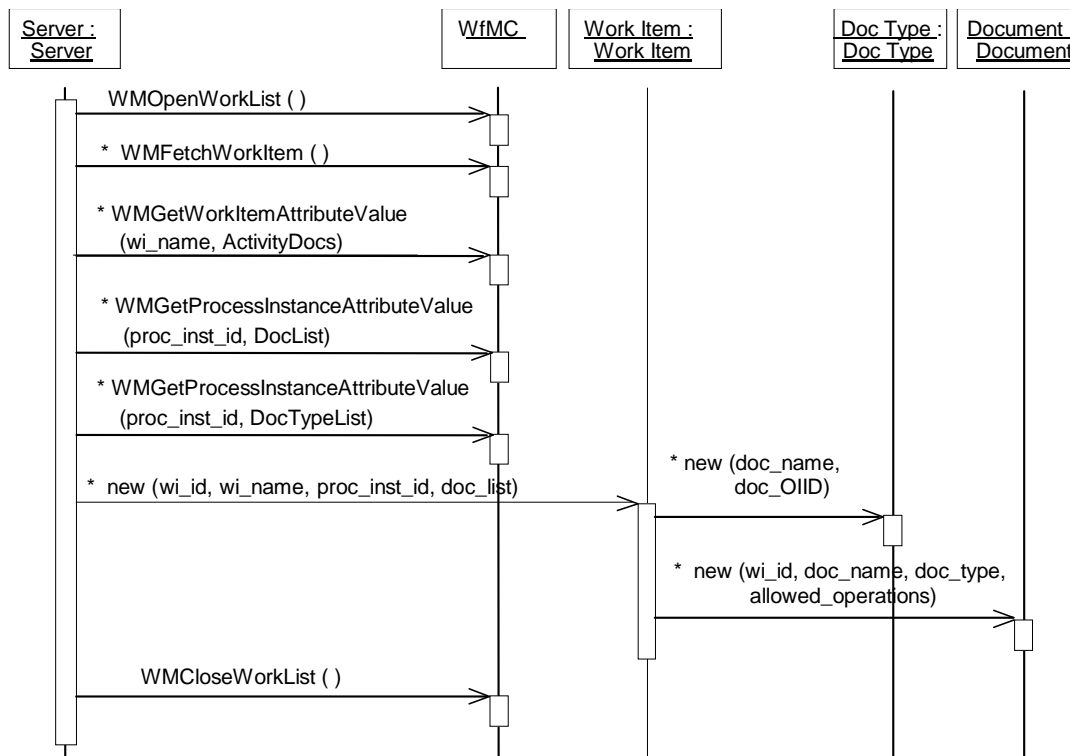


FIGURA 7.7 - Diagrama de seqüência para o método *GetWorkList*

Nesta implementação, foi decidido que o método *GetWorkList* retornará, além da descrição do item de trabalho em si, uma lista com os nomes dos documentos que esse item de trabalho manipula. Trata-se apenas da relação dos documentos, e não efetivamente de suas instâncias. Assim, é preciso invocar funções para recuperar esse dado que, como ilustrado na seção 6.2.1.3, está armazenado na variável do processo de *workflow ActivityDocs*. Para recuperar o valor dessa variável, a função WAPI *WMGetWorkItemAttributeValue* deve ser chamada, para cada item de trabalho encontrado. Essa função recupera uma lista de *DocIds* e de *Allowed Operations*.

É preciso, então, relacionar os *DocIds* obtidos com a lista de documentos manipulados pelo processo, armazenada no dado relevante de processo *DocList*, definido na seção 6.2.1.2. É preciso, também, poder relacionar cada um desses documentos com seu tipo, informação essa armazenada no dado relevante de processo *DocTypeList*, definido na seção 6.2.1.1. Para tal, é preciso invocar a função WAPI *WMGetProcessInstanceAttributeValue*, solicitando o retorno dessas informações.

Então, para cada item de trabalho recuperado, é criado um objeto correspondente na camada de serviços. Isso é ilustrado pela invocação do método *new* sobre o objeto *Work Item*. Entre os parâmetros passados na criação do objeto estão o nome e o identificador do item de trabalho, o identificador da instância de processo a que pertence, e uma lista contendo o nome, o tipo e as operações possíveis sobre cada documento manipulado pelo item de trabalho. Observe-se que, no próprio método construtor de *Work Item*, são invocados os métodos *new* das classes *Doc Type* e *Document*. Para a construção do objeto *Doc Type*, são passados como parâmetros o seu nome e o seu OIID. Para a construção do objeto *Document*, são passados como parâmetros o seu nome e as operações possíveis sobre o documento.

Os objetos *Work Item* criados podem, a partir de então, ser mostrados em uma interface para o usuário, que assim poderá, em um momento posterior, disparar algum desses itens de trabalho – causando, então, a invocação do método *Start* da classe *Work Item*.

Por final, é chamada a função WAPI *WMCloseWorkList*, encerrando a pesquisa junto ao WFMS.

7.2.2.6 Disparar um item de trabalho

Outro serviço identificado é o disparo (início) de um item de trabalho. Esse serviço é realizado pelo método *Start* da classe *Work Item*. A figura 7.8 mostra a implementação deste método.

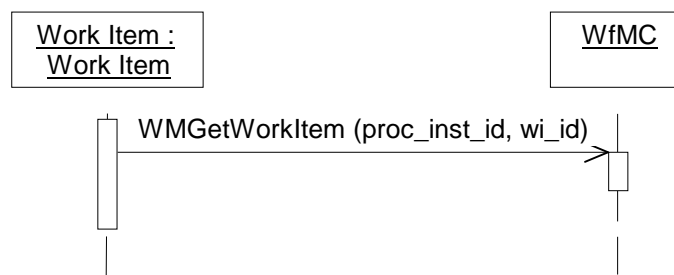


FIGURA 7.8 - Diagrama de seqüência para o método *Start*

O método *Start* não recebe parâmetros. Seu código consiste apenas da chamada à função WAPI *WMGetWorkItem*, passando como parâmetros o identificador da instância de processo (parâmetro *proc_inst_id*) e o identificador do item de trabalho (parâmetro *wi_id*).

O disparo de um item de trabalho pode ser acompanhado por ações internas do WFMS como, por exemplo, o início da contabilização do tempo para realizá-lo. Essas ações, no entanto, não são ainda padronizadas pela WfMC, ficando a cargo de cada fornecedor de sistemas de *workflow* a decisão de oferecê-las em seus produtos.

7.2.2.7 Criar um documento

Para a criação de um novo documento no DMS, deve ser invocado o método *CreateDocument* da classe *Document*. A figura 7.9 mostra a implementação desse método.

De forma geral, o método *CreateDocument* pode ser dividido em três partes. Na primeira, são criadas as estruturas DMA necessárias à gerência de versões, conforme apresentado na seção 4.6; na segunda, é criado o novo objeto *DocVersion*, a partir do objeto *Doc Type* associado; e, por final, na terceira, o novo objeto *DocVersion* é inserido na lista de versões, é criado o objeto *Version* da camada de serviços e é atualizada a variável do processo de *workflow* *DocInstanceList*. A segunda parte desse método, devido a sua complexidade, foi isolada no método *CreateDocumentFromType*, da classe *Doc Type*. Esse método será explicado no final desta seção.

Ao ser invocado, o método *CreateDocument* recebe três parâmetros: o OIID do objeto *Doc Type* associado ao documento que será criado (*DocTypeOIID*), o identificador do documento que será criado (*DocId*) e o identificador da instância de processo relacionada (*proc_inst_id*). O primeiro desses parâmetros será utilizado na invocação do método *CreateDocumentFromType*, para permitir a localização do *template* de documento a ser utilizado, enquanto os dois restantes serão utilizados para, ao final da execução do método, atualizar a variável do processo de *workflow* *DocInstanceList*.

Como citado anteriormente, as primeiras chamadas desse método servem para criar os elementos DMA relativos à gerência de documentos. Conforme exposto na seção 4.6, o modelo de versões da DMA envolve objetos *ConfigurationHistory*, *VersionSeries*, *VersionDescription* e *Reservation*. Esses objetos são criados através da invocação do método *CreateObject* da interface *IdmaDocSpace*. Cada uma dessas chamadas retorna o OIID do respectivo objeto DMA criado, o qual é utilizado nas chamadas seguintes. Por final, é invocado o método DMA *SetReserveNext*, que reserva a esta conexão o direito de criar um novo objeto *DocVersion* [AII 97].

Após a criação desses objetos, pode-se criar, então, um objeto *DocVersion*. A criação desse objeto, como já comentado, é responsabilidade do método *CreateDocumentFromType* da classe *Doc Type*. Esse método, explicado ao final da seção, retorna o OIID do objeto *DocVersion* criado.

Uma vez criado o objeto *DocVersion*, e de posse de seu OIID, é invocado o método DMA *SetCheckIn*. Esse método inscreve o novo objeto *DocVersion* como uma versão do objeto *VersionSeries* anteriormente criado. Ainda, é criado um objeto *Version* da camada de serviços, recebendo como parâmetros os atributos e o conteúdo do documento criado. Por final, é invocada a função WAPI

WMAssignProcessInstanceAttributeValue, atualizando a variável *DocInstanceList* com o OIID do objeto *VersionSeries* recém-criado. É importante ressaltar que não é necessário armazenar o OIID dos objetos *DocVersion* nessa variável, pois esses valores podem ser facilmente acessados através do objeto *VersionSeries* ao qual pertencem.

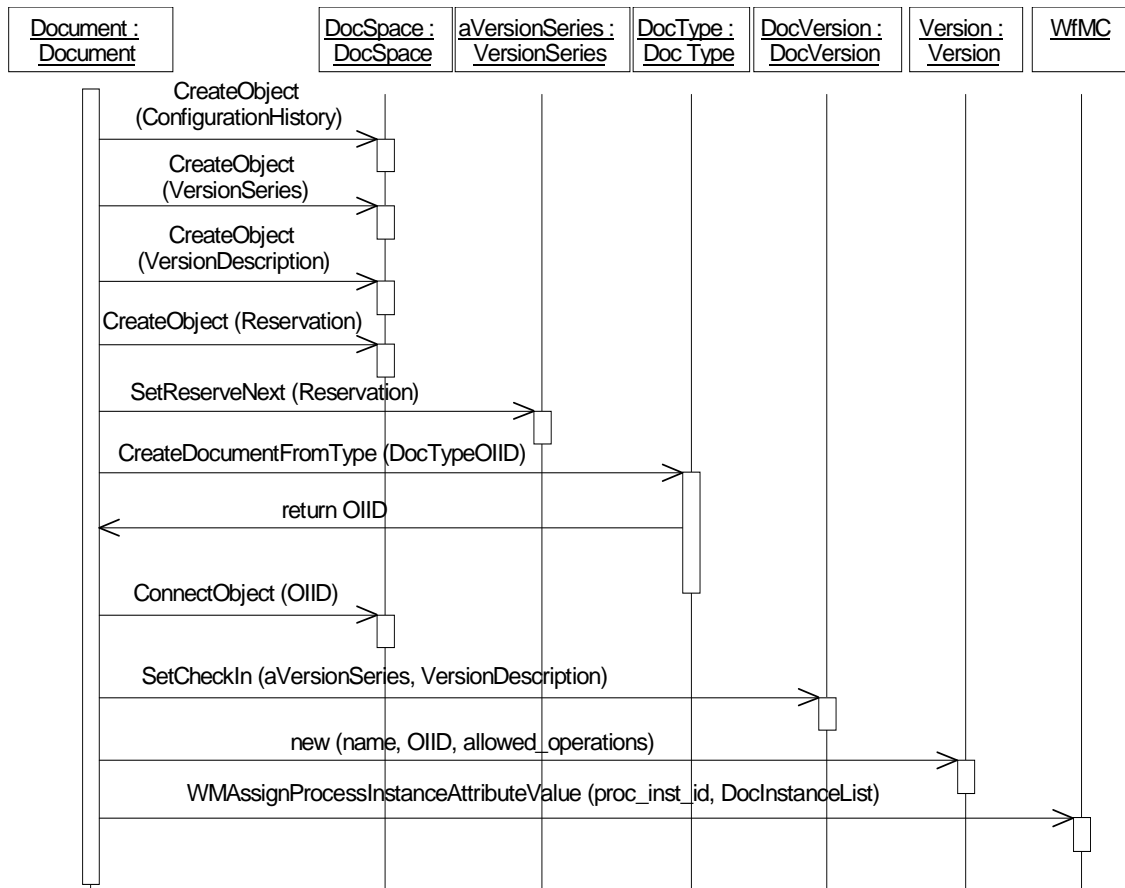


FIGURA 7.9 - Diagrama de seqüência para o método *CreateDocument*

Passe-se, então, à análise do método *CreateDocumentFromType* da classe *Doc Type*. Esse método, apresentado na figura 7.10, pode ser dividido em duas partes: na primeira, o conteúdo do objeto *Doc Type* é recuperado do DMS; na segunda, é criado um objeto *DocVersion* que possui esse mesmo conteúdo. Na figura, os objetos DMA ligados ao objeto *Doc Type* estão designados pelo prefixo 'DT', simbolizando '*Doc Type*', enquanto os objetos DMA ligados ao novo documento estão designados pelo prefixo 'ND', simbolizando '*New Document*'.

Como afirmado anteriormente, o método recebe como parâmetro o OIID do objeto *Doc Type* (*DocTypeOIID*). Esse parâmetro é utilizado para recuperar o objeto, através da invocação do método *ConnectObject* da interface *IdmaDocSpace*. Após a conexão, devem ser buscados, sucessivamente, os objetos *Rendition* e *ContentTransfer* associados ao objeto, através, respectivamente, de chamadas aos métodos *GetRendition* e *GetContentTransfer*. Através, então, da invocação do método *CopyToResource*, o conteúdo do objeto pode ser transferido para um arquivo do sistema operacional, representado pela variável *local_file*.

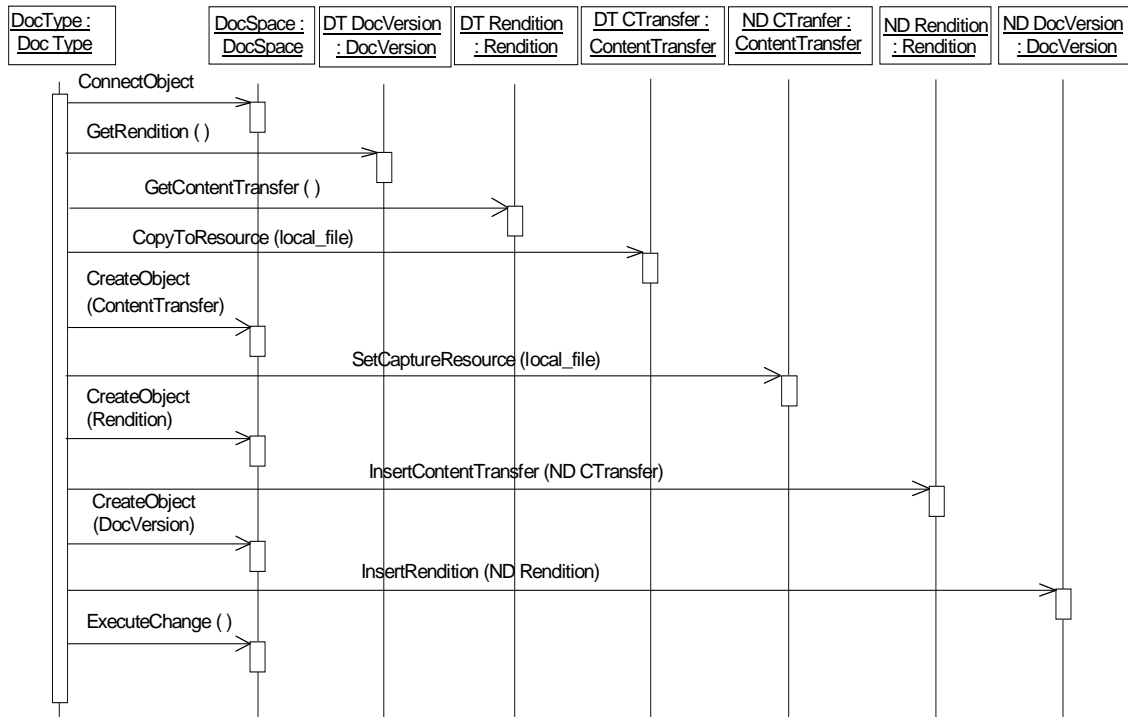


FIGURA 7.10 - Diagrama de seqüência para o método *CreateDocumentFromType*

A criação do novo documento obedece aos seguintes passos. Inicialmente, é criado um objeto *ContentTransfer*, através da invocação do método *CreateObject*. Esse objeto recebe, então, através de uma chamada ao método *SetCaptureResource*, o conteúdo do objeto *DocType* relacionado, que está armazenado no arquivo do sistema operacional *local_file*. Então, é criado um objeto *Rendition*, pela invocação do método *CreateObject*, e, na seqüência, o objeto *ContentTransfer* anteriormente criado é associado a ele, pela invocação do método *InsertContentTransfer*. Então, é criado um objeto *DocVersion*, ao qual é ligado o objeto *Rendition* criado, pela invocação do método *InsertRendition*. Por final, é executado o método *ExecuteChange*, que consolida todas as alterações promovidas, e recupera o OIID do objeto *DocVersion* criado.

Por final, de forma a tornar persistente no DMS o novo documento criado, deve ser invocado o método *ExecuteChange* da interface *IdmaDocSpace*. Além disso, esse método permite recuperar o OIID gerado para o objeto *DocVersion* criado. Nesse momento, então, esse OIID pode ser retornado, como valor de saída, ao método *CreateDocument*, que invocou esse método.

7.2.2.8 Abrir uma versão do documento

A abertura de um documento, no contexto dos serviços propostos, é uma operação que envolve duas operações distintas. Uma vez que diversas versões de um mesmo documento podem ser geradas e armazenadas, é preciso, primeiramente, obter o conjunto de versões atualmente existente para aquele documento. Feito isso, então, pode-se selecionar uma determinada versão e solicitar a sua abertura. Para tal, logo, a camada de serviços oferece dois métodos distintos: o método *GetVersionList*, da classe *Document*, responsável por obter a lista de versões para um certo documento, e o

método *Open*, da classe *Version*, responsável por retornar o conteúdo de uma versão específica do documento.

A implementação do método *GetVersionList* está ilustrada na figura 7.11. Esse método recebe como parâmetros o identificador da instância de processo relacionada (*proc_inst_id*) e o identificador do documento (*DocId*). Inicialmente, é preciso recuperar o OIID do objeto *VersionSeries* associado a esse documento. Esse valor está armazenado na variável de processo de *workflow* *DocInstanceList*, descrita na seção 6.2.1.2. Para tal, é invocada a função WAPI *WMGetProcessInstanceAttributeValue*, utilizando como parâmetros o identificador da instância de processo e o identificador de documento recebidos. De posse desse OIID, então, o objeto *VersionSeries* pode ser recuperado do DMA através da invocação do método *ConnectObject*. Após, todos os objetos *VersionDescription* ligados ao objeto *VersionSeries* são recuperados, pela invocação do método *GetVersionDescription*. Os objetos *VersionDescription* armazenam, em sua propriedade *dmaProp_Version*, um apontador para o objeto *DocVersion* ao qual estão relacionados. O valor dessa propriedade pode ser acessado através da invocação do método *GetPropValObjectById*, da interface *IdmaProperties*. Assim, é possível recuperar o objeto *DocVersion* relacionado.

Tendo recuperado o objeto *DocVersion*, pode-se obter o seu OIID através da invocação do método *GetPropValOIIDById*. Por final, então, são criados os objetos *Version* da camada de serviços, recebendo como parâmetros o nome do documento, seu OIID e as operações sobre ele permitidas. É conveniente recordar que o nome do documento e as operações sobre ele permitidas já haviam sido anteriormente recuperados, quando da invocação do método *GetWorkList*.

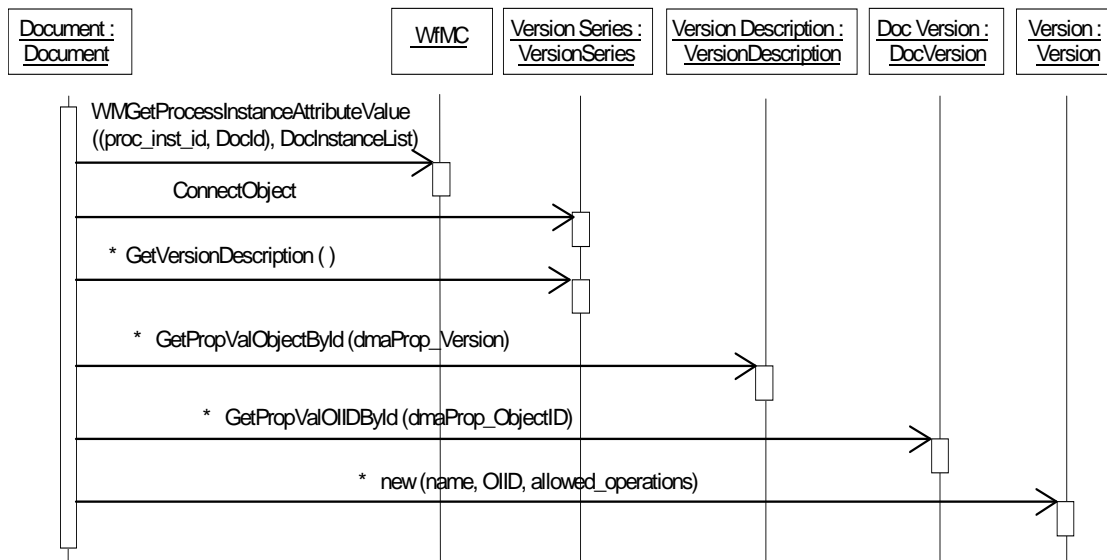


FIGURA 7.11 - Diagrama de seqüência para o método *GetVersionList*

Ao final, é retornado a quantidade de versões do documento recuperadas. Um valor de retorno '0' (zero) indica que esse documento ainda não possui nenhuma versão

criada no DMS. O aplicativo chamador pode então, por exemplo, invocar o método *CreateDocument* para criar uma primeira versão de documento.

Por sua vez, o método *Open*, da classe *Version*, tem sua implementação ilustrada na figura 7.12. O método *Open* recebe como parâmetro o OIID do objeto *VersionSeries*. Primeiramente, são invocados métodos relativos à gerência de versões. O modelo DMA adota uma política de *check-in* de documentos [AII 97], similar à descrita em [KHO 95]. Assim, um aplicativo, ao solicitar a abertura de um documento, deve *reservar* o direito de gerar a próxima versão deste documento. Essa reserva é implementada através da criação de um objeto *Reservation*, através da invocação do método *CreateObject*, e pela subsequente invocação do método *SetReserveNext*. É importante observar que, com tal mecanismo, o DMS garante o controle de concorrência de acesso às versões.

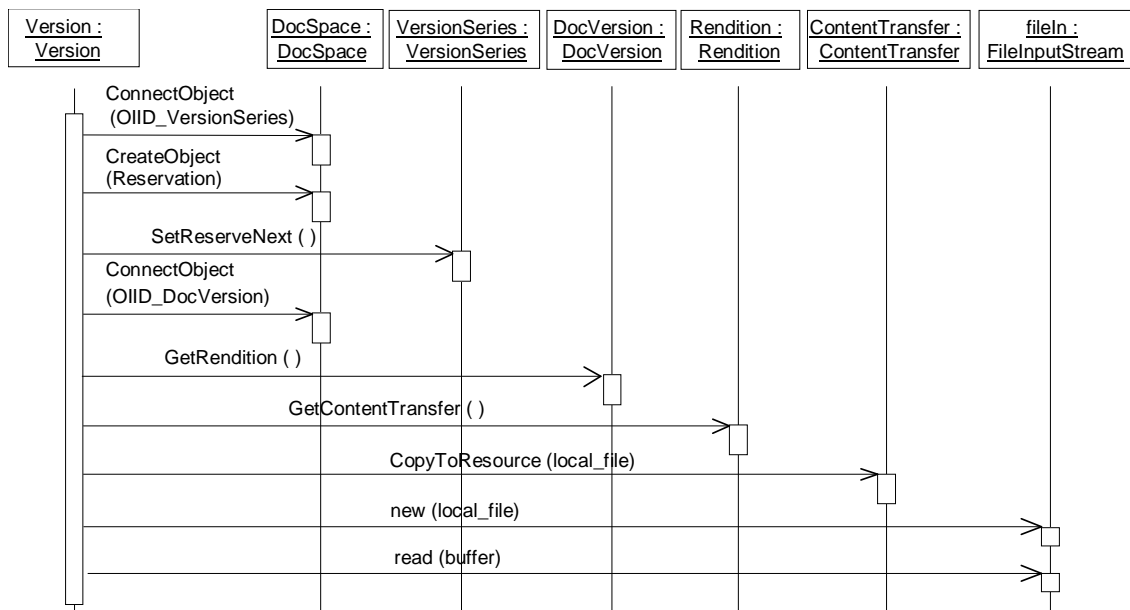


FIGURA 7.12 - Diagrama de seqüência para o método *Open*

Após a execução da reserva, o conteúdo da versão pode ser recuperado. Para tal, inicialmente, é invocado o método *ConnectObject*, tendo como parâmetro o OIID do objeto *DocVersion* a ser recuperado. A partir do objeto *DocVersion*, pode-se ter acesso aos objetos *Rendition* a ele ligados, pela chamada do método *GetRendition*. Por sua vez, a partir de um objeto *Rendition*, pode-se ter acesso aos objetos *ContentTransfer* a ele ligados, pela chamada do método *GetContentTransfer*. Através da invocação do método *CopyToResource* pode-se, então, recuperar o conteúdo do documento, colocando-o em um arquivo do sistema operacional, representado na figura 7.12 pela variável *local_file*. Esse arquivo pode, então, ser aberto através de objetos da classe Java *FileInputStream* [WEB 97]. O método *read* desta classe permite colocar o conteúdo do arquivo em uma variável de memória, representada na figura 7.12 pela variável *buffer*.

O conteúdo recuperado é, então, retornado como parâmetro de saída do método. O aplicativo que invocou o método *Open* pode então, por exemplo, exibir o conteúdo em uma interface de usuário.

7.2.2.9 Salvar uma versão do documento

Após a abertura de uma versão de um documento, e caso o participante possua direitos para tal, o seu conteúdo pode, eventualmente, ser alterado. O participante pode, então, decidir tornar persistentes as modificações realizadas, sobrescrevendo o conteúdo da versão atual. Para tais situações, a camada de serviços oferece o método *Save*, da classe *Version*. A figura 7.13 mostra a implementação deste método.

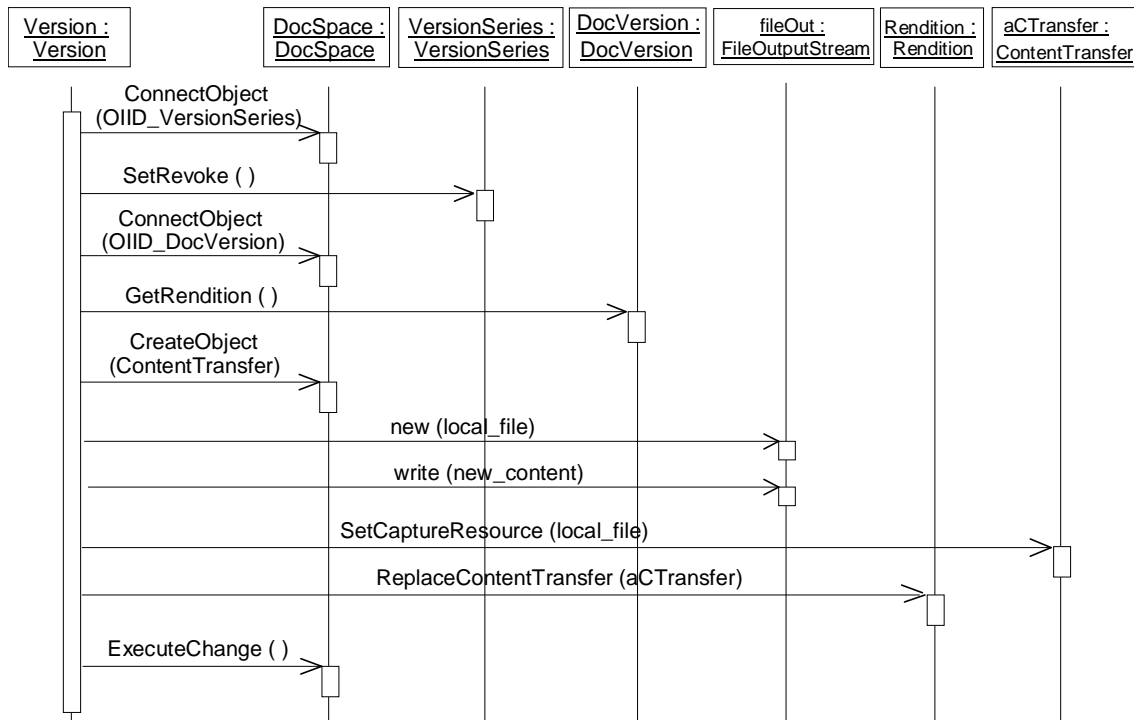


FIGURA 7.13 - Diagrama de seqüência para o método *Save*

O método *Save* recebe como parâmetros o OIID do objeto *VersionSeries* (OID_VersionSeries) e uma variável possuindo o novo conteúdo da versão (new_content). Primeiramente, são invocados métodos relativos à gerência de versões. Devido às chamadas de métodos realizadas na abertura da versão do documento, o DMS espera, a princípio, que seja criada uma nova versão do documento. Uma vez que essa não é a intenção do método *Save*, é necessário invocar o método DMA *SetRevoke*. Esse método indica uma *desistência* do direito de criar uma nova versão do documento, direito esse obtido na invocação do método *Open*.

Após, procede-se com a alteração do conteúdo da versão do documento. Para tal, inicialmente, é invocado o método *ConnectObject*, tendo como parâmetro o OIID do objeto *DocVersion* a ser alterado. A partir do objeto *DocVersion*, pode-se ter acesso aos objetos *Rendition* a ele ligados, pela chamada do método *GetRendition*.

No modelo DMA, a alteração do conteúdo dá-se pela substituição do objeto *ContentTransfer* corrente por um novo, possuidor do novo conteúdo. Assim, é criado um novo objeto *ContentTransfer*, através da invocação do método *CreateObject*. É importante observar que os métodos atualmente definidos pela DMA exigem que o

novo conteúdo esteja armazenado em arquivos do sistema operacional. Assim, é gerado um arquivo temporário para armazenar o novo conteúdo, armazenado na variável *new_content*. O novo arquivo é criado através de objetos da classe Java *FileOutputStream* [WEB 97]. O método *write* dessa classe permite colocar o conteúdo da variável *new_content* em um arquivo, representada na figura 7.13 pela variável *local_file*.

Estando o novo conteúdo em um arquivo, pode-se invocar o método *SetCaptureResource* sobre o objeto *ContentTransfer* criado. Esse método recebe como parâmetro o arquivo *local_file* e coloca o seu conteúdo no respectivo objeto *ContentTransfer*. O passo final dessa substituição é invocar o método *ReplaceContentTransfer* sobre o objeto *Rendition*. Esse método descarta o objeto *ContentTransfer* correntemente ligado ao objeto *Rendition* e o substitui pelo objeto *ContentTransfer* passado como parâmetro, representado na figura 7.13 pelo objeto *aCTransfer*.

Por final, de forma a tornar persistentes no DMS as alterações realizadas, deve ser invocado o método *ExecuteChange* da interface *IdmaDocSpace*.

7.2.2.10 Criar uma nova versão de um documento

Após a abertura de uma versão de um documento, e caso o participante possua direitos para tal, o seu conteúdo pode, eventualmente, ser alterado. O participante pode, então, decidir tornar persistentes as modificações realizadas, criando uma nova versão do documento em questão. Para tal, a camada de serviços oferece o método *CreateVersion*, da classe *Version*. A figura 7.14 mostra a implementação deste método.

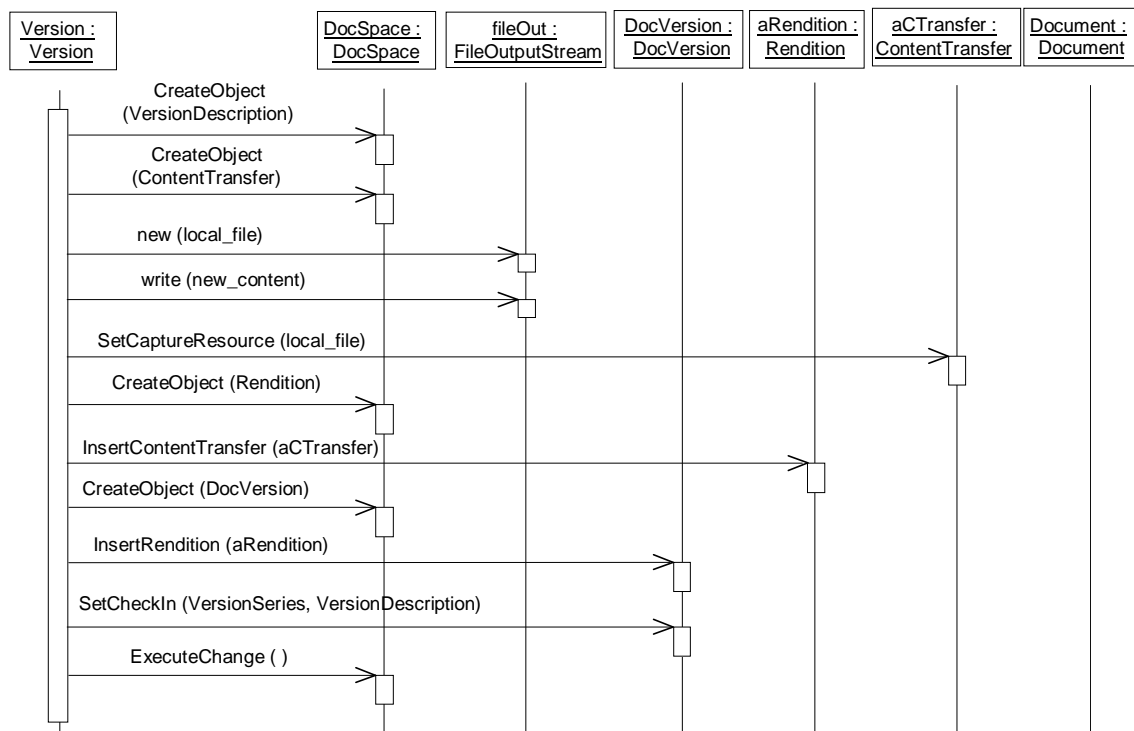


FIGURA 7.14 - Diagrama de seqüência para o método *CreateVersion*

O método *CreateVersion* recebe como parâmetros o OIID do objeto *VersionSeries* (OIID_VersionSeries) e uma variável possuindo o novo conteúdo da versão (*new_content*). Inicialmente, é criado um objeto *VersionDescription*, responsável por relacionar um objeto *VersionSeries* aos seus objetos *DocVersion*, conforme o modelo de versões DMA apresentado na seção 4.6.

Em seguida, são criados os objetos que irão compor a nova versão do documento. Assim, é criado um novo objeto *ContentTransfer*, através da invocação do método *CreateObject*. É importante observar que os métodos atualmente definidos pela DMA exigem que o conteúdo da nova versão esteja armazenado em arquivos do sistema operacional. Assim, é gerado um arquivo temporário para armazenar o novo conteúdo, armazenado na variável *new_content*. O novo arquivo é criado através de objetos da classe Java *FileOutputStream* [WEB 97]. O método *write* dessa classe permite colocar o conteúdo da variável *new_content* em um arquivo, representado na figura 7.14 pela variável *local_file*. Então, o objeto *ContentTransfer* pode receber o conteúdo do arquivo, através da invocação do método *SetCaptureResource*. Esse método recebe como parâmetro o arquivo *local_file*.

Após, deve ser criado um novo objeto *Rendition*. Para tal, é invocado o método *CreateObject*. Em seqüência, o método *InsertContentTransfer* é chamado, para associar o objeto *Rendition* criado ao objeto *ContentTransfer* passado como parâmetro. Então, é criado o novo objeto *DocVersion*, pela invocação do método *CreateObject*. O objeto *Rendition* criado é ligado ao novo objeto *DocVersion* através da invocação do método *InsertRendition*.

Então, é invocado o método *SetCheckIn*, recebendo como parâmetros os OIIDs dos objetos *VersionSeries* e *VersionDescription*. A execução desse método registra o objeto *DocVersion* criado como a mais nova versão do documento representado pelo objeto *VersionSeries*. Por final, de forma a tornar persistentes no DMS as alterações realizadas, deve ser invocado o método *ExecuteChange* da interface *IdmaDocSpace*.

7.2.2.11 Remover uma versão de um documento

Uma determinada versão de um documento pode ser removida do DMS através da invocação do método *Remove* da classe *Version*. A figura 7.15 mostra a implementação deste método.

O método *Remove* não recebe parâmetros. Inicialmente, invoca-se o método *ConnectObject* da interface *IdmaDocSpace*, passando como parâmetro o OIID do documento, e obtendo-se um objeto *DocVersion*. Então, é invocado sobre esse objeto *DocVersion* o método *SetDeletePending*, tendo como parâmetro o valor DMA_TRUE. Esse método registra que o documento atualmente conectado está marcado para exclusão. Para efetivar a exclusão, deve-se chamar o método *ExecuteChange*. Quando esse método for chamado, o objeto *DocVersion* existente e todos seus objetos relacionados (objetos *Rendition* e *ContentTransfer*) serão igualmente removidos do repositório do DMS.

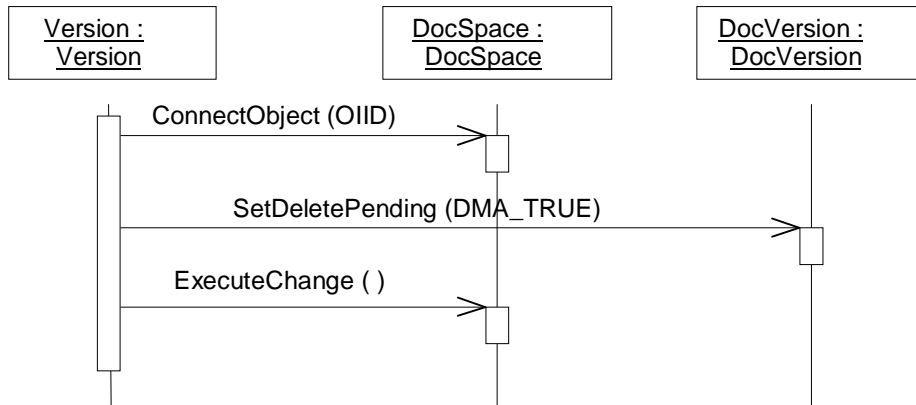


FIGURA 7.15 - Diagrama de seqüência para o método *RemoveVersion*

7.2.2.12 Completar um item de trabalho

Ao terminar as tarefas relativas a um determinado item de trabalho, um participante deve comunicar esse fato ao WFMS. Para disponibilizar essa funcionalidade, o método *Complete* da classe *Work Item* deve ser invocado. A figura 7.16 mostra a implementação deste método.

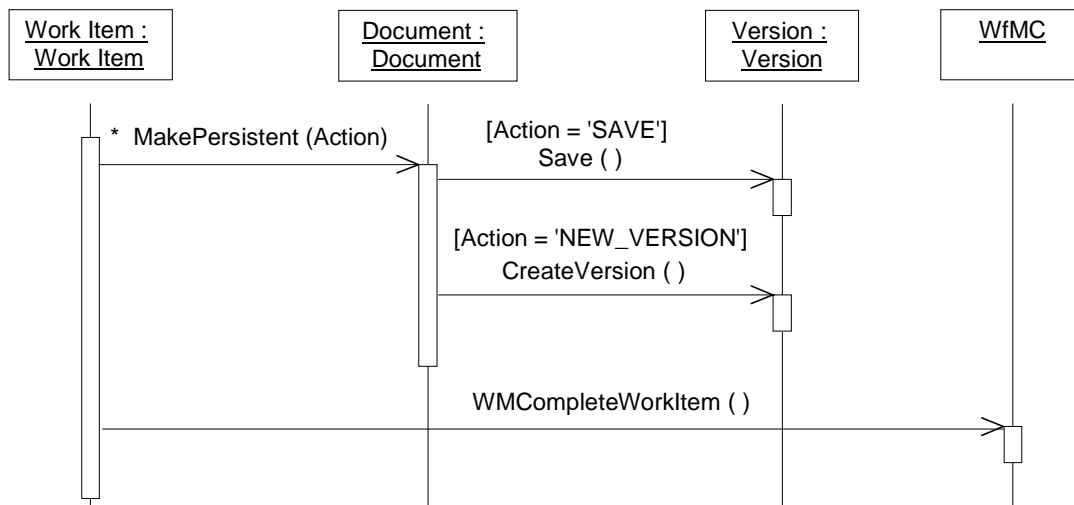


FIGURA 7.16 - Diagrama de seqüência para o método *Complete*

O método *Complete* não recebe parâmetros. Inicialmente, devem ser tomadas certas ações quanto aos documentos manipulados pelo item de trabalho que está sendo concluído, a fim de garantir a sua persistência e evitar problemas de concorrência de acesso. Assim, para cada documento ligado ao item de trabalho (objeto *Document*), será invocado o método *MakePersistent*, passando como parâmetro qual ação deve ser tomada sobre o documento (parâmetro *Action*). A camada de serviços suporta três possíveis ações: *SAVE*, *NEW_VERSION* e *IGNORE*. A ação *SAVE* indica que deve ser invocado o método *Save* sobre a versão atual do documento (objeto *Version*). A ação *NEW_VERSION* indica que deve ser invocado o método *CreateVersion* sobre a versão

atual do documento (objeto *Version*). A ação IGNORE não invoca método algum da WAPI ou DMA.

Por final, deve ser chamada a função WAPI *WMCompleteWorkItem*, passando como parâmetros o identificador da instância de processo a qual o item de trabalho pertence e o identificador do item de trabalho que está sendo terminado.

7.2.2.13 Delegar um item de trabalho

Um participante, por diversas razões, pode decidir delegar um determinado item de trabalho a outro participante. Para tal, o método *Reassign* da classe *Work Item* deve ser invocado. A figura 7.17 mostra a implementação deste método.

O método *Reassign* recebe como parâmetro o nome do participante para o qual o item de trabalho será delegado. Assim como no método *Complete*, devem ser tomadas certas ações quanto aos documentos manipulados pelo item de trabalho que está sendo concluído, a fim de garantir a sua persistência e evitar problemas de concorrência de acesso. Assim, para cada documento ligado ao item de trabalho (objeto *Document*), será invocado o método *MakePersistent*, passando como parâmetro qual ação deve ser tomada sobre o documento (parâmetro *Action*). A camada de serviços suporta três possíveis ações: SAVE, NEW_VERSION e IGNORE. A ação SAVE indica que deve ser invocado o método *Save* sobre a versão atual do documento (objeto *Version*). A ação NEW_VERSION indica que deve ser invocado o método *CreateVersion* sobre a versão atual do documento (objeto *Version*). A ação IGNORE não invoca método algum da WAPI ou DMA.

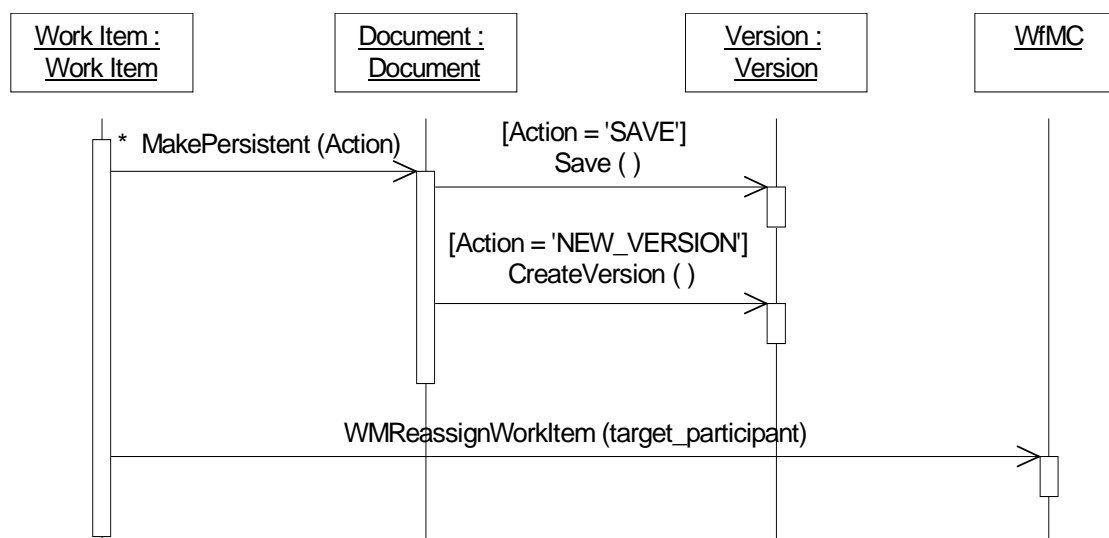


FIGURA 7.17 - Diagrama de seqüência para o método *Reassign*

Então, deve ser chamada a função WAPI *WMReassignWorkItem*, passando como parâmetros o identificador da instância de processo a qual o item de trabalho pertence, o identificador do item de trabalho que está sendo delegado, e o nome do participante que receberá o item de trabalho.

7.3 Aplicativo Cliente

O aplicativo cliente desenvolvido neste trabalho tem como objetivo ilustrar como as funcionalidades oferecidas pela camada de serviços podem ser utilizadas por um usuário final. É importante observar que, uma vez que a arquitetura proposta é de três camadas (*three-tier* [ORF 96]), a lógica do sistema integrado de gerência de documentos e gerência de *workflow* encontra-se na camada de serviços. O aplicativo cliente, além de controlar a interação com o usuário, basicamente apenas invoca os métodos da camada de serviços, relacionados na seção 7.2. Dessa forma, o volume de código presente no aplicativo cliente é relativamente pequeno, o que representa uma interessante vantagem, pois facilita o *download* do código e permite a execução do aplicativo mesmo em plataformas menos poderosas, como os *Network Computers*.

Ainda, é importante ressaltar que, embora seja um componente essencial da arquitetura proposta, o aplicativo cliente aqui apresentado não oferece recursos inovadores em relação a demais produtos comerciais atualmente existentes, como o Staffware97 [STA 98] ou o Panagon Visual WorkFlo [FIL 98a].

Essa seção é dividida em duas subseções. Primeiramente, a interface do aplicativo é descrita, enumerando suas funcionalidades. Após, é descrita a operação do aplicativo, ilustrando, através de telas, o seu comportamento em tempo de execução.

7.3.1 Descrição da interface

A figura 7.18 mostra a tela principal do aplicativo.

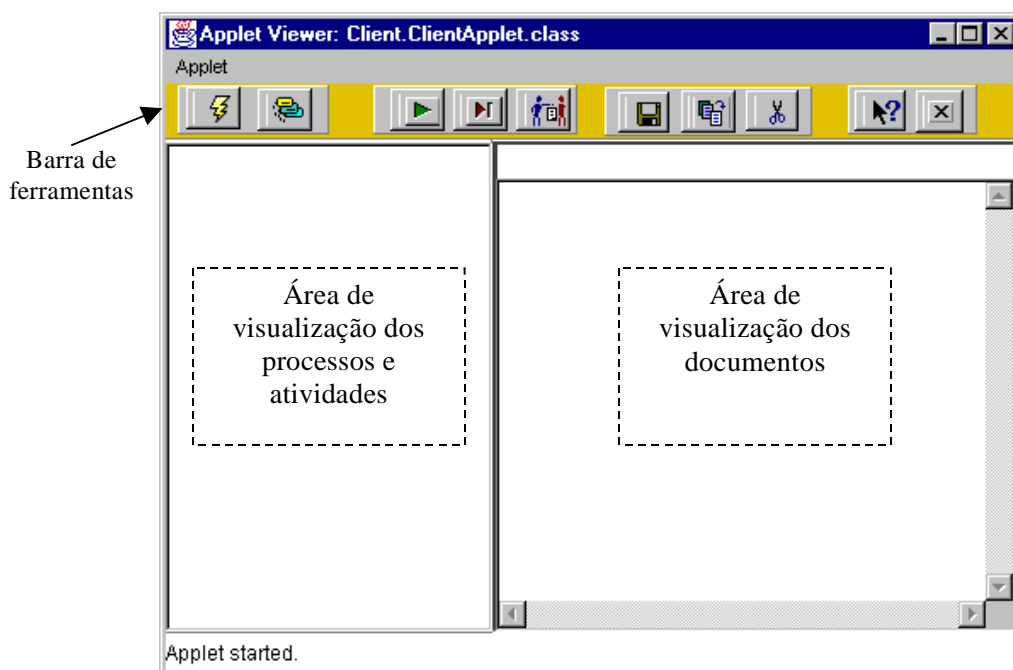


FIGURA 7.18 - Tela principal do aplicativo cliente

A tabela 7.2 apresenta a função de cada um dos elementos de interface identificados.

TABELA 7.2 - Função das partes identificadas da interface

Elemento	Função
Barra de ferramentas	Agrega botões cuja finalidade é invocar métodos da camada de serviços
Área de visualização dos processos e atividades	Abriga a lista de trabalho do participante e a lista de processos que podem ser por ele disparados
Área de visualização de documentos	Abriga o conteúdo do documento recuperado do sistema de gerência de documentos

A maioria dos elementos da barra de ferramentas está ligada a um determinado método da camada de serviços. A tabela 7.3 relaciona, para cada botão, sua funcionalidade e o método da camada de serviços por ele invocado, quando existente.

TABELA 7.3 - Descrição dos botões da barra de ferramentas

Botão	Função	Método da camada de serviços invocado
	Disparar uma nova instância da definição de processo selecionada	<i>ProcessDefinition.StartProcess</i>
	Alternar entre a visualização da lista de definições de processos e a visualização da lista de trabalho	<i>Server.GetProcessDefs</i> ou <i>Server.GetWorkList</i>
	Disparar o item de trabalho selecionado	<i>WorkItem.Start</i>
	Indicar o término da execução do item de trabalho selecionado	<i>WorkItem.Complete</i>
	Delegar o item de trabalho selecionado a outro participante	<i>WorkItem.Reassign</i>
	Salva a última versão do documento selecionado	<i>Version.Save</i>
	Cria uma nova versão do documento selecionado	<i>Version.CreateVersion</i>
	Remove a última versão do documento selecionado do repositório de documentos	<i>Version.Remove</i>
	Aciona a ajuda ao usuário	Nenhum
	Encerra o aplicativo e desconecta-se da camada de serviços	<i>Server.Logout</i>

É importante observar que nem todas funcionalidades do aplicativo estão disponíveis nos botões da barra de ferramentas. Algumas funcionalidades, como a conexão à camada de serviços, são invocadas automaticamente no início da execução do aplicativos, enquanto outras, como a abertura de um documento, são invocadas pela seleção de elementos da área de visualização dos processos e atividades. O mecanismo exato da invocação desses serviços será descrito na seção 7.2.2.

7.3.2 Operação do aplicativo

Quando iniciado, a primeira ação do aplicativo é solicitar, através de uma caixa de diálogo, a entrada das informações necessárias à conexão com a camada de serviços. Entre essas informações estão o nome do sistema de documentos, o nome do espaço de documentos e o nome do sistema de gerência de *workflow*. Além disso, devem ser entrados o nome do participante e sua senha. A seleção do botão “Ok” resulta na invocação do método *Login* da classe *Server*, apresentado na seção 7.2.2.1.

Uma vez que a conexão à camada de serviços tenha sido bem sucedida, a tela principal do aplicativo é apresentada ao usuário. Inicialmente, a área de visualização de processos e atividades é preenchida com a lista de definições de processos que o participante está autorizado a disparar. Essa lista é obtida com a invocação do método *GetProcessDefs* da classe *Server*. A figura 7.19 apresenta esta tela.

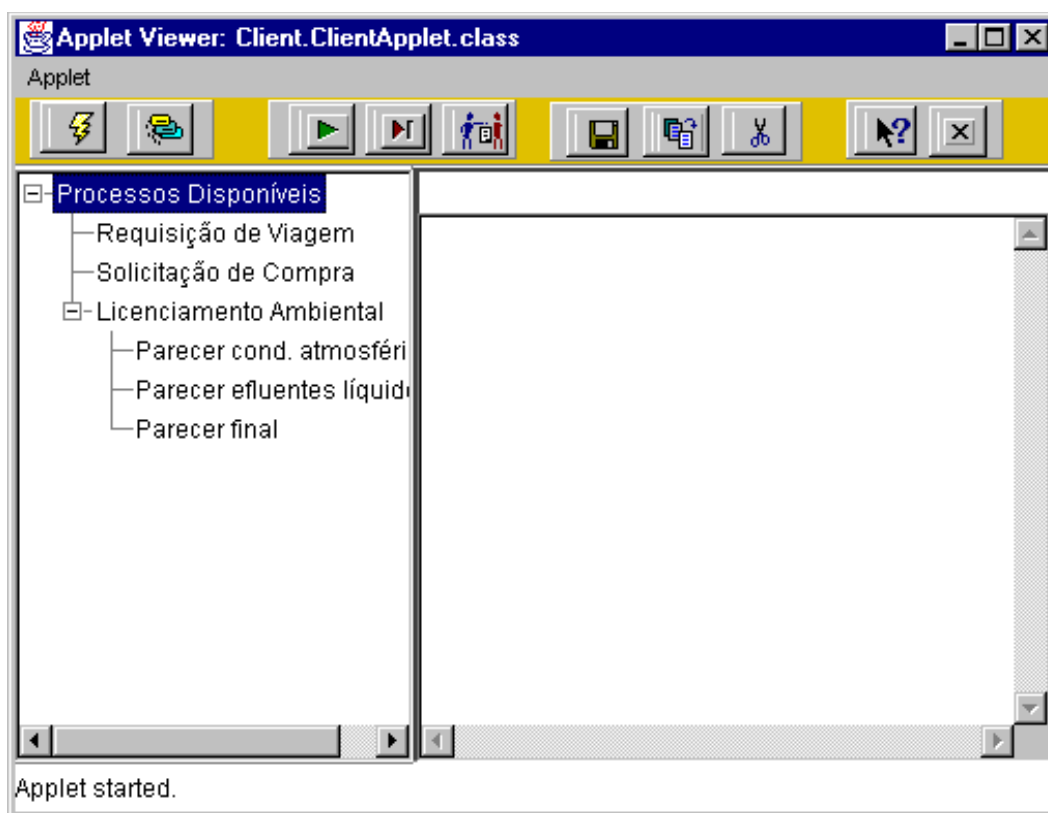





FIGURA 7.19 - Tela principal com a lista de processos a disparar

Para dar criar uma nova instância de processo, basta o participante selecionar um dos elementos da lista e pressionar o botão . Nesse momento, uma caixa de diálogo será apresentada ao usuário, solicitando um nome para a instância a ser criada. Na seqüência, o método *StartProcess* da classe *Process Definition* é invocado, recebendo como parâmetro o nome da instância a ser criada.

A lista de trabalho do participante pode ser visualizada pressionando o botão . Esse botão preenche a área de visualização de processos e atividades com a lista de trabalho do participante, como ilustrado na figura 7.20. Para tal, é invocado o método *GetWorkList* da classe *Server*. É importante observar que não apenas o título do item de trabalho é apresentado, mas também os nomes de todos os documentos com os quais o item de trabalho se relaciona. Esse recurso é possibilitado pela integração realizada através da WPDL-DOC, permitindo associar a cada atividade, através de variáveis do processo, os documentos com os quais se relaciona.

Após a obtenção da lista de trabalho, o participante pode percorrer a árvore de itens de trabalho gerada. No momento que decidir executar um certo item de trabalho, deve pressionar o botão . Esse botão invoca o método *Start* da classe *Work Item*, e permite que os documentos ligados ao item de trabalho sejam consultados e, caso exista permissão para tal, modificados, removidos e tenham novas versões criadas.

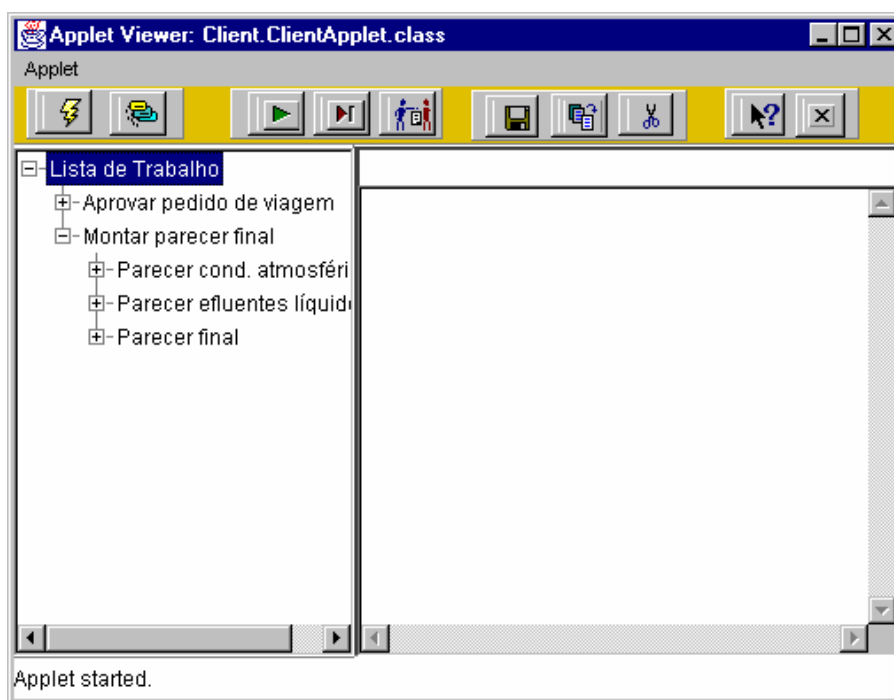


FIGURA 7.20 - Lista de trabalho do participante

É importante comentar como opera exatamente o acesso aos documentos utilizados por um certo item de trabalho. Como ilustra a figura 7.20, o aplicativo gera uma lista com todos os documentos ligados ao item de trabalho. Se o item de trabalho já

tiver sido disparado, poderão ser visualizados os atributos de cada documento, assim como o conteúdo de cada versão, com seus respectivos OIIDs. A figura 7.21 ilustra a abertura de um documento 'Parecer de Efluentes Líquidos', apresentando o conteúdo de sua versão 'Versão corrigida' na área de visualização de documentos do aplicativo.

Uma questão importante é definir como ocorrerá a criação da primeira versão de um documento, situação na qual não haverá conteúdo prévio a ser exibido. Na implementação desenvolvida, o aplicativo cliente não possui meios de descobrir, *a priori*, se um determinado documento relacionado ao item de trabalho selecionado já foi criado. Assim, ele deve invocar o método *GetVersionList* da classe *Document*, buscando recuperar a lista de versões para aquele documento. O aplicativo cliente invoca esse método quando é efetuado um duplo clique de mouse sobre o nome do documento. Caso não exista nenhuma versão no DMS, o método *GetVersionList* retornará o valor '0' (zero). Nesse caso, o aplicativo apresentará ao usuário uma caixa de diálogo indagando-o se deseja efetivamente criar o documento. É, então, invocado o método *CreateDocument* da classe *Document*, definido na seção 7.2.2.7. Através desse método, o documento receberá seus primeiros atributos e o conteúdo do *template* a ele associado será exibido na área de visualização de documentos.

Caso já existam outras versões do documento (ou seja, o valor retornado pelo método *GetVersionList* é diferente de zero), elas serão recuperadas e exibidas em uma lista, onde poderão, a qualquer momento, ser visualizadas através de um duplo clique de mouse sobre o seu nome na lista, ou, caso existam as permissões correspondentes, ser manipulada pelas outras operações suportadas pela camada de serviços.

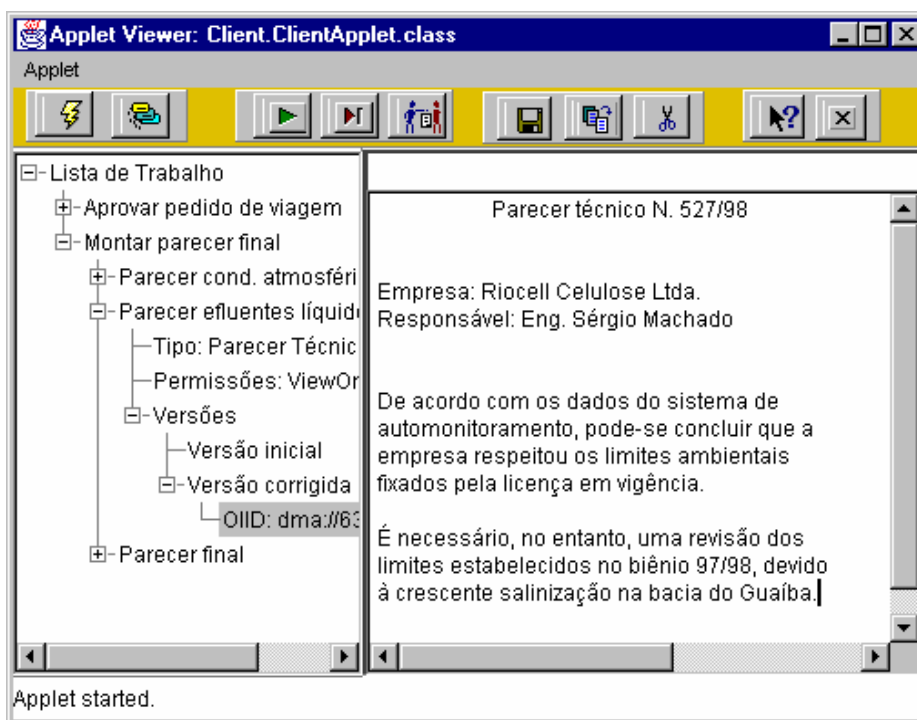


FIGURA 7.21 - Acesso aos documentos ligados a um item de trabalho

É importante observar que a habilitação dos botões da barra de ferramentas que invocam as operações de manipulação de documentos depende dos direitos definidos no *script* WPDL-DOC, através da variável do sistema de *workflow* ActivityDocs. Assim, ao ser selecionado um determinado documento, somente estarão habilitados os botões que representem operações permitidas sobre aquele documento. Além disso, é importante ressaltar que, em respeito à definição de versões estabelecida em [SOA 95], as operações de modificação só podem incidir sobre a última versão de cada documento, podendo as demais apenas ser visualizadas.

8 Conclusões

Neste capítulo são apresentadas as conclusões deste trabalho de mestrado. Inicialmente, são apresentadas as três principais contribuições do presente trabalho. Em seguida, são mostrados os possíveis cenários de aplicação onde poderiam ser utilizadas as ferramentas definidas e implementadas, na busca de ilustrar a viabilidade da proposta de integração realizada. Por final, são apresentados diversos possíveis trabalhos futuros, os quais podem vir a incrementar a sofisticação, a aplicabilidade e a facilidade de utilização da arquitetura integrada proposta nesta dissertação.

8.1 Contribuições

A contribuição central deste trabalho é a proposta de um conjunto de mecanismos, coerentemente agrupados, que permitam a integração entre sistemas de gerência de documentos e sistemas de gerência de *workflow* de forma *aberta*, isto é, utilizando os respectivos padrões de cada uma dessas indústrias. A necessidade dessa integração decorre das fortes restrições que as arquiteturas proprietárias de integração impõem a seus usuários. Entre essas restrições, pode-se citar a pouca liberdade de escolha de produtos de gerência de documentos e de gerência de *workflow*, a dificuldade de integração com sistemas já existentes e a grande dependência em relação aos fornecedores dos produtos de gerência de documentos e de gerência de *workflow*.

A concretização da proposta de integração aberta deu-se através da concepção e desenvolvimento de três mecanismos: primeiramente, um modelo para a representação integrada de documentos e processos de *workflow*; em segundo lugar, a construção de uma camada de software que integra, de forma transparente, os serviços oferecidos pelos padrões WAPI e DMA; e, por final, o desenvolvimento de um aplicativo cliente dessa camada de software. Cada um desses mecanismos encerra, em seu desenvolvimento, diversas contribuições relevantes às respectivas áreas.

Seja, inicialmente, analisado o modelo integrado de documentos e *workflow* desenvolvido. Esse modelo permite representar, de forma unificada, aspectos essenciais da interação entre os conceitos de gerência de documentos e de gerência de *workflow*. Torna-se possível, por exemplo, definir precisamente quais documentos serão manipulados pelo processo de *workflow*, além de determinar quais operações poderão ser executadas sobre esses documentos em cada uma das atividades do processo. A disponibilidade de tais recursos de modelagem permite a garantia de um nível superior de integridade e consistência dos processos de *workflow* definidos, constituindo-se, assim, em uma contribuição à modelagem conceitual desses elementos.

Além de delinear um modelo conceitual integrado, este trabalho definiu um mapeamento entre esse modelo e os padrões WAPI e DMA. Dessa forma, é possível representar a riqueza conceitual do modelo integrado em termos de elementos padronizados, os quais podem ser implementados naturalmente em sistemas de gerência de *workflow* aderentes ao padrão WAPI e em sistemas de gerência de documentos

aderentes ao padrão DMA. Um dos componentes-chave desse mapeamento é a introdução da WPDŁ-DOC, uma forma organizada de utilizar a WPDŁ para representar a integraçŁo entre conceitos de gerŁncia de documentos e de gerŁncia de *workflow*, permitindo assim expandir significativamente o poder de expressŁo de uma definiçŁo de processo de *workflow*. Ł importante observar que, apesar de acrescentar recursos poderosos Ł WPDŁ, a WPDŁ-DOC segue exatamente a mesma sintaxe da WPDŁ, podendo assim ser reconhecida por qualquer WFMS que aceite *scripts* WPDŁ. O mapeamento do modelo conceitual e a definiçŁo da WPDŁ-DOC constituem-se, com toda a certeza, em algumas das mais importantes contribuiçŁes desta dissertaçŁo.

Ainda, pode-se relacionar uma terceira contribuiçŁo relacionada ao modelo integrado: a definiçŁo de uma extensŁo Ł notaçŁo de *workflow Trigger Modeling* [JOO 94] capaz de representar aspectos de gerŁncia de documentos, como a definiçŁo das operaçŁes possŁveis sobre um documento no contexto de uma determinada atividade.

Passe-se a analisar, entŁo, o desenvolvimento da camada de serviços. Essa camada de software consegue integrar, de forma absolutamente transparente, qualquer WFMS aderente ao padrŁo WAPI a qualquer DMS aderente ao padrŁo DMA. Essa integraçŁo Ł obtida atravŁs da disponibilizaçŁo de um conjunto de mŁtodos Java que invocam serviços oferecidos pelos padrŁes WAPI e DMA, sem utilizar nenhuma API proprietŁria.

A implementaçŁo da camada de serviços permite solucionar diversos dos problemas apontados como motivaçŁo para este trabalho. Aumenta significativamente, por exemplo, a liberdade das organizaçŁes usuŁrias na escolha de produtos de gerŁncia de documentos e gerŁncia de *workflow*, pois pode-se escolher qualquer par de produtos que atenda aos padrŁes, e nŁo mais apenas aqueles integrados de forma proprietŁria. AlŁm disso, torna-se sensivelmente mais fŁcil a integraçŁo de novos produtos (sejam WFMS ou DMS) com os sistemas jŁ existentes, pois a integraçŁo Ł realizada atravŁs dos padrŁes da indŁstria. Ainda, reduz-se grandemente a dependŁncia das organizaçŁes usuŁrias em relaçŁo aos fornecedores de WFMS e DMS, pois Ł possŁvel, sem a necessidade de reescrita de cŁdigo, substituir um produto por outro de diferente fornecedor. Dessa forma, o desenvolvimento da camada de serviços constitui-se em uma das mais importantes contribuiçŁes desse trabalho.

Por final, passe-se Ł anŁlise do aplicativo cliente desenvolvido. AtravŁs desse aplicativo, Ł possŁvel utilizar os recursos oferecidos pela camada de serviços. Ł importante observar que tal acesso realiza-se atravŁs de *browsers Web*, facilitando significativamente a instalaçŁo, configuraçŁo e manutençŁo desse aplicativo.

Ł essencial ressaltar que, por abranger tanto o aspecto da modelagem de documentos e *workflow* quanto o aspecto de execuçŁo de sistemas de gerŁncia de documentos e de gerŁncia de *workflow*, a arquitetura apresentada nesse trabalho engloba as duas fases da construçŁo de sistemas dessa espŁcie. Trata-se portanto, nesse aspecto, nŁo apenas de uma arquitetura aberta, mas tambŁm uma arquitetura *completa*, oferecendo subsŁdios ao uso de padrŁes em ambas as etapas de desenvolvimento de um sistema de documentos e *workflow*.

8.2 Cenários de Aplicação

É de fundamental importância, neste momento, avaliar diversos possíveis cenários onde as contribuições oferecidas por esta dissertação podem ser aplicadas. Dessa forma, podem ser estimados o impacto e a relevância dos estudos ora conduzidos. Tendo por base o próprio processo de desenvolvimento deste trabalho, assim como a experiência pessoal do autor nas áreas de gerência de documentos e de gerência de *workflow*, é possível vislumbrar três cenários de utilização: primeiramente, como colaboração à tentativa de integração anunciada pelas entidades WfMC e AIIM [AII 98a]; em segundo lugar, como um referencial para a integração conceitual entre documentos e *workflow*; e, por final, como um produto disponível para facilitar a integração de sistemas de gerência de documentos e sistemas de gerência de *workflow*. Cada um desses cenários é discutido nos parágrafos a seguir.

O primeiro cenário frisa a capacidade deste trabalho de mestrado em contribuir para a evolução dos padrões WAPI e DMA rumo a uma integração mais consistente. Atualmente, tais padrões são absolutamente ortogonais e, apesar da crescente atenção que a integração entre eles vem recebendo de seus respectivos órgãos padronizadores, esta discussão ainda encontra-se em estágio pouco avançado. Uma das formas mais importantes de utilização desta dissertação seria, portanto, contribuir de forma inovadora nesse debate. Particularmente, durante a escrita desta dissertação, foi mantido estreito contato com membros-chave das entidades WfMC e AIIM, tendo sido firmado, inclusive, um compromisso para o repasse dos resultados deste trabalho aos membros dessas entidades responsáveis pelas propostas de integração. Dessa forma, é possível que idéias apresentadas nesse trabalho sejam aproveitadas em uma eventual padronização conjunta entre a WfMC e a AIIM.

As contribuições realizadas por esse trabalho poderiam ser aproveitadas de diversas formas nessa hipotética padronização conjunta. Os recursos introduzidos pela WPD-L-DOC poderiam, por exemplo, ser suportados diretamente por meio de novas cláusulas específicas da WPD-L. Assim, seria reduzida a complexidade de geração do *script* e a dificuldade de recuperação dos atributos, em tempo de execução. Além disso, poderiam ser introduzidos novos métodos nos padrões WAPI e DMA, similares aos métodos oferecidos pela camada de serviços, capazes de tratar de forma transparente a associação entre processos de *workflow* e documentos.

O segundo cenário ressalta a importância do modelo conceitual integrado definido neste trabalho. De fato, os conceitos relacionados nesse modelo são gerais para praticamente qualquer integração entre DMS e WFMS, não sendo, de forma alguma, específicos dos padrões DMA e WAPI. Uma vez que o suporte aos padrões DMA e WAPI em produtos comerciais é atualmente ainda bastante limitado, existe ainda uma forte necessidade de suporte conceitual para a construção de integrações proprietárias entre DMS e WFMS. Assim, a compreensão desse modelo pode colaborar de forma incisiva para a construção dessas integrações proprietárias entre DMS e WFMS, ao oferecer um referencial conceitual integrado e rico.

Por final, o terceiro cenário de aplicação vislumbrado envolve a utilização do software desenvolvido como um produto para a integração entre DMS e WFMS. Por

exemplo, uma empresa especializada em integração de produtos de gerência de documentos e produtos de gerência de *workflow* poderia utilizar a camada de serviços e o aplicativo cliente desenvolvidos para interligar um DMS qualquer, aderente ao padrão DMA, a outro WFMS qualquer, aderente ao padrão WAPI. Certamente essa empresa auferiria ganhos expressivos em termos de produtividade, pois a integração entre dois produtos exigiria, na pior das hipóteses, a construção de *drivers* DMA ou WAPI. Por sua vez, a utilização de uma abordagem de integração proprietária exigiria a integração direta das API proprietárias dos produtos, muito mais custosa e dificilmente reutilizável.

Um outro exemplo para esse cenário seria utilizar o software desenvolvido como um módulo básico, que fornece serviços integrados de gerência de documentos e *workflow*. Assim, sistemas que focam um domínio de problema específico, como o Transcoop [IOC 96], poderiam utilizar o software, evitando assim a necessidade de implementar módulos de gerência de documentos, de gerência de *workflow* ou integrações entre esses módulos.

Ainda, certos tipos de empresa poderiam beneficiar-se em utilizar o software desenvolvido nesse trabalho. Por exemplo, um fornecedor de DMS poderia acoplar a camada de serviços ao seu próprio produto e assim torná-lo, automaticamente, integrável com qualquer WFMS que suporte o padrão WAPI. Analogamente, um fornecedor de WFMS poderia acoplar a camada de serviços ao seu próprio produto e assim torná-lo, automaticamente, integrável com qualquer DMS aderente ao padrão DMA. Dessa forma, pode-se aumentar a competitividade de um determinado produto, tornando sua integração com demais sistemas menos complexa e custosa.

8.3 Trabalhos Futuros

Para finalizar esta dissertação, faz-se necessário relacionar diversos trabalhos adicionais que podem ser realizados a fim de aprimorar as contribuições aqui apresentadas. De modo geral, os trabalhos futuros podem ser classificados em três grupos: trabalhos referentes à modelagem integrada, trabalhos referentes à camada de serviços, e trabalhos referentes ao aplicativo cliente.

Em relação à modelagem integrada, um trabalho de grande valia seria oferecer suporte computacional, através de um conjunto de ferramentas de software. Seria interessante a construção de duas ferramentas: primeiramente, um editor gráfico de processos e documentos, suportando a notação integrada apresentada na seção 6.3.1; em segundo lugar, um programa capaz de mapear automaticamente o modelo integrado definido para os padrões WAPI e DMA, conforme as regras definidas na seção 6.2 e, em particular, gerando o *script* WPDL-DOC correspondente.

Ainda em termos de modelagem, um interessante trabalho futuro seria oferecer a possibilidade de especificar-se *documentos estruturados*. A utilização de documentos estruturados facilita o reuso de componentes de um documento, além de permitir um controle mais preciso de seu conteúdo [GRA 97]. A adoção de documentos estruturados possibilitaria, por exemplo, a especificação de direitos de modificação distintos para cada parte do documento.

A incorporação de documentos estruturados ao modelo integrado desenvolvido poderia ocorrer de duas maneiras distintas. A primeira seria a possibilidade de o padrão DMA, futuramente, ser acrescido de objetos referentes a documentos estruturados. Nesse caso, o modelo integrado seria incrementado com objetos equivalentes aos novos, o que causaria um impacto relativamente pequeno no mecanismo de mapeamento atualmente existente. Uma segunda possibilidade seria incorporar ao modelo integrado um determinado modelo de documentos estruturados, como o ODA [BRO 89] ou o modelo GDOC [GRA 97]. Nesse caso, seria provavelmente necessário redefinir o mapeamento do modelo integrado para os padrões WAPI e DMA, em função de possíveis não-equivalências entre os objetos dos dois modelos.

Em relação à camada de serviços, um importante trabalho adicional seria expandir a sua funcionalidade, acrescentando a ela novos métodos. De fato, apenas uma parcela dos métodos usualmente oferecidos por sistemas de gerência de documentos e por sistemas de gerência de *workflow* está contemplada na atual implementação da camada de serviços. Entre as funções de gerência de *workflow* que poderiam ser adicionadas estão a manutenção do processo e a manutenção das configurações dos participantes. Entre as funções de gerência de documentos que poderiam ser introduzidas estão os serviços de pesquisa de documentos por atributos e a definição e manutenção de *folders* de documentos.

Outro trabalho muito interessante ligado à camada de serviços seria a definição de uma camada CORBA sobre ela. Conforme [MOW 95] e [SIE 96], CORBA se apresenta, atualmente, como o mais flexível e poderoso mecanismo de comunicação distribuída entre processos. A adoção dessa camada facilitaria a construção de aplicativos clientes, pois estes poderiam ser desenvolvidos em qualquer linguagem de programação com suporte a CORBA.

Por final, coloca-se um interessante trabalho futuro relacionado ao aplicativo cliente. A implementação do aplicativo cliente desenvolvida neste trabalho possui capacidade de visualizar apenas arquivos textos simples. O fato de o aplicativo cliente executar em um *browser Web*, no entanto, permite que tal situação seja alterada. Todos os documentos cujo formato o aplicativo cliente não compreendesse poderiam ser repassados, automaticamente, a um plug-in específico [WEB 97]. A capacidade de reconhecimento de formatos de documentos do aplicativo cliente poderia ser, então, multiplicada, tornando-o de fato um cliente altamente poderoso e versátil.

Bibliografia

- [ACT 98a] ACTIVEDOC CORPORATION. **ODMA and Applications**. Disponível por WWW em <http://www.activedoc.com/odma/odmaapp.htm> (05 set. 1998).
- [ACT 98b] ACTIVEDOC CORPORATION. **ODMA and Service Providers**. Disponível por WWW em <http://www.activedoc.com/odma/odmadms.htm> (05 set. 1998).
- [AII 97] AIIM INTERNATIONAL. **DMA 1.0 Specification**. Disponível por WWW em <http://www.aiim.org/dma/dma10/index.htm> (09 set. 1998).
- [AII 98a] AIIM INTERNATIONAL. **Introduction to DMA**. Disponível por WWW em http://www.aiim.org/dma/dma_exec_overview.html (09 set. 1998).
- [AII 98b] AIIM INTERNATIONAL. **Open Document Management API**. Disponível por WWW em <http://www.aiim.org/odma/odma.htm> (09 set. 1998).
- [AMA 97a] AMARAL, V. et al. Uma Arquitetura Aberta para a Integração de Sistemas de Gerência de Documentos e Sistemas de Gerência de Workflow. In: WORKSHOP EM MULTIMÍDIA E HIPERMÍDIA, 3., 1997, São Carlos. **Anais...** São Carlos: UFSCAR, 1997. 242p. p.131-142.
- [AMA 97b] AMARAL, V.; GRALA, A.; LIMA, J. Workflow e Gerência de Documentos. In: JORNADAS DE ATUALIZAÇÃO EM INFORMÁTICA, 16., 1997, Brasília. **Anais...** Brasília: UNB, 1997. 512p. p.401-447.
- [BAI 93] BAIR, J. H. Constrasting Workflow Models: Getting to the Roots of Three Vendors. In: GROUPWARE, 1993, San Jose, Califórnia. **Proceedings...** San Mateo: Morgan Kaufamm Publishers, 1993. 490p. p.229-237.
- [BEL 95] BELL, R. Case Study: Dow Corning. In: **New Tools for New Times: The Workflow Paradigm**. Lighthouse Point: Future Strategies, 1995. 348 p. p.197-214.

- [BER 96] BERNSTEIN, P. Middleware: A Model for Distributed System Services. **Communications of the ACM**, New York, v.39, n.2, p.86-98, Feb. 1996.
- [BRO 89] BROWN, H. Standards for Structured Documents. **The Computer Journal**, New York, v.32, n.6, p.505-515, Jun. 1989.
- [CAS 95] CASATI, F. et al. Conceptual Modeling of Workflows. In: OO-ER CONFERENCE, 15., 1995, Gold Coast, Australia. **Proceedings...** [S.l.:s.n], 1995.
- [COA 98] COAD, P; MAYFIELD, M. **Projeto de Sistemas em Java: Construindo Aplicativos e Melhores Applets**. São Paulo: Makron Books, 1998. 232p.
- [DAL 95] D'ALLEYRAND, M. **Workflow em Sistemas de Gerenciamento Eletrônico de Imagens**. São Paulo: CENADEM, 1995. 112p.
- [DEN 95] DENNING, P.; MEDINA-MORA, R. Case Study: George Mason University. In: **New Tools for New Times: The Workflow Paradigm**. Lighthouse Point: Future Strategies, 1995. 348p. p.59-74.
- [DOC 97] DOCUMENTUM INC. **Documentum First to Extend Document Control to ODMA Standards**. Disponível por WWW em http://www.documentum.com/level_3.html?category=comp§ion=press&item=applic.html (03 set. 1998).
- [FAY 98] FAY, C. **Mezzanine DMA layer**. Disponível por E-mail em vla@inf.ufrgs.br. (31 jan. 1998).
- [FER 93] FERREIRA, B. J. P. **Projeto e Implementação de um Sistema de Gerência de Documentos Segundo o Paradigma de Objetos**. Porto Alegre: CPGCC da UFRGS, 1993. Dissertação de mestrado.
- [FIL 98a] FILENET CORPORATION. **Panagon Visual Work Flo**. Disponível por WWW em <http://www.filenet.com/prods/vwtext.html> (19 jul. 1998).
- [FIL 98b] FILENET CORPORATION. **Panagon Mezzanine**. Disponível por WWW em http://www.filenet.com/prods/edm/doc_mgr.html (19 jul. 1998).
- [FOW 97] FOWLER, M. **UML Distilled: Applying the Standard Object Modeling**. Reading: Addison-Wesley, 1997. 180p.

- [FRU 96] FRUSCIONE, J. **Workflow Automatizado – Como Desenvolver Projetos Gerais e Planejamento de Suporte**. São Paulo: CENADEM, 1996. 100p.
- [GAM 94] GAMMA, E. et al. **Design Patterns: Elements of Reusable Object-Oriented Software**. Reading: Addison-Wesley, 1995. 395p.
- [GEO 95] GEORGAKOPOULOS, D.; HORNICK, M.; SHETH, A. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. **Distributed and Parallel Databases**, [S.l.], v.3, n.2, p.119-153, Apr. 1995.
- [GRA 97] GRALA A.; HEUSER, C. GDOC: A System for Storage and Authoring of Documents Through WEB Browsers. In: INTERNATIONAL CONFERENCE OF THE CHILEAN COMPUTER SCIENCE SOCIETY, 17., 1997, Valparaíso, Chile. **Proceedings...** Los Alamitos: IEEE Computer Society, 1997. p.115-124.
- [HAM 94] HAMMER, M.; CHAMPY, J. **Reengenharia: Revolucionando a Empresa**. Rio de Janeiro: Campus, 1994. 198p.
- [INC 97] INCONCERT INC. **InConcert Integrates with Documentum**. Disponível por WWW em <http://www.inconcert.com/press/pr10.htm> (09 set. 1998).
- [INP 98] INPRISE CORPORATION. **Jbuilder Product Brief**. Disponível por WWW em <http://www.inprise.com/jbuilder/jb2brief/> (10 ago. 1998).
- [IOC 96] IOCHPE, C.; MANGAN, M.; ZIELINSKI, J. **First Report on Modelling Software Development through Project and Workflow Management with Integrated Human Interactions**. Porto Alegre: CPGCC da UFRGS, 1996. 32p. (Relatório Técnico, 264).
- [JOO 94] JOOSTEN, S. Trigger Modelling for Workflow Analysis. In: CON: WORKFLOW MANAGEMENT, CHALLENGES, PARADIGMS AND PRODUCTS, 1994, Viena, Austria. **Proceedings...** Viena: Oldenbourg, 1994. p.236-247.
- [JOO 95] JOOSTEN, S. A Method for Analysing Workflows. In: EUROPEAN CSCW CONFERENCE, 5., 1995, Estocolmo, Suécia. **Proceedings...** [S.l.:s.n], 1995.

- [KEY 98] KEYFILE CORPORATION. **Keyfile 4 Product Overview**. Disponível por WWW em http://www.keyfile.com/Products/KF_PB.HTM (16 jul. 1998)
- [KHO 95] KHOSAFIAN, S.; BUCKIEWICZ, M. **Introduction to Groupware, Workflow and Workgroup Computing**. New York: John Wiley & Sons, 1995. 376p.
- [KIN 97] KINKOPH, S. **Microsoft Word 97**. Rio de Janeiro: Campus, 1997. 249p.
- [KOU 97] KOULOPOULOS, T. **The Evolution and Future of Workflow**. Disponível por WWW em <http://www.actiontech.com/resource/expert/EvolutionandFutureofWorkflow.html> (13 abr. 1998).
- [KRE 97] KREPLIN, K. **WPD L Extended Attributes Usage**. Disponível por E-mail em vla@inf.ufrgs.br (29 set. 1997).
- [LAV 97] LAVA SYSTEMS INC. **Document Enabling the Enterprise**. Disponível por WWW em <http://www.lavasys.com/president.html> (17 ago. 1998).
- [LIB 95] LIBIT, J. Case Study: American President Lines. In: **New Tools for New Times: The Workflow Paradigm**. Lighthouse Point: Future Strategies, 1995. 348p. p.99-104.
- [MAR 95] MARSHAK, R. Rethinking Workflow. **Workgroup Computing Report**, New York, v.18, n.3, p.8-12, Mar. 1995.
- [MAR 97] MARSHAK, R. **DMS and WFMS integration**. Disponível por E-mail em vla@inf.ufrgs.br (27 jan. 1997).
- [MED 92] MEDINA-MORA, R.; WONG, H.; FLORES, P. **The ActionWorkflow Approach to Workflow Management**. In: CSCW, 1992, New York. **Proceedings...** [S.l.:s.n], 1992.
- [MOW 95] MOWBRAY, T.; ZAHAVI, R. **The Essential CORBA: Systems Integration using Distributed Objects**. New York: John Wiley & Sons, 1995. 316p.
- [NOR 98] NORONHA, M. **Uma proposta de suporte a versões em documentos estruturados**. Porto Alegre: CPGCC da UFRGS, 1998. Dissertação de mestrado.

- [ORF 96] ORFALL, R; HARKEY, D; EDWARDS, J. **The Essential Distributed Objects Survival Guide**. New York: John Wiley & Sons, 1996. 604p.
- [PCD 98a] PC DOCS INC. **DOCS Open**. Disponível por WWW em http://www.pcdocs.com/products/docs_open.htm (09 set. 1998).
- [PCD 98b] PC DOCS INC. **DOCS Enterprise Suite**. Disponível por WWW em http://www.pcdocs.com/products/e_suite.htm (09 set. 1998).
- [RAM 94] RAMAGE, M. **Engineering a smooth flow?** A study of workflow software and its connections with business process reengineering. Brighton: School of Cognitive & Computing Sciences, University of Sussex, 1994. Dissertação de mestrado.
- [REA 97a] REACH CORPORATION. **WorkMAN and DocBase**. Disponível por WWW em <http://www.reachsoft.com/docbase.htm> (17 abr. 1998).
- [REA 97b] REACH CORPORATION. **WorkMAN and PCDocs**. Disponível por WWW em <http://www.reachsoft.com/PcDocs.htm> (17 abr. 1998).
- [REN 94] RENAUD, P. **Introdução aos Sistemas Cliente/Servidor**. Rio de Janeiro: Infobook, 1994. 336p.
- [ROS 95] ROSE, L. Case Study: Logicon Software Engineering. In: **New Tools for New Times: The Workflow Paradigm**. Lighthouse Point: Future Strategies, 1995. 348p. p.139-148.
- [SAD 97] SADIQ, W.; ORLOWSKA, M. Applying a Generic Conceptual Workflow Modeling Technique to Document Workflows. In: AUSTRALIAN DOCUMENT COMPUTING SYMPOSIUM, 2., 1997, Melbourne, Austrália. **Proceedings...** [S.l.:s.n], 1997. p.36-45.
- [SIE 96] SIEGEL, P. **CORBA: Fundamentals and Programming**. New York: John Wiley & Sons, 1996. 694p.
- [SIL 95] SILVER, Bruce. Automating the Business Environment. In: **New Tools for New Times: The Workflow Paradigm**. Lighthouse Point: Future Strategies, 1995. 348p. p.173-196.
- [SOA 95] SOARES, L. F. G.; RODRIGUEZ, N. L. R.; CASANOVA, M. A. Nested composite nodes and version control in an open hypermedia system. **Information Systems, Special issue on Multimedia Information Systems**, New York, v.20, n.6, p.501-591, 1995.

- [STA 98] STAFFWARE PLC. **Staffware97 product overview**. Disponível por WWW em <http://www.staffware.com/home/products/97overview.htm> (10 set. 1998).
- [SWE 95] SWENSON, K. Workflow Management Standards and Interoperability. In: **New Tools for New Times: The Workflow Paradigm**. Lighthouse Point: Future Strategies, 1995. 348 p. p.25-36.
- [TAK 95] TAKAHASHI, K.; HIGUCHI, M. **Hypermedia Support for Workflow Management in Collaborative Document Production**. Disponível por WWW em http://www.nttlabs.com/people/kt/WF_pp.html (01 set. 1998).
- [THE 95] THE, L. Workflow Tackles the Productivity Paradox. **Datamation**, New York, v.40, n.3, Mar. 1995.
- [WEB 97] WEBER, J. **Special Edition Using Java 1.1**. Indianapolis: Que, 1997. 1234 p.
- [WMC 94] WORKFLOW MANAGEMENT COALITION. **The Workflow Reference Model**. Disponível por WWW em <http://www.aiim.org/wfmc/standards/docs/rmv1-16.pdf> (10 set. 1998).
- [WMC 96a] WORKFLOW MANAGEMENT COALITION. **Terminology & Glossary**. Disponível por WWW em <http://www.aiim.org/wfmc/standards/docs/glossary.pdf> (10 set. 1998).
- [WMC 96b] WORKFLOW MANAGEMENT COALITION. **Interface 4: Interoperability Abstract Specification**. Disponível por WWW em <http://www.aiim.org/wfmc/standards/docs/if4-a.pdf> (10 set. 1998).
- [WMC 96c] WORKFLOW MANAGEMENT COALITION. **Interface 4: Interoperability Internet e-mail MIME Binding**. Disponível por WWW em <http://www.aiim.org/wfmc/standards/docs/if4-m.pdf> (10 set. 1998).
- [WMC 96d] WORKFLOW MANAGEMENT COALITION. **Interface 5: Audit Data Specification**. Disponível por WWW em <http://www.aiim.org/wfmc/standards/docs/if59611.pdf> (10 set. 1998).

- [WMC 97] WORKFLOW MANAGEMENT COALITION. **Interface 2: Workflow Application Programming Interface (WAPI) Specification.** Disponível por WWW em <http://www.aiim.org/wfmc/standards/docs/if2v20f.pdf> (10 set. 1998).
- [WMC 98a] WORKFLOW MANAGEMENT COALITION. **Introduction to the Workflow Management Coalition.** Disponível por WWW em <http://www.aiim.org/wfmc/about-fr.htm> (10 set. 1998).
- [WMC 98b] WORKFLOW MANAGEMENT COALITION. **Interface 1: Process Definition Interchange v. 1.0 Beta.** Disponível por WWW em <http://www.aiim.org/wfmc/standards/docs/if19807r11.pdf> (10 set. 1998).
- [WOD96] WODTKE, D. et al. The Mentor Project: Steps Towards Enterprise-Wide Workflow Management. In: IEEE INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 1996, New Orleans. **Proceedings...** [S.l:s.n], 1996.



CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

"Uma Arquitetura Aberta para a Integração de Sistemas de Gerência de Documentos e
Sistemas de Gerência de Workflow"

por

Vinícius Leopoldino do Amaral

Dissertação apresentada aos Senhores:

Prof. Dr. Marcos Roberto da Silva Borges (NCE/UFRJ)

Prof. Dr. José Palazzo Moreira de Oliveira

Prof. Dr. Cirano Iochpê

Vista e permitida a impressão.

Porto Alegre, 21/06/99.

Prof. Dr. José Valdeni de Lima,
Orientador.

Dra. Sônia Maria Das Neves Torres
Coordenadora do Programa de Pós-Graduação
em Computação - PPGC
Instituto de Informática - UFRGS