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

Independência entre Aplicações e Processos Modelados

por

RODRIGO PASCHOAL DO VALLE

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

Prof. José Palazzo Moreira de Oliveira
Orientador

Porto Alegre, janeiro de 2002.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Valle, Rodrigo Paschoal do

Independência entre Aplicações e Processos Modelados / por Rodrigo Paschoal do Valle. – Porto Alegre: PPGC da UFRGS, 2002.

114 f. : il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR – RS, 2002. Orientador: Oliveira, José Palazzo Moreira de.

1. Workflow. 2. Desenvolvimento de Aplicações. 3. Arquitetura Independente. I. Oliveira, José Palazzo Moreira de. II Título.

UNIVERIDADE FEDERAL DO RIO GRANDE DO SUL
Reitora: Profª. Wrana Panizzi
Pró-Reitor de Ensino: Prof. José Carlos Ferraz Hennemanh
Pró-Reitor Adjunto de Pós-Graduação: Prof. Jaime Evaldo Fensterseifer
Diretor do Instituto de Informática: Prof. Philippe Olivier Alexandre Navaux
Coordenador do PPGC: Prof. Carlos Alberto Heuser
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

Agradecimentos

Gostaria de agradecer, antes de tudo, pela paciência, compreensão e apoio de meus pais, não só por mais esta etapa que acaba, mas todas as que já se passaram e pelas que ainda estão por vir. Pois, sem o seu apoio, muitas das etapas já ultrapassadas não teriam sido superadas ou mesmo almejadas.

Um agradecimento especial ao meu orientador José Palazzo Moreira de Oliveira, o qual me auxiliou e direcionou no desenvolvimento deste estudo. Pelo incentivo e pelas críticas, sempre apresentadas de forma construtiva.

Agradeço, também, a todas os colegas e amigos que colaboraram direta ou indiretamente como participantes na elaboração deste trabalho; tanto como ouvintes das dúvidas e dos problemas enfrentados, quanto como participantes com análises e opiniões sobre o seu conteúdo e estrutura. Um outro agradecimento especial é dedicado a meu colega Sandro Coutinho Emerim, o qual participou ativamente em várias etapas, inclusive com a contribuição de seu Trabalho de Diplomação, o qual foi baseado na proposta exposta neste estudo.

Sumário

Lista de Abreviaturas 6

Lista de Figuras 7

Lista de Tabelas 9

Resumo 10

Abstract 11

1 Introdução 12

1.1 Histórico e Contextualização da Tecnologia de "Workflow" 13
1.2 Padrões e Conceitos sobre "Workflow" 13
1.3 Arquitetura proposta pela WfMC 15
1.4 Estudos e Trabalhos que Motivaram o Desenvolvimento deste Estudo 21

2 Arquiteturas de "Workflow" 22

2.1 Arquitetura Baseada em um Único Ambiente 22
2.2 Arquitetura Baseada em Ambientes Distintos 25

3 O Enfoque das Aplicações para cada uma das Arquiteturas 30

3.1 Aplicações para Arquiteturas Baseadas em um Único Ambiente 30
3.2 Aplicações para Arquiteturas Baseadas em Modelagem e Execução Distintas 31

4 Dependência entre as Aplicações e o Fluxo Modelado 33

4.1 Dependência para as Aplicações com Ambiente Único 34
4.2 Dependência para as Aplicações com Ambientes Distintos 36

5 Problemas gerados pela Dependência entre Processos e Aplicações 40

5.1 Problemas existentes durante o desenvolvimento das aplicações 41
5.2 Problemas existentes durante a manutenção dos fluxos 42

6 Proposta de Solução para o Problema da Dependência 45

6.1 Introdução 45
6.2 Objetivos 45
6.3 Características 46
6.4 Estrutura 47
6.5 Banco de Dados da Camada intermediária 51
6.5.1 Banco de Dados da Camada de Independência 52
6.5.2 Banco de Dados da Camada de Integração 64
6.6 Camada de Independência 66
6.6.1 API's da Camada de Independência 66
6.7 Camada de Integração 71
6.7.1 API's da Camada de Integração 73

6.7.2 Motor de “Workflow” - W4 (*World Wide Web Workflow*) 74
6.8 Arquitetura Completa Detalhada 78

7 Enfoque das aplicações para este novo modelo proposto 81

8 Conclusões 92

9 Trabalhos Futuros 93

9.1 Suporte a Modelagem dos Fluxos (CASE) 93
9.1.1 Estudo dos padrões utilizados para modelagem de fluxos 93
9.1.2 Definição de uma metodologia de modelagem 94
9.1.3 Adequação aos conceitos utilizados pela Camada de Independência 98
9.1.4 Implementação da interface de utilização 98
9.2 Padronização visando integração entre motores 98

Anexo API’s da Camada de Independência 100

Bibliografia 107

Lista de Abreviaturas

ACL	*Access Control List*
API	*Application Programming Interface*
B2B	*Business to Business*
B2C	*Business to Commerce*
CASE	*Computer Assited Software Engineering*
CGI	*Commom Gateway Interface*
CORBA	*Common Object Request Broker Architecture*
ER	Entidade-Relacionamento
GED	Gerenciamento Eletrônico de Documentos
GUI	*Graphics Users Interface*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transport Protocol*
HTTPS	*Hypertext Transport Protocol Security*
IIS	*Internet Information Server*
IMAP	*Internet Message Access Protocol*
JDBC	Java *Database Conectivity*
LDAP	*Lightweight Directory Access Protocol*
MAPI	*Messaging Application Programming Language*
MIME	*Multipurpose Internet Mail Extensions*
NNTP	*Network News Transfer Protocol*
OSI-TP	*Open System Interconnection Transaction Processing*
PHP	*Hypertext Preprocessor*
POP	*Post Office Protocol*
RPC	*Remote Procedure Call*
SGBD	Sistema de Gerenciamento de Banco de Dados
SMTP	*Simple Mail Transport Protocol*
SQL	*Structure Query Language*
SSL	*Secure Sockets Layer*
TCP/IP	TCP = *Transmission Control Protocol*, IP = *Internet Protocol*
UML	*Unified Modeling Language*
VB	Visual Basic
WAPI	*Workflow* API *and Interchange Formats*
WFDL	*Workflow Delinition Language*
WFM	*Workflow Management*
WfMC	*Workflow Management Coalition*
WFMS	*Workflow Management Syetem*
WWW	*World Wide Web*
XML	*Extensible Markup Language*
XSL	*Extensible Stylesheet Language*

Lista de Figuras

FIGURA 1 - Modelo de referência sobre arquitetura de *workflow* [WFM 96]. 15
FIGURA 2 - Modelo aprofundado da arquitetura proposta em [WFM 96]. 17
FIGURA 3 - Modelo da Arquitetura Baseada em um Único Ambiente 23
FIGURA 4 - Arquitetura de um motor de *workflow* 26
FIGURA 5 - Escopo da aplicação para arquitetura com ambiente único. 30
FIGURA 6 - Escopo da aplicação para arquitetura com ambientes distintos. 31
FIGURA 7 - Diagrama da Arquitetura Baseada em um Único Ambiente 34
FIGURA 8 - Diagrama da Arquitetura Baseada em Diferentes Ambientes 37
FIGURA 9 - Exemplo de uma reutilização de processos 44
FIGURA 10 - Diagrama da arquitetura baseada na proposta para independência 46
FIGURA 11 - Camada Intermediária com o Motor de *Workflow* 47
FIGURA 12 - Arquitetura da Camada Intermediária 49
FIGURA 13 - Camada Intermediária e Camada de Integração 50
FIGURA 14 - Representação dos bancos de dados 51
FIGURA 15 - Tabela de Ação da Camada de Independência 53
FIGURA 16 - Tabela de Acesso da Camada de Independência 54
FIGURA 17 - Tabela de Campo da Camada de Independência 54
FIGURA 18 - Tabela de Estado da Camada de Independência 55
FIGURA 19 - Tabela de Fluxo da Camada de Independência 56
FIGURA 20 - Tabela de Grupo da Camada de Independência 57
FIGURA 21 - Tabela de Papel da Camada de Independência 58
FIGURA 22 - Tabela de Tipo da Camada de Independência 58
FIGURA 23 - Tabela de Usuário da Camada de Independência 59
FIGURA 24 - Relacionamentos do Acesso para os estados 60
FIGURA 25 - Relacionamentos do Acesso para o Fluxo 61
FIGURA 26 - Relacionamentos para as Ações e Usuários 61
FIGURA 27 - Relacionamentos para os Campos 62
FIGURA 28 - Relacionamentos em entre os Estados, Ações e Fluxo. 62
FIGURA 29 - Relacionamentos entre Grupo e demais tabelas 63
FIGURA 30 - Relacionamentos para as Regras ou Papéis 63
FIGURA 31 - Tabela de Fluxo_Procedure da Camada de Integração 64
FIGURA 32 - Tabela de Campo_Variable da Camada de Integração 65
FIGURA 33 - Tabela de Acao_Activity da Camada de Integração 65
FIGURA 34 - API's da Camada de Independência 67
FIGURA 35 - Diagrama das classes da Camada de Independência 71
FIGURA 36 - Representação do mapeamento da API's da Camada de Integração 73
FIGURA 37 - Exemplo de estrutura utilizada para modelagem no W4 75
FIGURA 38 - Start Node 76
FIGURA 39 - Stop Node 76
FIGURA 40 - Basic Activity 76
FIGURA 41 - Condition 77
FIGURA 42 - Automated Activity 77
FIGURA 43 - Join 77
FIGURA 44 - Conditional Transition 77
FIGURA 45 - Transition 77
FIGURA 46 - Arquitetura completa do motor de *workflow* 78

FIGURA 47 - Execução das aplicações ao utilizar a Camada Intermediária 82
FIGURA 48 - Código das aplicações utilizando a Camada Intermediária 86
FIGURA 49 - Tela inicial do protótipo 87
FIGURA 50 - Tela principal para o fluxo de solicitação de promoção 88
FIGURA 51 - Tela principal para o fluxo de viagens 88
FIGURA 52 - Visualização dos documentos no estado de “Em Elaboração” 89
FIGURA 53 - Visualização de um do documento específico 90
FIGURA 54 - Exemplo de modelagem de fluxo 96

Lista de Tabelas

TABELA 1 - Exemplo de um *flag* único para o fluxo 41
TABELA 2 - Exemplo de um conjunto de *flags* para o fluxo 42
TABELA 3 - Representação dos símbolos da modelagem proposta 97

Resumo

O presente trabalho expõe uma análise sobre duas arquiteturas disponíveis para o desenvolvimento de sistemas que utilizam a tecnologia de *workflow*. Estas arquiteturas são: a Arquitetura Baseada em Modelagem e Execução em um Único Ambiente e a Arquitetura Baseada em Modelagem e Execução em Ambientes Distintos. São expostas: características, vantagens e desvantagens destas arquiteturas. A Arquitetura Baseada em Modelagem e Execução em Ambientes Distintos é analisada em seu principal ponto negativo: a falta de independência existente entre as aplicações e os processos modelados, bem como são discutidos os problemas que esta falta de independência gera no desenvolvimento e manutenção das aplicações.

Uma alternativa à dependência entre o motor de *workflow* (*engine*) e as aplicações é proposta para a Arquitetura Baseada na Modelagem e Execução em Ambientes Distintos. Esta proposta é baseada em um modelo que consiste de uma Camada Intermediária, armazenada em um banco de dados relacional, capaz de extender as funcionalidades do motor de *workflow*. Este banco de dados armazena toda a estrutura dos processos modelados, sendo responsável pela função que atualmente é repassada às aplicações: o controle da lógica dos processos.

Este trabalho produziu de uma Camada Intermediária, dividida em Camada de Independência (suportando a independência) e Camada de Integração (responsável pela comunicação com o motor de *workflow*). O estudo apresentada as estruturas do banco de dados, as funções disponibilizadas pela API da Camada Intermediária e um pequeno protótipo para demonstrar a arquitetura proposta.

Palavras-chave: *Workflow*, desenvolvimento de aplicações, arquitetura independente.

TITLE: "INDEPENDENCE BETWEEN APPLICATIONS AND MODELLED PROCESSES"

Abstract

This work presents an analyse of two available arquitectures for the development of systems that use the workflow technology. They are: The Arquitecture Based on Modelling and Execution in a Single Environment and the Arquitecture Based on Modelling and Execution in a Separated Environment. Features, advantages and disadvantages for these arquictetures are explained. The one Based on Modelling and Execution in a Separated Environment is analysed in its main negative point: the lack of independence that exists between the applications and the modelled processes, as well as the consequence this problem generates in developing and maintaining workflow applications.

An alternative for the dependence between the workflow engine and the applications is proposed for the Arquitecture Based on Modelling and Execution in a Separated Environment. This proposal is based on a model that consists of an Intermediary Layer storaged in a relational database, capable of extending the features of the workflow engine. This database stores all structures of the modelled processes, being responsable for a function that currently is performed by the applications: the control over the processes's logic.

This work produced an Intermediary Layer, divided into Independency Layer (suporting the independence) and Integration Layer (responsable for the communication with the workflow engine). The work presents the structures of the database, the available functions for the API of the Intermediary Layer and a small prototype to demonstrate the proposed arquitecture.

Keywords: Workflow, application development, independent arquitecture.

1 Introdução

Este estudo realiza uma análise aprofundada sobre a arquitetura de *workflow* utilizada atualmente, apontando suas características, vantagens e desvantagens. É proposto, para sua maior limitação, a dependência entre aplicações e modelagem, um enfoque diferenciado no desenvolvimento das aplicações, de modo a contornar ou, ao menos, minimizar este seu ponto negativo.

Será demonstrado o problema gerado, para as aplicações, em decorrência desta dependência. Uma das principais razões desta dependência está no enfoque adotado pelos fornecedores de ambientes em *workflow*; pois o enfoque principal dos fabricantes deste ambientes está centralizado na modelagem dos processos. Não existe, portanto, um esforço visando especificações e padronizações quanto à forma como as aplicações irão realizar a sua interação com o ambiente, conseqüentemente, com os processos modelados.

A proposta, para esta limitação dos ambientes atuais de *workflow*, está calcada em uma Camada Intermediária, que vem a ser uma etapa complementar para os ambientes de *workflow*. Foi desenvolvido um protótipo que implementa esta Camada Intermediária, assim como uma pequena aplicação, de modo a permitir uma melhor visualização e compreensão desta proposta de melhoria.

O trabalho está estruturado da seguinte forma:

- Na Introdução, compondo o capítulo 1, temos a contextualização dos sistemas de *workflow*: suas características, padrões, conceitos, arquitetura proposta e motivações para a realização deste trabalho;

- No capítulo 2 são abordados os detalhes sobre as arquiteturas de *workflow* atuais, realizando um comparativo com a Arquitetura Baseada em um Único Ambiente para Modelagem e Execução, e a Arquitetura Baseada em Ambientes Distintos para a Modelagem e Execução dos processos;

- No capítulo 3 temos o enfoque para as aplicações desenvolvidas sobre cada uma destas duas arquiteturas, exploradas no capítulo 2;

- No capítulo 4 é apresentada a dependência existente entre as aplicações e os processos modelados sobre o ambiente de *workflow*;

- No capítulo 5 são abordados os problemas gerados pela dependência analisada no capítulo 4, onde temos explicitado o desenvolvimento das aplicações sobre a Arquitetura Baseada em Ambientes Distintos para Modelagem e Execução;

- No capítulo 6 é apresentada a proposta de solução para o problema da falta de independência entre as aplicações e os processos. Neste capítulo é exposta a Camada Intermediária, com: sua arquitetura, suas divisões e todas as demais características de seu funcionamento;

- No capítulo 7 é explorado o enfoque que teremos, para as aplicações, ao utilizarem a Camada Intermediária, aspectos do desenvolvimento do protótipo implementado serão abordados neste capítulo;

- No capítulo 8 temos as conclusões sobre o estudo realizado;

- No capítulo 9 são apresentadas propostas para novos trabalhos provindos, a partir desta análise realizada, sobre as arquiteturas de ambientes de *workflow*.

1.1 Histórico e Contextualização da Tecnologia de "Workflow"

O avanço da tecnologia, nos últimos anos, tornou-se um grande aliado das organizações de modo a alcançar maior competitividade. Novas técnicas de administração baseada em processos e o surgimento de um ambiente computacional distribuído, heterogêneo e mais amigável permitiram que a área de sistemas de *workflow* tivesse um crescimento significativo. Os objetivos de um sistema baseado em tecnologia de *workflow*, principalmente quando estamos no contexto de uma empresa de médio ou grande porte, são basicamente de definir, padronizar e otimizar os processos existentes. Uma vez implantadas estas modificações, adiciona-se uma série de vantagens à maneira como as tarefas, até então, eram realizadas [GEO 95], [NIC 96] e [DUI 95].

Como a tecnologia de *workflow* é relativamente nova, existem ainda muitas carências pela falta de padrões ou normas para se definirem características importantes dentro desta tecnologia. Podemos citar como exemplo destas características: diferenças entre as categorias de *workflow*, dificuldade de formalizar conceitos ou mesmo conceitos diferenciados, falta de padrões para a modelagem dos processos, diferentes arquiteturas de desenvolvimento e mesmo diferentes arquiteturas de execução dos ambientes de *workflow*.

Ao iniciar os estudos sobre sistemas baseados nesta tecnologia, temos uma forte referência que se trata de uma coalizão, entre empresas e universidades, com a finalidade de definir padrões e normas para o desenvolvimento de sistemas de *workflow*, a WfMC (*Workflow Management Coalition*). Esta coalizão foi criada em 1993, é uma entidade formada por empresas e universidades, com o objetivo, dentre outros, de estabelecer padrões para os sistemas baseados em *workflow* [WFM 96].

1.2 Padrões e Conceitos sobre "Workflow"

Ao estudarmos os conceitos e padrões sugeridos pela WfMC recaímos inicialmente sobre as categorias de sistemas de *workflow*, onde temos antes mesmo de sistemas de *workflow* os sistemas chamados de *groupware*. Os sistemas desta forma denominados são, basicamente, sistemas em que seu principal objetivo é o auxílio nos processos utilizando a troca de mensagens e documentos; ou seja, são sistemas baseados fortemente em um ambiente de correio eletrônico [ELL 91] e [GRU 97]. Neste contexto de sistemas baseados em assistir aos grupos de usuários na comunicação, colaboração e coordenação de suas atividades, estão enquadrados ambientes como Lotus Notes (IBM), Exchange (Microsoft), Group-Wise (Novell), entre outros.

Podemos conceituar, as três características citadas acima sobre sistemas de *groupware*, como sendo:

- Comunicação: ajudar pessoas a compartilhar informações;
- Coordenação: ajudar pessoas a coordenar seus papéis pessoais com outras pessoas;
- Colaboração: ajudar o trabalho entre estas pessoas.

Existem outros autores que são fortemente citados quando referenciamos o trabalho com *groupware*, que são [BOR 95], [BER 97] referências aos trabalhos de Ellis, Gibbs e Rein [ELL 91], [NUM 91] e Grudin [GRU 94]. Dentre estes autores existem, algumas diferenciações entre os seus conceitos. Alguns dos autores enfocam os sistemas de *groupware* segundo tempo e espaço; outros, por sua vez, segundo proximidade ou ainda por previsibilidade.

A diferença entre os sistemas denominados *groupware* e *workflow* varia de acordo com a bibliografia estudada; porém, poderíamos dizer que uma das principais diferenças seria o enfoque que está sendo dirigido ao trabalho realizado. Os sistemas de *workflow* têm mais enfocado o conceito de tarefas ou processos que devem ser realizados pelos usuários, juntamente estão presentes etapas de modelagem dos fluxos e controle das instâncias de processos que estão em execução. Sendo assim, sistemas de *workflow* têm a característica de serem mais estruturados e complexos que os sistemas enquadrados como *groupware*.

Quanto a categorias, os sistemas de *workflow* dividem-se em 3 grandes categorias, que são: *Workflow* de Produção, *Workflow* Administrativo e *Workflow* Ad-Hoc. Essas categorias devem ser analisadas de forma delicada; pois, existentes divergências quanto à definição de cada uma delas de acordo com a literatura à qual o estudo está baseado. Porém, de acordo com [MCC 92], [BAR 95] podemos definir as categorias em:

- *Workflow* de Produção: como sendo sistemas de *workflow* envolvidos em processos de informação complexos, onde existe a necessidade de acesso a múltiplos sistemas de informação. São sistemas críticos para a empresa, que não podem interromper sua execução sob o risco de causar sérios prejuízos para a organização. Seu enfoque está relacionado com características de sistemas transacionais;

- *Workflow* Administrativo: como sendo sistemas de *workflow* baseados em processos, característico pela complexidade e detalhamentos nos seus fluxos. Possuem características semelhantes ao *workflow* de produção; porém são voltados para as atividades administrativas internas da organização. São menos exigentes do que os de produção e menos críticos também;

- *Workflow* Ad-Hoc: como sendo sistemas de *workflow* onde os fluxos não são previamente especificados, durante o trâmite dos processos serão resolvidas as pessoas envolvidas no mesmo e o fluxo que será realizado. Seu fluxo pode sofrer grandes modificações, não são críticos para o funcionamento da organização, podendo suspender sua execução por alguns momentos sem repercutir em grandes prejuízos.

1.3 Arquitetura proposta pela WfMC

Apesar dos vários trabalhos existentes nesta área, são poucos os estudos realizados sobre a arquitetura de funcionamento e a execução das aplicações baseadas neste contexto. Muitas diretrizes são sugeridas pela WfMC como padrões para arquitetura do ambiente de *workflow* e mesmo, funcionalidades mínimas para sistemas de *workflow*. De acordo com as referências citadas pela WfMC, estes sistemas deveriam seguir a seguinte arquitetura:

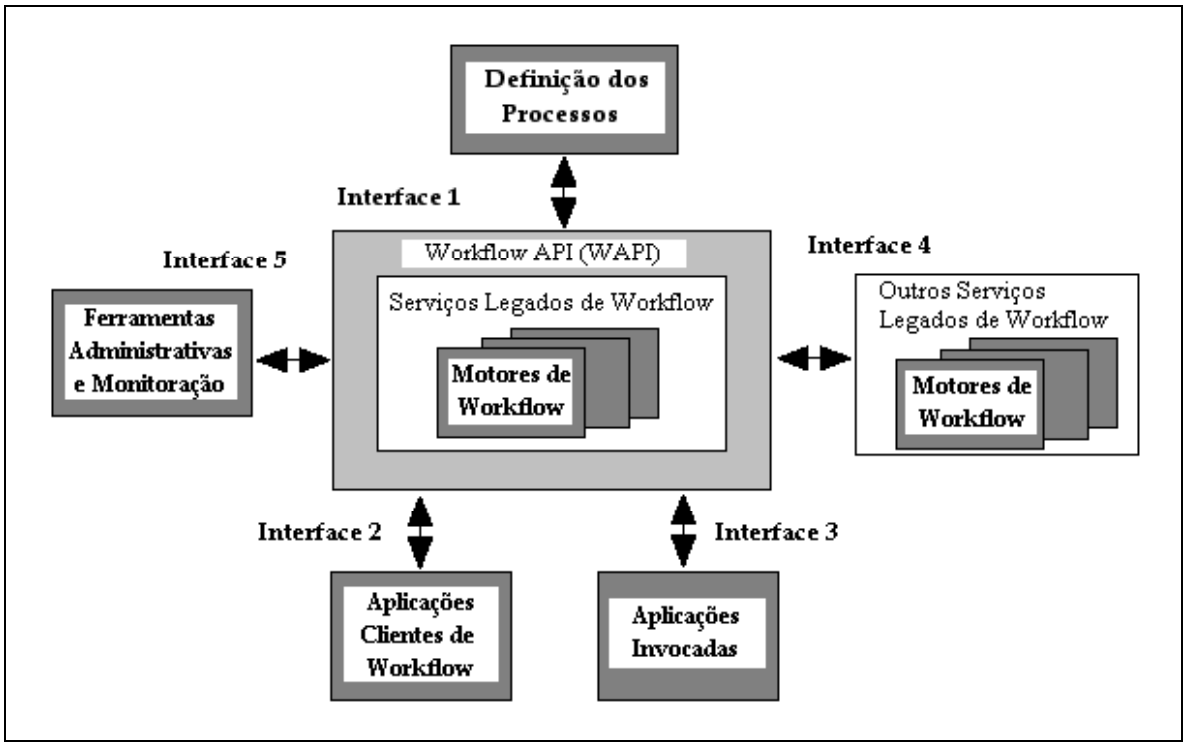


FIGURA 1 - Modelo de referência sobre arquitetura de *workflow* [WFM 96].

Temos, na Figura 1, o modelo sugerido por [WFM 96], o qual referencia a arquitetura a ser seguida pelos ambientes de *workflow*. Onde temos, para cada um dos elementos componentes do diagrama uma função específica. Juntamente neste diagrama, estão representadas as interfaces sugeridas pela WfMC, separada em cinco (interface 1 até a interface 5). Os componentes são descritos a seguir, conforme sua definição.

- **Ferramenta de Definição dos Processos:**
  Este componente tem como principal função possibilitar a modelagem dos processos, gerando uma descrição dos fluxos a serem automatizados. Neste módulo será utilizada a modelagem adotada pelo fabricante dos produtos; pois, não dispomos de um padrão "de fato" para o desenvolvimento das definições sobre os processos. Estas ferramentas são utilizadas para analisar, modelar e descrever os processos;

- **Definição dos Processos:**
  Este módulo contém toda a informação necessária sobre os processos para possibilitar sua execução pelo sistema, incluindo condições de início e fim,

atividades, tarefas de usuários, referências a aplicações externas e demais elementos componentes dos sistemas. Representa, formalmente, as rotinas que interpretam o resultado da modelagem, gerando as informações necessárias para possibilitar o controle das instâncias e eventos pelo motor de *workflow*;

- **"Workflow Enactment Service":**
  Interpreta a descrição dos processos e controla sua instanciação, disparando atividades, adicionando itens de trabalho para usuários e invocando aplicações externas (tarefa é realizada por um ou mais motores de *workflow*, que administram a execução de instâncias individuais de vários processos). Mantém os dados internos de controle de forma centralizada ou distribuída por mais de um motor (*workflow engine*);

- **"Workflow Relevant Data and Application Data":**
  São dados gerados por aplicações externas e acessados pelo *engine*, visando o controle da execução do *workflow*;

- **Listas de Tarefas:**
  Lista das tarefas de um usuário que é alimentada pelo *engine*. Este módulo é extremamente interessante, pois subsidia informações para um controle de extrema utilidade para os usuários finais do sistema – sua lista de tarefas ou pendências;

- **"Worklist Handler & User Interface":**
  Realiza a gerência sobre a interação dos usuários com o *enactment service*, sendo responsável pelo progresso do trabalho que necessita de intervenção do usuário. O *worklist handler* pode ou não estar acoplado com a interface de usuário. Também é possível o uso de um mesmo *software* cliente para acesso a vários serviços de *workflow*, unificando a *worklist* de um usuário. Responsável pela invocação de aplicações externas ligadas à gerência da *worklist*;

- **Operações de Supervisão:**
  Utilizadas por usuários especiais para a administração do sistema. No momento que se fizerem necessárias intervenções para o tratamento de exceções ou uma modificação sobre uma específica instância em execução, este módulo será ativado;

- **WAPI ("Workflow API & Interchange Formats"):**
  A camada disponibilizada pela API suportada pelo *workflow enactment service* em sua fronteira com demais componentes. A WAPI contém um núcleo comum de funções existentes nas cinco interfaces padronizadas, com extensões específicas para necessidades de cada uma. É a responsável pela comunicação entre as aplicações e o motor de *workflow*;

- **Motores de Workflow:**
  O *workflow engine*, ou simplesmente motor, é o coração da arquitetura; pois, a maior parte dos controles realizados no ambiente, ficam sob sua responsabilidade, podemos citar entre outras de suas funções, as seguintes:

- Interpretar a definição de um processo;
- Controlar as instâncias de processos;
- Gerenciar as atividades de um processo (operações seqüenciais ou paralelas, definição de prazos, interpretação de dados relevantes do *workflow*, controle de eventos entre outras funções de controle para as atividades);
- Realizar o *login* e *logout* de usuários;
- Identificar os itens de trabalho;
- Manter os dados de controle e dados relevantes e passagem desses dados para usuários e aplicações;
- Subsidiar a interface para invocação de aplicações externas;
- Coordenar as ações de supervisão para controle, administração e auditoria dos sistemas.

Temos, no diagrama da Figura 2, a exposição, a um nível mais aprofundado, da interação entre os elementos componentes da arquitetura proposta pela WfMC. Temos, neste diagrama, a totalidade dos elementos propostos, bem como a relação que é formada entre cada um dos elementos.

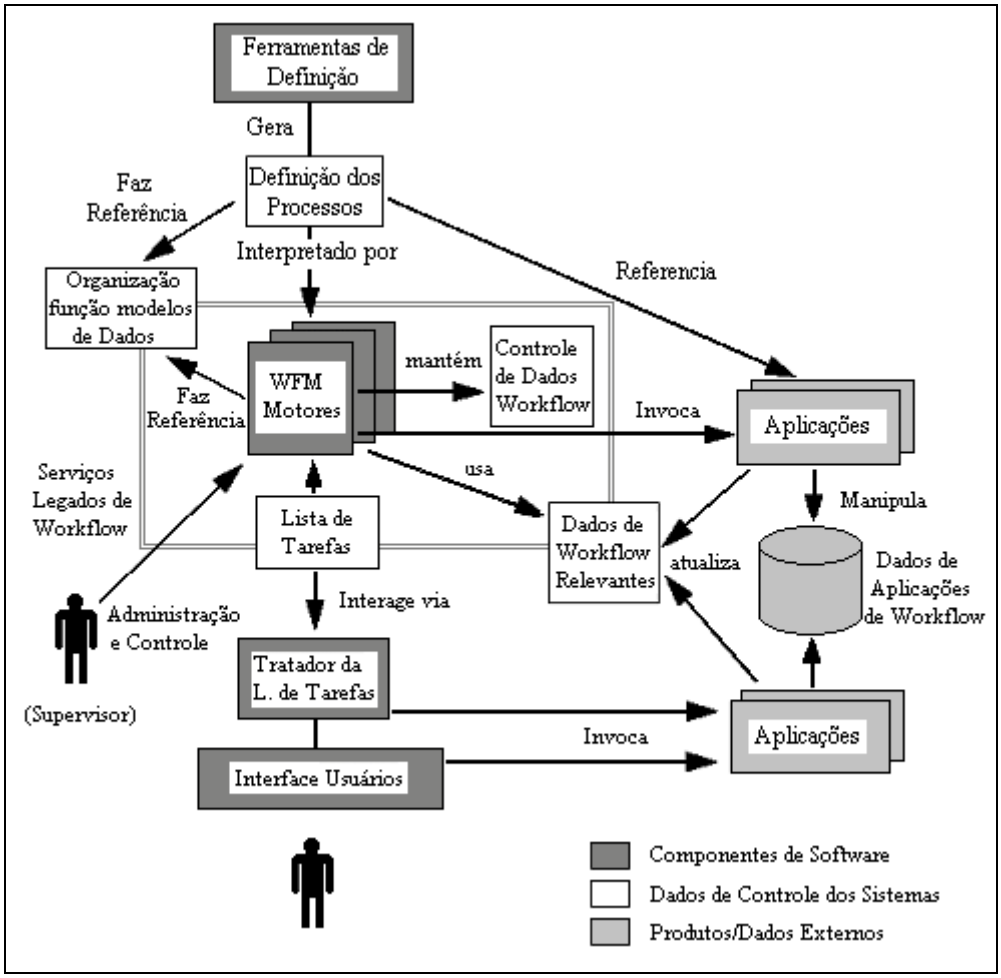


FIGURA 2 - Modelo aprofundado da arquitetura proposta em [WFM 96].

Temos ainda a definição de cinco interfaces sugeridas pela WfMC, onde sua separação ocorre em função dos aspectos de operação que são realizados. As cinco interfaces definidas são: Interface 1 (responsável pelo formato e definição dos processos), Interfaces 2 e 3 (responsáveis pelo acesso a aplicações cliente e externas), Interface 4 (responsável pela integração com outros sistemas de *workflow*) e por fim a Interface 5 (interface responsável por prover o acesso às ferramentas de administração).

As características detalhadas de cada uma das interfaces sugeridas pela WfMC são enfocadas a seguir, onde temos:

- **Interface 1 – "Process Definition":**

  A princípio, qualquer ferramenta pode ser usada na definição de processos; porém, como foi citado anteriormente, deve-se ter cuidado para que padrões utilizados pelo motor e pela ferramenta de modelagem sejam compatíveis. Se a ferramenta de modelagem for fornecida em conjunto com o produto, a transferência de dados pode ou não respeitar o padrão. Caso seja utilizada uma ferramenta separada, os dados devem ser passados ao *enactment service* segundo o seu formato, e muitos dos produtos existentes ainda não estão de acordo com os padrões definidos.
  A primeira interface define um formato de intercâmbio e um conjunto de funções de API para acesso aos dados, é definido juntamente um ponto claro de separação entre ambientes de modelagem e execução, permitindo a combinação de ferramentas independentes.
  A API de acesso à definição de processos deve conter uma série de funções, podemos citar algumas delas. Suas funções são funções dos tipos:

  - Estabelecimento de sessão;
  - Definição do *workflow*;
  - Definição de objetos de uma modelagem.

- **Interface 2 – "Client Applications":**

  Define a interação entre as aplicações cliente e o *enactment service* através das *worklists*. Uma *worklist* pode conter itens relativos a várias instâncias ativas de um mesmo processo e/ou itens individuais de diferentes processos.
  Um *worklist handler* pode interagir com diferentes *engines* e *enactment services*.
  A interface entre clientes e *engines* deve oferecer funções para:

  - Estabelecimento de sessão;
  - Controle de processos e atividades;
  - Status de processos;
  - Manipulação da *worklist*;
  - Administração;
  - Invocação de aplicações.

  A camada de API's pode ser mapeada para vários mecanismos de comunicação, contanto que sejam desenvolvidos em linguagens compatíveis em seus padrões.

- **Interface 3 – "Invoked Aplications":**

  Esta interface define a interação do *enactment service* com outras aplicações externas necessárias para a realização de tarefas. Bem como, tipos de interface para invocação de aplicações externas passíveis de padronização, juntamente com tipos de passagem de dados relevantes.

  Alguns exemplos de chamadas e protocolos existem são:

  - Chamadas CORBA (parâmetro da chamada);
  - RPC (parâmetro da chamada);
  - Mensagens, como X.400 (por referência ou anexados);
  - Transações, como OSI-TP (por referência ou anexados).

  Algumas das funções possíveis da API são:

  - Estabelecimento de sessão;
  - Gerência de atividades;
  - Manipulação de dados.

- **Interface 4 – "Workflow Interoperability":**

  A WfMC define diferentes tipos de interoperabilidade entre produtos de *workflow*:

  - Discreto conectado (encadeado): conexão entre uma tarefa do processo A e uma tarefa do processo B, com transferência de um único item de trabalho (instância de processo ou atividade) e sem futura sincronização;
  - Hierárquico (aninhado): processo completo em um domínio é encapsulado em uma tarefa de um processo em outro domínio;
  - Ponto a ponto: ambiente misto onde um processo pode ter atividades executadas em diferentes *engines*. Os *engines* devem suportar uma API comum para comunicação e devem interpretar uma mesma definição de processos;
  - Paralelo Sincronizado: processos independentes com pontos de sincronização.

  Tipos de funções da API para interoperabilidade:

  - Invocação de atividade ou sub-processo;
  - Controle e status de atividades e processos;
  - Transferência de dados relevantes de aplicação e *workflow*;
  - Coordenação de sincronização;
  - Leitura e escrita de definição de processos.

- **Interface 5 – "Administration and Monitoring Tools":**

  Define a interação com ferramentas de administração e auditoria do sistema. Tipos de funções da API:

  - Gerência de usuários e papéis;
  - Gerência de auditoria (criação, consultas, impressão, entre outras);
  - Controle de recursos;
  - Supervisão de processos;
  - Gerência de status de processos.

Cada desenvolvedor de sistemas de *workflow* pode optar por quais interfaces serão disponibilizadas abertamente para outros produtos, o que será definido através de níveis de conformidade com o padrão WfMC.

Um dos maiores problemas atuais de modelagem de sistemas de *workflow* está, no fato de que cada sistema de gerência de *workflow* (WFMS) utiliza sua própria técnica de modelagem, contextualizado na Interface 1 de [WFM 96]. Não há um modelo conceitual amplamente aceito para sistemas de *workflow* como existe o modelo E-R (Entidade-Relacionamento) na área de banco de dados.

Devido a este fato, dois problemas podem ser observados:

1. De interoperabilidade:
   - As ferramentas de modelagem de *workflow* são geralmente associadas a um único WFMS;
   - Apesar de os modelos muitas vezes serem semelhantes, a linguagem de definição do *workflow* (*Workflow Definition Language* – WFDL) é sintaticamente diferente entre os vários produtos;
   - Não é possível modelar o *workflow* em um WFMS e implantá-lo em outra ferramenta.

2. De definição, ou seja, existem profunda diferenças teóricas sobre o que é o trabalho e como ele se desenvolve dentro de um determinado ambiente. Estas diferenças permitem dividir os modelos de *workflow* em dois grandes grupos:

   - **Modelos baseados em Comunicação**: é visto como um conjunto de interações humanas bem definidas (laço de trabalho), representando compromissos realizados entre as pessoas envolvidas. O modelo de Ações encontra-se nesta categoria.
   - **Modelos baseados em Atividades**: é visto como uma seqüência de atividades, onde cada atividade recebe um certo conjunto de entradas e produz um certo conjunto de saídas. Os modelos de gatilhos e de Casati/Ceri [CAS 95] encontram-se nesta categoria.

Após esta explanação sobre as arquiteturas de *workflow*, podemos perceber que existem muitos padrões propostos; porém, ainda não se tem definido, de fato, a forma como serão modelados, estruturados, gerenciados e manipulados os ambientes para o desenvolvimento de sistemas de *workflow*.

Sobre este contexto de indefinições que está calcado todo este estudo; pois, parte dos problemas que abordados pelo escopo do trabalho advém desta falta de padrões que existe de forma generalizada sobre as ferramentas de *workflow* existentes.

1.4 Estudos e Trabalhos que Motivaram o Desenvolvimento deste Estudo

Vários foram os fatores que incentivaram e nortearam a definição deste estudo; porém a principal motivação decorre da experiência adquirida com a participação de projetos e definições para ferramentas de *workflow*, e consolidada no desenvolvimento de várias aplicações, desenvolvidas com o enfoque sobre a Arquitetura Baseada em um Único Ambiente para Modelagem e Execução dos processos.

Uma vez iniciados os projetos para a seleção de um ambiente de *workflow*, que estivesse enquadrado na Arquitetura Baseada em Ambientes Distintos para Modelagem e Execução, uma análise sobre a sua arquitetura de desenvolvimento e execução foi necessária. Este projeto principal selecionou aproximadamente 30 ferramentas de *workflow*, realizando um levantamento e analisando suas características. Destes 30 fornecedores, 4 implementaram um protótipo sugerido, demonstrando o processo de modelagem e desenvolvimento das aplicações.

Ao analisar detalhadamente cada etapa da realização dos sistemas, ficou evidente a falta de uma definição, por parte dos fabricantes, sobre os conceitos relativos às aplicações e aos processos modelados; pois ambos os conceitos estavam dissociados na modelagem e implementação das aplicações. Esta falta de padronização sobre os conceitos de aplicações e de processos, para o ambiente analisado, gerou a necessidade de um estudo mais aprofundado de modo a levantar as dificuldades que seriam encontradas em função destas características.

Uma vez percebia a necessidade desta separação entre aplicações e processos, foram iniciados os estudos sobre a arquitetura e, como etapa inicial para a viabilidade de se gerar esta separação esbarrou-se na dependência existente entre as aplicações e os processos modelados. Neste ponto, surgiu a proposta de criar uma camada que se integrasse ao motor de *workflow*, de modo a propiciar a independência necessária.

Definida a elaboração desta camada, a próxima etapa foi definir o escopo e a abrangência que seria realizada, a decisão foi a de elaborar a estrutura de funcionamento e uma análise aprofundada da arquitetura e o desenvolvimento de um protótipo.

2 Arquiteturas de "Workflow"

Este estudo sobre as arquiteturas de *workflow* tem como foco principal, o modo como as aplicações estão contextualizadas durante o seu desenvolvimento e sua execução, no ambiente fornecido pelas ferramentas de *workflow*. Em decorrência deste escopo, ficam colocadas em um segundo plano, questões como: padrões de modelagem, performance dos sistemas, portabilidade, escalabilidade, segurança; entre outras características importantes sobre a arquitetura de sistemas.

Com o escopo definido - o modo como são implementadas as aplicações, sobre estes ambientes – podemos iniciar a análise separando, a partir do enfoque de modelagem e execução, as arquiteturas de *workflow* em duas arquiteturas distintas, que são: Arquitetura Baseada em um Único Ambiente para Modelagem e Execução e Arquitetura Baseada em Ambientes Distintos para a Modelagem e Execução.

O objetivo da análise é definir o comportamento, as dificuldades e facilidades encontradas para cada uma das arquiteturas, visando a modelagem e posterior utilização, pelas aplicações, dos recursos apresentados pelo ambiente de *workflow*. A seguir são enfocadas, separadamente, cada uma das arquiteturas.

2.1 Arquitetura Baseada em um Único Ambiente

Esta forma de desenvolvimento consiste em uma arquitetura de *workflow* em que toda a aplicação esta concentrada em um mesmo ambiente, tanto para construção quanto para execução dos processos. Na maioria dos casos, não se tem disponível um módulo para a modelagem dos fluxos e, uma vez implementado o sistema, somente pode-se utilizá-lo no próprio ambiente em que foi desenvolvido. Estas são as características dos primeiros ambientes de desenvolvimento em *workflow*; também caracterizados, segundo alguns autores como sistemas de *groupware*, dentre alguns dos autores que podemos citar, temos: [BOR 95], [BER 97] e [HIL 97].

Como exemplo desta arquitetura, temos, o ambiente Lotus Notes, onde todo o desenvolvimento está centrado em um arquivo único, de extensão ".nsf" (*Notes Storage Facility*), que representa todo o código da aplicação. No caso do Lotus Notes não existe uma ferramenta para a modelagem dos fluxos, o fluxo é implementado através de *flags* de controle (variáveis associadas a situações ou passos realizados no transcorrer do processo), que devem ser controlados pela aplicação de acordo com os encaminhamentos que estão sendo realizados.

Atualmente, dentro da arquitetura do Lotus Notes, está sendo lançada uma ferramenta que se propõem a realizar a modelagem dos processos; porém ela ainda está muito incipiente e não contempla o conceito proposto pelos padrões sugeridos pela WfMC, como arquitetura para o desenvolvimento dos processos. Esta ferramenta trata-se do Domino *Workflow*, que atualmente está em suas versões iniciais, e não dispõe de uma camada de API's, trabalhando assim, somente relacionada à arquitetura do Lotus Notes. E em função disso, ao menos por enquanto, não podemos considerá-la como uma ferramenta que se enquadre na próxima arquitetura que iremos analisar, a Arquitetura Baseada em Ambientes Distintos para Modelagem e Execução.

Nesta arquitetura, de ambiente único, o desenvolvimento do fluxo não segue padrões de modelagem ou uma forma padronizada de representação das tarefas que estão sendo modeladas, salvo empresas que tenham investido na elaboração de uma infra-estrutura para esta padronização. O processo simplesmente é analisado e, implementada uma solução que o represente, sem que exista uma ferramenta disponibilizada pelo ambiente para a uma estruturação. A lógica do fluxo modelado é implementada no interior da aplicação, e esta implementação ocorre através de: consistências existentes em botões, campos com controle de acesso, campos com o controle do fluxo, visualizações de documentos entre outros elementos característicos de cada ambiente. A lógica está diluída nos elementos componentes das aplicações, como parte integrante de seu código.

Os ambientes que trabalham com esta arquitetura não têm bem definidos os limites entre fluxo e aplicação; pois, se fundem ambos os conceitos. Sendo assim, impossibilitando uma separação do código que trata da lógica referentes à regra de negócio e o código que trata da lógica referente às regras dos processos. Temos com isso um código único responsável por toda a lógica do sistema.

Podemos analisar esta arquitetura de *workflow*, como sendo um ambiente em que é disponibilizada uma série de elementos que facilitam o desenvolvimento dos controles e tarefas. Porém, não dispõem de um elemento responsável pela estruturação dos processos. Temos na Figura 3 um diagrama desta arquitetura, tendo como ponto de vista um conjunto de elementos (subsídios para infra-estrutura) disponibilizados.

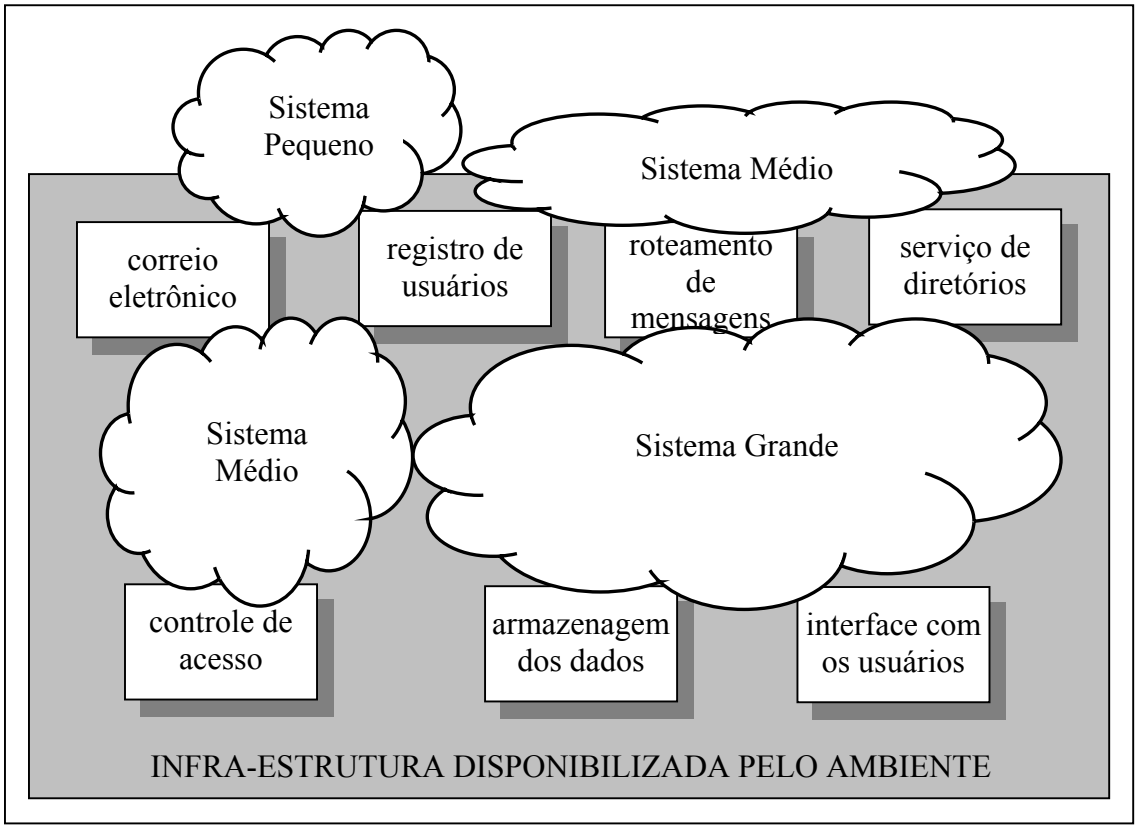


FIGURA 3 - Modelo da Arquitetura Baseada em um Único Ambiente

Temos, na Figura 3, as aplicações e os elementos componentes do ambiente em um mesmo contexto, como se não tivéssemos a separação da lógica que faz parte de código desenvolvido pela aplicação – regra do negócio – e a lógica que faz parte do ambiente de *workflow* – regras dos processos. Podemos perceber que as aplicações simplesmente se mesclam com os elementos existentes do ambiente do *workflow*. Assim sendo, teremos aplicações que utilizarão um conjunto maior de elementos – sistemas grandes – e aplicações que utilizarão um menor conjunto destes elementos – sistemas pequenos. No diagrama da Figura 3 foram esboçados apenas alguns dos elementos componentes do ambiente, e vale ser lembrado, que a quantidade destes elementos oferecidos irá depender da infra-estrutura disponibilizada pelo ambiente.

Utilizando o ambiente Lotus Notes como exemplo, teríamos para a utilização de suas aplicações o seguinte contexto:

- Seu servidor de correio padrão SMTP;
- Seu livro de endereços, armazenando os usuários;
- Roteamento entre os servidores;
- O serviço de diretórios, utilizado os grupos no livro de endereços;
- ACL (Lista de Controle de Acesso) existentes em todas as aplicações.

Foram enfocadas algumas características específicas do Lotus Notes, porém todos os ambientes que compartilham desta arquitetura seguem, com suas peculiaridades, estes padrões. Sendo assim temos um ambiente isolado com sua infraestrutura para o desenvolvimento das aplicações sem qualquer intenção de ser utilizado por um ambiente distinto.

Alguns exemplos de ferramentas que utilizam essa arquitetura são:

- Lotus Notes (IBM);
- Exchange (Microsoft);
- GroupWise (Novell).

Podemos, de forma a resumir esta arquitetura, enumerar as principais vantagens e desvantagens na utilização desta arquitetura - Modelagem e Execução em um Único Ambiente - para o desenvolvimento de aplicações em *workflow*.

A principal vantagem desta arquitetura é:

- **Linguagem única:**
  Necessidade de domínio de somente uma linguagem de desenvolvimento para a implementação dos sistemas.

As principais desvantagens desta arquitetura são:

- **Falta de uma ferramenta de modelagem (para alguns casos):**
  Devido à falta de uma ferramenta de modelagem não se têm facilidades para a modelagem dos processos de forma padronizada, podendo gerar, caso não seja realizada a definição de padrões, complexidade na modelagem e posterior manutenção dos processos. Como o enfoque principal não está em uma modelagem complexa, para muitas ferramentas não está disponível um módulo para elaboração dos fluxos;

- **A impossibilidade de integrar os conceitos de *workflow* a sistemas existentes em outras linguagens e ambientes:**
  Para sistemas já existentes torna-se necessário desenvolvê-los novamente sobre este ambiente de *workflow*;

- **Código gerado complexo e de difícil manutenção;**
  Apesar de termos apenas a própria linguagem do ambiente de *workflow*, o código torna-se complexo, isso ocorre em função da não separação entre a lógica das aplicações – regras do negócio – e a lógica dos processos – regras do fluxo.

2.2 Arquitetura Baseada em Ambientes Distintos

Em meados dos anos 80, quando se iniciou o desenvolvimento de sistemas de *workflow* utilizava-se o próprio ambiente de *workflow* como ambiente para aplicação, não havendo uma separação entre as duas camadas (a aplicação e as tarefas em *workflow*). Com isso o sistema era escrito de acordo com o produto com que estávamos trabalhando. Isso geralmente ocorria em função de se estar pensando em *workflow* como sendo um sistema novo a ser implementado. O sistema seria totalmente feito em um único ambiente e com o objetivo específico de um sistema para o controle dos fluxos, este contexto é onde está inserida a arquitetura exposta anteriormente.

No momento em que iniciou a expansão dos conceitos de *workflow*, começaram a surgir necessidades maiores do que as iniciais. Desde então, o enfoque dado às aplicações começou a se modificar, uma vez que passa a ser percebida, para praticamente todos os sistemas existentes, alguma etapa específica de fluxo. A partir deste ponto foi iniciada uma nova etapa no desenvolvimento de sistema em *workflow*, que objetivava "agregar" conceitos de *workflow* em sistemas já existentes. A partir deste ponto os ambientes de *workflow* tinham a necessidade de trabalhar baseados em aplicações existentes, surgindo assim o conceito de motor ou *engine* de *workflow*, que teria o papel semelhante a um "servidor" para os fluxos modelados e gerenciador dos documentos associados a estes fluxos.

Neste novo conceito seria possível adequar os sistemas existentes, permanecendo eles em suas plataformas de desenvolvimento iniciais, a um contexto de *workflow*. Esta adaptação tornou-se possível através da camada de API's dos motores, uma vez que fazem o papel de uma camada responsável pelo controle dos processos criados a partir da aplicação. As aplicações referenciam esta camada de API's e requisitam as informações que necessitam sobre os documentos.

Deve ser mencionado, que esta nova arquitetura não isenta as aplicações de modificações, visando à adequação ao novo modelo, de um ambiente contextualizado em um motor de *workflow*. O que a nova arquitetura vem permitir é a permanência de plataformas de execução e linguagens utilizadas para o seu desenvolvimento. Assim sendo, as aplicações não necessitam ser migradas para um novo ambiente, apenas o seu código ser adequado de forma a transferir os controles sobre o fluxo, que estavam a cargo das aplicações, para o motor de *workflow*.

A seguir temos um diagrama representando a arquitetura de um motor de *workflow*. São apresentados os seus elementos componentes e como as aplicações estão inseridas neste contexto.

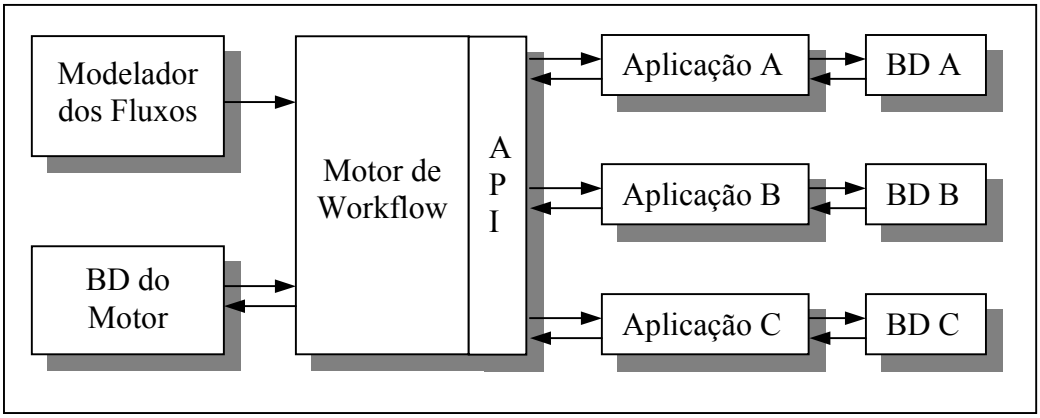


FIGURA 4 - Arquitetura de um motor de *workflow*

Na Figura 4 temos uma estrutura representando a arquitetura do motor de *workflow*, onde estão representados:

- **Motor ou *engine*:**
  O coração da arquitetura, ele é um serviço disponibilizado e funciona como um servidor, este processo é o responsável pelo controle e atendimento das requisições feitas pelas aplicações. As aplicações, através da camada de API's criam uma nova instância para um processo específico, ou seja, um novo processo é disparado. A partir deste momento, o motor de *workflow* assume a responsabilidade pelo controle e gerência desta nova instância. Outra tarefa do motor é disparar eventos relacionados à mudança do estado do processo, ocorrida em função de algum período de tempo especificado ou em função da interação realizada pelos usuários, sobre o processo;

- **A camada de API's:**
  A partir da qual as aplicações fazem referência aos processos modelados, as API's são de extrema importância; pois, de acordo com suas características, as aplicações terão maior capacidade de manipulação dos dados e documentos. Cada ambiente de *workflow*, atualmente no mercado, tem seu conjunto específico de funções ao acesso dos dados componentes do fluxo, não existe uma padronização sobre os métodos necessários ou básicos para todos os sistemas. Alguns padrões são sugeridos pela WfMC, na forma de um conjunto de funções necessárias, porém estas recomendações não são seguidas com o rigor que deveriam;

- **As aplicações:**
  Uma vez que a aplicação está acessando o motor através de sua camada de API's, a aplicação pode ser desenvolvida em qualquer ambiente ou linguagem de programação (contato que aceite chamadas a estas API's). O fato de ser realizadas requisições sobre os documentos ao invés de ter todo o controle no código da aplicação é um dos grandes diferenciais desta arquitetura. É de conhecimento da aplicação a maneira como devem ser requisitadas as

informações sobre as instâncias que estão sendo manipuladas naquele momento. Parte das informações são de conhecimento do motor e parte do conhecimento fica a cargo da aplicação, isso é variável de acordo com cada um dos produtos utilizados e também a análise realizada; porém, nenhuma das ferramentas existentes no mercado atualmente tem uma separação clara sobre até que ponto o controle é feito pela aplicação, e até que ponto as informações estão sendo controladas pelo motor. O grande problema desta falta de definição é que ela acaba gerando uma dificuldade tanto no momento da análise quanto no momento da implementação, pois alguns elementos podem ser de responsabilidade tanto do motor quanto da aplicação;

- **Modelador de fluxos:**
  O modelador de fluxos é de grande importância para um ambiente de *workflow*, no modelador de fluxo é onde o fabricante irá demonstrar qual metodologia e representação dos fluxos está sendo adotada (vale lembrar que atualmente temos várias propostas de padrões de modelagem, pode ser citado aqui o modelo proposto por Casati/Ceri [CAS 95], como exemplo de um dos mais completos, outros modelos que também poderiam ser citados são os propostos por [AAL 95] (baseado em redes de Petri), [JOO 94] (como seus modelo de Gatilhos) e também [BAR 95] (modelo semelhante ao de Gatilhos); porém nenhuma destas propostas é, de fato, seguida), caso o modelador seja pobre em recursos, a modelagem feita não poderá ter características muito complexas. Após ser realizada toda a modelagem do fluxo, esta modelagem será transferida (geralmente através de scripts) para o motor e, a partir deste momento, o motor terá a capacidade de instanciar novos processos para o fluxo modelado;

- **Banco de dados do motor:**
  O motor irá trabalhar de acordo com uma estrutura de banco de dados própria, a maioria dos motores trabalham com banco de dados relacionais (como Oracle ou SQL Server). É possível alterar os dados contidos no banco, porém recomendam-se modificações apenas através das API's, pois assim não se corre o risco de gerar inconsistências nos dados. Todas as ferramentas estudadas no mercado utilizam uma estrutura aberta para a manipulação dos dados armazenados sobre as instâncias, é possível após ter conhecimento suficiente de suas estruturas ousar modificações diretamente sobre o seu banco de dados. As modificações realizadas diretamente sobre o banco teriam sentido talvez em casos específicos em que se tenha uma grande necessidade de performance, uma vez que as modificações realizadas diretamente sobre o banco não utilizam a camada de API's do produto;

- **Banco de dados das aplicações:**
  O banco de dados da aplicação fica a cargo da própria aplicação, não é relevante para o motor o tipo de banco de dados que se está utilizando; pois, o motor referencia somente o seu próprio banco. O banco da aplicação fica de responsabilidade da aplicação, apenas deve-se ter a chave que relaciona o registro da aplicação com o registro que representa aquela instância no banco do motor. Este controle das chaves, gerenciando os relacionamentos, para algumas ferramentas acontece de maneira transparente, não onerando as aplicações deste controle.

Alguns exemplos de ferramentas que utilizam essa arquitetura são:

- Action Workflow Process Builder (Action Techonologies);
- AIM/Workflow (Intergraph);
- Computron Workflow (Computron Software);
- COSA Workflow (Ley GmbH);
- CSE (CSE Systems);
- Eastman Software WorkFolder (Eastman Software - Kodak);
- FlowMark (IBM);
- Genexus Workflow (Genexus);
- I-Flow (Fujitsu);
- INCOME Workflow (PROMATIS);
- InConcert (InConcert - Xerox);
- Intempo (jetForm);
- J-Flow (Workflow Automation Corporation);
- METEOR (Large Scale Distributed Information Systems Lab.);
- Metro Action Works (Action Technologies);
- MQSeries Workflow (IBM);
- Multidesk Workflow;
- Oracle Workflow (Oracle)
- Panagon Visual Workflo (FileNet);
- Staffware 2000 (Staffware);
- Step2000 (UDMS – Universal Document Management Systems, Inc.);
- TeaWARE Flow2.0 (Fujitsu);
- Ultimus (Ultimus);
- WFX (Eastman Software);
- W4 (World Wide Web Workflow).

As principais vantagens desta arquitetura são:

- **Diferentes linguagens de programação ou ambientes:**
  Possibilidade de integrar os conceitos de *workflow* a sistemas implementados em diferentes linguagens de programação ou mesmo ambientes de execução, gerando flexibilidade para empresas que desenvolvem seus sistemas em diversos ambientes e tecnologias. Outro ponto positivo é a não necessidade de se reescrever o código em uma nova linguagem, para as aplicações existentes que necessitam conceitos de *workflow*. Essa flexibilidade existe uma vez que podem ser adaptadas ao invés de serem reescritas para uma linguagem específica;

- **Ferramenta para modelagem dos fluxos:**
  É extremamente importante existir uma ferramenta capaz de gerar definições gráficas sobre o processo que se está modelando, tanto para documentação, quanto para o momento das definições de processo, etapa em que é realizada a análise junto aos usuários;

- **Reutilização dos processos existentes:**
  Se a análise for realizada corretamente se terá a reutilização dos processos, definidos previamente. Este item é dependente do modo como forem definidos os processos e o enfoque adotado. Desta forma, utilizando um dos principais conceitos defendidos pela análise e programação orientada a objetos, onde temos o reuso dos códigos e análises, possibilitando assim um ambiente capaz de agregar o conhecimento já existente, de forma a minimizar ao máximo o re-trabalho. Podemos citar o enfoque dado por [MAR 94], onde descreve sobre reaproveitamento: "Um repositório deve ser constituído de uma coleção sempre crescente de classes reutilizáveis. Bibliotecas de classes normalmente crescem rapidamente. Um dos principais objetivos das técnicas baseadas em objetos é o de se conseguir reaproveitamento em massa na construção de software". Podemos claramente substituir sua referência a "classes" por "processos" ou "fluxos".

As principais desvantagens desta arquitetura são:

- **Falta de padrões para a arquitetura de desenvolvimento:**
  Deve-se analisar, de forma criteriosa, o modo como serão implementados os sistemas; pois temos, nesta arquitetura, uma grande flexibilidade com relação a linguagens e ambientes. Em função desta flexibilidade, podem ser geradas facilmente muitas divergências na estrutura dos sistemas sobre ela implementados. Acarretando, desta forma, uma complexidade que poderia ser evitada, ou ao menos minimizada por padrões seguidos para a análise e desenvolvimento;

- **Falta de integração entre diferentes motores:**
  Considerando o motor como "coração" do *workflow*, seria desejável que fosse enfocada a integração entre diferentes fornecedores de motores de *workflow*; porém isso não acontece. Não existe a intenção por parte dos fabricantes de tornar os produtos compatíveis em seus conceitos e estruturas de armazenamento; gerando assim, uma dependência do fornecedor contratado;

- **Falta de padronizações para a modelagem de representação dos fluxos:**
  Apesar de existir a etapa de modelagem e ela ser contemplada com uma ferramenta CASE geradora de fluxos, cada fabricante dispõe de seus métodos e características de modelagem. Da mesma forma, não existe uma norma ou padrão que seja respeitado com relação à etapa de modelagem dos processos.

3 O Enfoque das Aplicações para cada uma das Arquiteturas

Após expor ambas as arquiteturas de *workflow*, há um ponto importante a ser analisado, o enfoque das aplicações para cada uma das arquiteturas. Este enfoque em cada uma das arquiteturas é diferenciado e deve ser bem compreendido; pois, o problema da dependência - abordado adiante - está diretamente relacionado com o modo como as aplicações estão estruturadas.

3.1 Aplicações para Arquiteturas Baseadas em um Único Ambiente

Para a arquitetura em que há um único ambiente, a separação entre *workflow* e aplicações praticamente não existe. Todo o controle, tanto referente à própria aplicação quanto ao fluxo do *workflow*, está colocado como um código único. A aplicação é responsável por gerenciar todas as atividades que serão realizadas juntamente com todos os controles sobre o fluxo e os eventos necessários.

Para esta arquitetura o ambiente de *workflow* está presente, subsidiando a intraestrutura para alguns elementos, isentando a aplicação do desenvolvimento de certas etapas, porém o controle das estruturas permanece com a aplicação. Na Figura 5, estão ilustrados: o escopo da aplicação e os elementos por ela gerenciados.

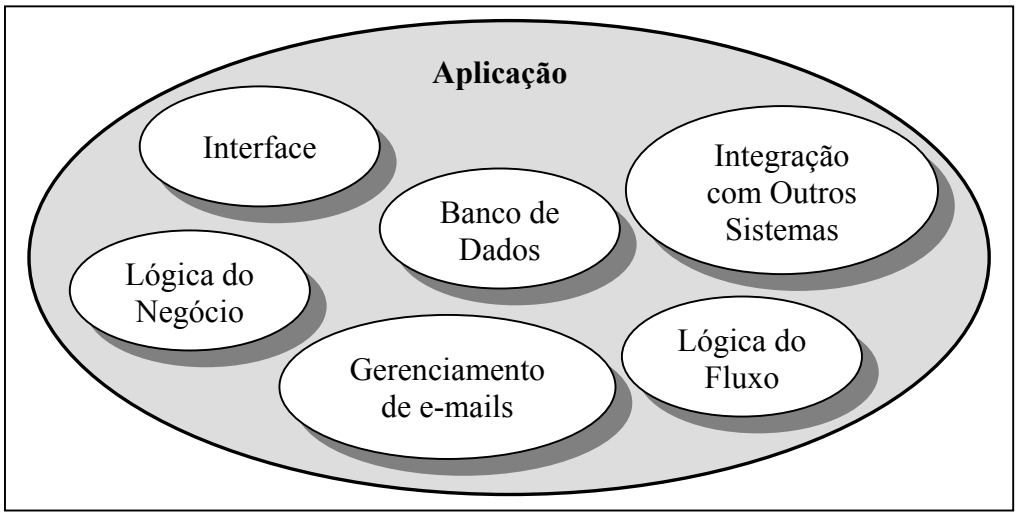


FIGURA 5 - Escopo da aplicação para arquitetura com ambiente único.

Na Figura 5 percebe-se um número elevado de elementos gerenciados pela aplicação. Isso decorre em função de existirem elementos de *workflow* associados às tarefas a serem controladas pela aplicação. Todo o controle, tanto da lógica da regra de negócios, quanto da lógica dos processos está sob sua responsabilidade. Uma aplicação neste contexto consiste em uma aplicação extensa, seu código se torna longo e complexo, pois estão sendo agregadas as regras da aplicação e do fluxo em um único momento.

3.2 Aplicações para Arquiteturas Baseadas em Modelagem e Execução Distintas

Em muitos casos, o código das aplicações desenvolvidas não é evidenciado como uma peça importante, no conjunto da solução de um processo de *workflow*. Geralmente o que ocorre é um enfoque mais direcionado para a ferramenta de modelagem do fluxo. O CASE, ou ferramenta de modelagem utilizada, é de extrema importância, porém, o aspecto de como as aplicações interagirão com os processos modelados deve ser analisado conjuntamente, não sendo deixado como uma etapa posterior ou como uma etapa desvinculada do processo que está sendo modelado.

O objetivo não é elaborar o fluxo de acordo com o modo como as aplicações serão implementadas, mas analisar o trabalho que será necessário para utilizar o fluxo modelado. Esta dificuldade, em se utilizar os processos pelas aplicações, não deve ser transferida para a modelagem, mas sim, tratada pela camada de API's disponibilizada pelo motor de *workflow*.

Comparando o controle necessário para as aplicações baseadas neste contexto, existe uma sensível diferença com relação às aplicações contextualizadas na arquitetura exposta anteriormente. Basicamente o enfoque é o de que, a aplicação não é responsável pelo controle de elementos referentes ao *workflow* como era necessário na Arquitetura Baseada em um Único Ambiente.

As aplicações devem estar comprometidas somente com as suas regras de negócio, não tendo necessidade de controlar qualquer elemento referente ao fluxo. Definindo, assim, o seu escopo à própria lógica do negócio; desta forma, desvinculando-se da lógica do fluxo, que representa o processo que se está utilizando. Na Figura 6 temos um diagrama exemplificando alguns dos elementos que compõe o escopo das aplicações.

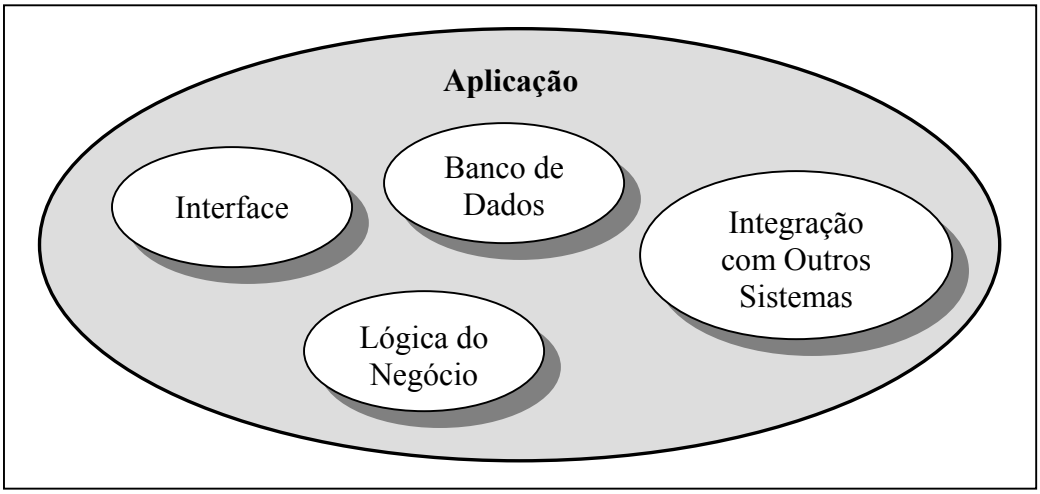


FIGURA 6 - Escopo da aplicação para arquitetura com ambientes distintos.

Na Figura 6 temos um número menor de elementos, comparando-se com a Figura 5, onde tínhamos dentro do mesmo escopo, juntamente com os elementos referentes à aplicação, os elementos referentes ao fluxo.

Esta diferença existente entre as Figuras 5 e 6, exemplifica um dos objetivos da arquitetura baseada em ambientes distintos de modelagem e execução: retirar o controle dos processos do interior das aplicações e transferi-lo ao motor de *workflow*. Reservando assim, para as aplicações apenas o controle das regras de negócios específicos das mesmas.

4 Dependência entre as Aplicações e o Fluxo Modelado

Partindo do princípio que cada uma das arquiteturas citadas tem suas vantagens e desvantagens, iremos neste capítulo analisar as características de dependência das arquiteturas com relação às aplicações desenvolvidas. Será principalmente enfocada a Arquitetura Baseada em Ambientes Distintos para Execução e Modelagem, uma vez que a proposta deste trabalho baseia-se nesta arquitetura de desenvolvimento de sistemas em *workflow*.

Outra forte razão de não ser salientada com muita ênfase a dependência na arquitetura de modelagem e execução em um único ambiente é o fato de que nestes ambientes, não se tem a separação das camadas de fluxo do processo e da aplicação.

Deve-se entender por dependência, entre aplicações e fluxo modelado, o quão necessário torna-se modificar o código associado à execução da aplicação, no momento em que o fluxo do processo sofre alterações. Estas alterações tendem a ser algo relativamente constante, ou ao menos uma etapa prevista, devido às características de dinamicidade dos sistemas de *workflow*. Estas características são vistas com maior detalhe no capítulo seguinte.

Não dispomos de métricas precisas para definir, enquadrando-se em padrões, o quão dependente uma aplicação está de um fluxo. Um modo que pode ser utilizado é definir cada um dos elementos existentes sobre um fluxo de *workflow*; e para cada um destes elementos, verificar se uma modificação específica sobre ele repercute sobre as aplicações. Deste modo podemos ter uma noção, para cada um dos elementos, a dependência com o código da aplicação. Agregando assim, meios de categorizar quanto uma aplicação, ou seu código, está dependente do processo.

Ao definirmos estes elementos nos deparamos com um problema: Os elementos de *workflow* têm uma variação dependendo do ambiente analisado. Algumas ferramentas dispõem de mais recursos, ou seja, uma maior infra-estrutura, para o desenvolvimento dos fluxos, tornando assim menos oneroso o código da aplicação. Ao contrário, outras ferramentas, disponibilizam uma menor infra-estrutura, gerando um maior trabalho para as aplicações. Em função desta diversidade, será utilizado um conjunto de elementos obtidos de diferentes ferramentas e ambientes de formar a ter um universo maior destes elementos para a análise.

Nos casos em que não existe uma correspondência entre os elementos definidos e o ambiente de *workflow*, este elemento será considerado como um controle a cargo da aplicação, tornando assim uma modificação diretamente dependente da aplicação, uma vez que não está sendo controlada pelo ambiente de *workflow*.

Dentro do conjunto de elementos que poderíamos definir estariam:

- Atividades ou Estados
  Temos uma diferença entre conceitos neste elemento, pois alguns fabricantes têm o fluxo, representado em sua ferramenta, baseado em um modelo de atividades e outros fabricantes, baseado em um modelo de estados. Porém o que é relevante neste momento é o fato deste elemento representar o fluxo, propriamente dito, ou seja, as etapas que ocorrem no processo;

- Ações:
  As ações associadas ao processo, o que permitirá aos usuários manipular os documentos dentro do processo, são exemplos de ações: o envio de documentos para próxima etapa no fluxo, uma aprovação ou uma revisão. Assim sendo, as ações realizadas pelos usuários, através da interação pelo sistema, sobre o fluxo;

- Papéis, Regras ou Grupos:
  A definição de grupos de regras ou papéis sobre o fluxo. Podemos ter papéis ou regras associadas diretamente sobre um usuário ou sobre um grupo de usuários. Este grupo pode ser interno do ambiente do *workflow* ou alimentar-se de um organograma definido pela organização;

- Controle de Acesso:
  Para o controle de acesso existem diferentes contextos e níveis, o controle pode ser em nível de processo, de campo, de visualizações, ou de manipulação sobre os processos. São encontradas muitas diferenças para o controle de acesso entre os vários ambientes existentes;

- Códigos associados às ações:
  O código associado a cada ação realizada pelos usuários, na maioria dos casos este é um elemento bem constante, não necessitando de modificações, de acordo com o contexto do ambiente ele ficará a cargo da aplicação ou controlado pelo motor de *workflow* (o que é mais interessante).

4.1 Dependência para as Aplicações com Ambiente Único

Como descrito anteriormente, para ambientes onde temos modelagem e execução em um mesmo contexto, temos dificuldade em delimitar onde está implementado o fluxo dos processos e, onde está definida a aplicação que dele faz uso. Em função desta falta de delimitação, torna-se evidente a total dependência entre a aplicação desenvolvida e o fluxo modelado.

Na Figura 7, é apresentado um diagrama onde se pode perceber esta dependência; pois baseado em uma representação de conjuntos, temos uma grande parte dos elementos comuns a lógica dos fluxos e a lógica das aplicações.

FIGURA 7 - Diagrama da Arquitetura Baseada em um Único Ambiente

Na Figura 7 temos a elipse da esquerda representando à lógica do fluxo dos processos e a elipse da direita representando a lógica das aplicações. Percebe-se a grande fusão que existe entre a aplicação e fluxo dos processos. Praticamente não existe separação entre as ambas as lógicas. Nesta arquitetura temos a dependência como sendo uma característica da arquitetura, pois não existe uma separação formal, o objetivo não é desmembrar a aplicação do fluxo do processo.

Alguns conceitos e dificuldades são melhores expostos utilizando-se exemplos, para isso será representado como seria implementada uma parte do código para este ambiente único de modelagem e execução.

Vamos utilizar como exemplo o contexto, enfocando a programação necessária, de um botão ao qual estão associadas às regras do processo e de controle de acesso: O botão tem o código associado à ação de aprovar um documento que está na situação de "Pendente". O acesso a esta aprovação é dado para todos usuários associados ao papel de "Aprovador". Ao ser tomada a ação deve ser atualizado o estado do documento e enviada uma mensagem, via e-mail, para os participantes do fluxo. Está sendo utilizada a nomenclatura por vezes de "processo" e por vezes de "documento", porém permanece o mesmo significado, uma instância em execução sobre determinado processo.

As etapas realizadas pela aplicação seriam:

1. **A aplicação verifica se a condição atual do documento é a de "Pendente";**
   Estão definidos valores para um campo de controle (como exemplo o *flag* "Status") e para cada valor está associada uma situação do documento (0-Elaboração; 1-Pendente; 2-Finalizado). A aplicação controla de acordo com sua implementação a situação dos documentos por este campo "Status". Caso o valor do campo para o documento for igual a "1" então a aplicação "compreende" que sua situação é de "Pendente".

2. **A aplicação verifica se o usuário atual está associado ao papel de "Aprovador";**
   Após verificar a situação do documento a aplicação terá de confirmar se o usuário pertence ao papel de "Aprovador", novamente a aplicação "entende" que para a ação ser disponibilizada através de um botão que pode ou não ser habilitado, é necessário o usuário pertencer ao papel específico. Esta regra é validada e para isso a aplicação necessita ter conhecimento dos papéis existentes e as pessoas e/ou grupos associados a cada um deles.

3. **É exibido o botão de ação para que o usuário execute a ação;**
   Após todas as validações terem sido examinadas pela aplicação, é disponibilizado o botão ao usuário, de acordo com sua interface e seus padrões, podendo este ser um botão, um link, um objeto *radio*, ou qualquer outro elemento da interface. O importante é que a aplicação disponibiliza a ação para que ela seja realizada pelo usuário.

4. **É executado o código associado à ação realizada.**
   No momento em que o usuário realizar a ação o código associado é executado, cabe lembrar que o código está na aplicação e não sendo referenciado em algum outro local (como um código armazenado em algum repositório de funções ou alguma maneira análoga). O código a ser executado faz parte da aplicação, no caso de uma linguagem interpretada, os comandos estarão descritos no código da aplicação ou para uma linguagem compilada seu código será parte integrante do código executável.

Com este exemplo é exposta a dependência que existe entre a aplicação e a modelagem do fluxo, pois qualquer modificação realizada, em qualquer dos elementos definidos (estados dos documentos, ações, papéis ou grupos de usuários, controles de acesso, ou mesmo códigos de execução) repercutirá diretamente no código da aplicação. Esta dependência é uma característica própria das aplicações contextualizadas nesta arquitetura (Modelagem e Execução em um Ambiente Único).

De acordo com o modo como forem estruturadas as aplicações, o custo associado às modificações poderá ser minimizado, porém sempre haverá o custo de modificar o seu código. Essa característica não pode ser modificada e, torna necessário para o desenvolvedor, além de domínio da ferramenta, compreender a lógica que foi utilizada para a implementação do fluxo dos processos. Esta necessidade por parte dos desenvolvedores existe, uma vez que o fluxo está implementado através de *flags* e lógicas na aplicação. Assim, torna-se necessário o entendimento de cada um dos campos de controle de fluxo, de controle de acesso, controle de permissões para as ações e demais elementos do sistema.

4.2 Dependência para as Aplicações com Ambientes Distintos

Para contextos de desenvolvimento em que há aplicações podendo ser utilizadas em um ambiente ou linguagem e o motor de *workflow* sobre outro contexto, é possível comparar os conceitos relacionados a esta dependência de forma mais simples. Esta facilidade existe em função da própria natureza de contextos diferenciados, pois agora, existe uma separação formal entre aplicação e motor.

Apesar de termos, em função da arquitetura utilizada, uma separação bem característica entre aplicação e fluxo dos processos, ao analisarmos a arquitetura com enfoque sobre o desenvolvimento das aplicações esta separação, antes clara e explícita, começa a se confundir em seus limites entre aplicação e motor. Quando estamos modelando um processo algumas características começam a se confundir entre o que deveria estar sendo modelado no fluxo e o que deveria estar sendo implementado na aplicação. Como exemplo, ao modelarmos uma tarefa ou atividade devemos colocar as condições para a execução desta tarefa no fluxo ou na aplicação? O que acontece é que as condições acabam por serem colocadas na modelagem do processo e em um segundo momento, surge a necessidade de colocar as mesmas condições na aplicação. Ou seja, parte da lógica do fluxo sobre o processo acaba estando replicada no código das aplicações.

O ponto crucial para estudarmos, a dependência existente, está totalmente relacionado com a API disponibilizada pelo ambiente de *workflow*. Uma vez que a integração entre as aplicações e o núcleo do *workflow* se dá através da camada de funções disponibilizadas por esta API.

As informações disponibilizadas pela API serão as informações que anteriormente foram modeladas no fluxo, ou seja, devemos ter um módulo muito completo para a representação dos fluxos de modo a poder gerar subsídios para a camada de API retornar tais informações.

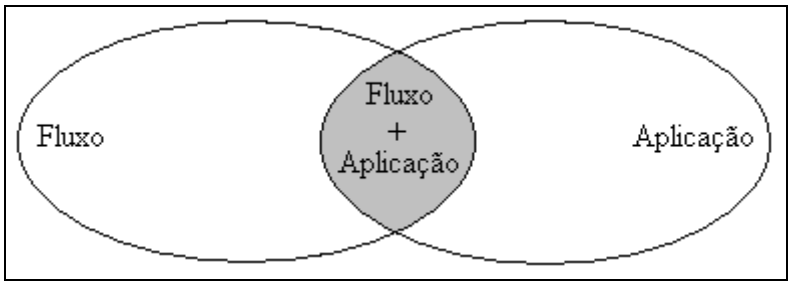


FIGURA 8 - Diagrama da Arquitetura Baseada em Diferentes Ambientes

Assim como na Figura 7, temos na Figura 8 a elipse da esquerda representando a lógica do fluxo e a elipse da direita representando a lógica das aplicações. Percebe-se uma maior separação entre as aplicações e fluxo dos processos, porém ainda assim existem pontos em comum. Estes pontos em comum representam a dependência existente; pois, este ponto de união está representando para as aplicações o momento em que temos parte da lógica do fluxo sob sua responsabilidade. Este é o ponto de partida para este estudo.

Vamos analisar o que ocorre geralmente para cada um dos elementos que compõem os sistemas de *workflow*, sempre deixando clara a existência de variações de ferramentas para ferramentas, fazendo assim que existam diferentes níveis de dependência. Porém, para todas as ferramentas, por mais completa que seja a infra-estrutura que disponibilizem, geram em algum momento a dependência com as aplicações.

- **Atividades ou Estados:**

  Geralmente temos os fluxos modelados a partir das atividades ou tarefas a serem realizadas, porém também pode haver fluxos modelados a partir de estados. Tanto para atividades quanto para estados identificamos facilmente a dependência; pois, a aplicação terá "conhecimento" sobre o fluxo. Este "conhecimento", para a arquitetura, é necessário uma vez que para disponibilizar as informações ou ações para os usuários, a aplicação necessita identificar a situação que a instância se encontra no processo.

- **Ações:**

  As ações geralmente são disponibilizadas para os usuários em função de condições que acabam estando dispostas na própria aplicação, condições como: o processo encontrar-se em determinado estado, ou o usuário pertencer a determinado papel ou grupo, ou mesmo, um campo informado ser superior ou inferior a um determinado valor. Todo este código está no código da aplicação e não no processo modelado. O código pode, em alguns ambientes, estar modelado no fluxo, pela ferramenta CASE; porém, como fica a cargo da aplicação analisar os valores e as condições para disponibilizar ou não as possíveis ações. A aplicação acarreta este "ônus" de "conhecimento" como parte do seu código.

- **Papéis, Regras ou Grupos:**

  Ao ocorrerem modificações em papéis, regras ou grupos muito provavelmente haja necessidade de se modificar o código das aplicações igualmente. Para grande parte dos ambientes de *workflow*, as aplicações são responsáveis por verificar quais os usuários estão associados para determinado papel ou grupo, e com isso, se um papel for modificado, por exemplo, teremos que re-codificar a aplicação. Um exemplo pode ser: um código em que se tenha a exibição de uma ação para os usuários que exercessem o papel de "gerentes", porém agora não mais os "gerentes" são os responsáveis por aquela ação, mas sim o papel de "diretores".

- **Controle de Acesso:**

  Pouquíssimos são os ambientes de *workflow* que implementam este elemento, o controle de acesso pode ser dividido em vários níveis, o mais simples deles seria o controle de acesso para documentos, ou instâncias em execução. Podendo ser específico a ponto de ser utilizado a um nível de campos. Um exemplo seria, quando se realiza uma seleção sobre os processos em execução, o ambiente de *workflow* faz uma seleção somente dos processos aos quais o usuário, que está interagindo, pode visualizar. Porém este controle assim como os demais controles de acesso ficam a cargo da aplicação, portanto, este elemento é um dos que causam uma grande dependência entre o fluxo e as aplicações.

- **Códigos associados às ações:**

  Este elemento geralmente não causa muito problema com relação à dependência, pois o código, que está associado a uma ação, fica em partes separadas do código da aplicação, na maioria dos casos. Costumam ser criados arquivos específicos ou biblioteca que contém os códigos associados às ações, dependendo do ambiente que estejamos deverá ser re-compilado ou apenas modificado algum script, para o caso de uma linguagem interpretada.

Tendo definido alguns dos elementos, componentes de sistemas de *workflow*, e expondo as situações em que ocorre a dependência para cada um deles; conclui-se que o enfoque dos fabricantes de motores de *workflow* não está baseado na independência entre as aplicações e o motor. O papel do motor é ser o responsável pela modelagem (através de seu módulo de representação de fluxos) e gerência das instâncias criadas, assim como controlar o atendimento aos pedidos realizados pelas aplicações. O ambiente de *workflow* não tem características e funcionalidades que objetivem a independência entre os seus elementos componentes.

Assim sendo, o objetivo das ferramentas de mercado é atender todas as necessidades com relação à modelagem, gerenciamento e controle das ações e eventos referentes às instâncias; porém, não existe a preocupação de atender as aplicações com relação a uma independência no momento da utilização dos processos. Em função disso, no motor não consta, como parte da sua camada de API's, funções capazes de responder a requisições de informações referentes à modelagem do fluxo. As suas funções, disponibilizadas através de sua camada de API's, são capazes somente de retornar informações existentes sobre os processos, ou seja, para as instâncias, e realizar o controle das ações que podem ser executadas ou os eventos que devem ser disparados. Na camada de API's não estão presentes funções capazes de informar o que pode ser feito em cada momento com os processos em execução.

Como visto, são poucos os elementos que não causam uma dependência para as aplicações, cada um com suas características. Esta dependência será explorada com maiores detalhes, evidenciando seus problemas e conseqüências, com maior detalhe nos capítulos posteriores, onde será realizada uma análise aprofundada da dependência uma vez que ela já foi exposta.

5 Problemas gerados pela Dependência entre Processos e Aplicações

Após expor a dependência existente entre as aplicações e os processos modelados, podemos descrever com maiores detalhes os problemas e dificuldades que esta dependência causa às aplicações.

Os problemas existentes não são claramente percebidos durante o desenvolvimento da aplicação, ou mesmo durante a sua utilização, mas sim, no momento em que surgem modificações sobre os fluxos; pois devido à dependência, é gerado um grande esforço em função do re-trabalho que acaba sendo criado.

O desenvolvimento de sistemas em *workflow* reflete sobre uma série de comportamentos, por parte dos usuários finais, uma vez que está sendo lidado diretamente com a forma como é realizado o seu trabalho. Em função dessa proximidade existente, entre os sistemas e os seus usuários, uma série de reações pode ser verificada. Muitas destas reações e comportamentos refletem, diretamente sobre a dificuldade em se analisar os processos de forma correta, gerando assim vários problemas para o desenvolvimento, devido a análises mal realizadas.

Essa análise mal realizada acaba potencializando o problema do re-trabalho; pois gera a necessidade de um novo levantamento e posterior modificação de código, dentre os comportamentos e características existentes para o ambiente de sistemas em *workflow* podemos citar, segundo [MAT 2000]:

- O sistema de *workflow* pode substituir o trabalho antes feito pelas pessoas;
- As pessoas podem entender que o trabalho não será melhorado com o uso do sistema;
- A resistência dos usuários pode se manifestar como uma antipatia, passando para formas mais graves, como sonegação de informações, o boicote ao uso e a recusa em utilizar o processo reformulado;
- Erros na representação de trabalho no sistema de *workflow* ocorrem, principalmente pela falta de um projeto participativo;
- Os usuários desejam manter a flexibilidade individual do seu trabalho. Mesmo usando um sistema de *workflow*, querem gerenciar sua carga de trabalho, seus horários e a ordem em que vão executar as tarefas cotidianas. Tais comportamentos colocam em risco o bom andamento do sistema, uma vez que o processo tem que seguir a seqüência de ações pré-definidas;
- A tecnologia de sistemas de *workflow* pode reduzir a necessidade de convívio entre os funcionários, mudar as relações de poder, alterar os níveis de responsabilidades e, todas essas mudanças podem gerar certo desconforto às pessoas;

- O usuário acredita que seu trabalho pode ser desvalorizado, que perdeu a autonomia sobre a carga de trabalho e que deixou de personalizar a resolução de problemas cotidianos. Assim, é exigida do usuário uma reavaliação do seu trabalho e do seu próprio papel dentro do processo;

- O sistema de *workflow* pode controlar rigidamente o trabalho, quanto ao desempenho e a maneira de solucionar problemas;

- O sistema de *workflow* pode estar implantado em um fluxo de trabalho que não satisfaz as expectativas das pessoas.

Podemos agrupar os problemas existentes em duas categorias, que são: os problemas que existem durante o desenvolvimento das aplicações e os problemas existentes durante a manutenção dos fluxos. As próximas seções descrevem características específicas para cada umas das categorias:

5.1 Problemas existentes durante o desenvolvimento das aplicações

O problema existente durante o desenvolvimento das aplicações ocorre uma vez que, para controlar o fluxo dos processos, são utilizados *flags* (campos ou variáveis), de modo a definir as situações ou estados aos quais os documentos ou instâncias irão ser submetidos. Desta forma a lógica do fluxo fica baseada em determinados valores atribuídos a estes *flags* de controle. Isso é negativo, pois para o desenvolvimento da aplicação deve-se ter em mente qual a lógica utilizada pelo analista ou desenvolvedor no momento da definição destes *flags* e os valores a eles atribuídos.

Este problema pode ser contornado, ou ao menos minimizado, definindo-se padrões para a criação destes *flags*. Uma vez criados estes padrões podem ser utilizados por toda a empresa, facilitando assim a compreensão dos processos modelados e, da mesma forma, facilitando um esforço futuro originado em função de manutenções sobre os fluxos modelados.

Para exemplificar a utilização destes *flags*, a seguir temos duas tabelas, em cada tabela será utilizada uma lógica para representar o fluxo do processo. Neste caso iremos modelar um processo que representa a seguinte realidade: Para um processo hipotético, teríamos um total de cinco possíveis estados para os documentos, que seriam: Não Solicitado, Solicitado, Para Aprovação, Aprovados, Finalizados. Onde o encaminhamento dos documentos ou processos ocorre seqüencialmente, uma etapa a cada vez. Existe assim, uma seqüência de passos a serem realizados pelos documentos.

TABELA 1 - Exemplo de um *flag* único para o fluxo

Flag	**Valor**	**Estado do documento**
<tramitacao>	“1”	Não Solicitado
<tramitacao>	“2”	Solicitado
<tramitacao>	“3”	Para Aprovação
<tramitacao>	“4”	Aprovados
<tramitacao>	“5”	Finalizados

Na tabela 1 temos uma representação onde utilizamos apenas um *flag* (<tramitacao>) para representar o estado do documento, e seu valor indica o estado atual desta instância específica, para este caso todo o controle fica a cargo de somente um *flag*. Esta padronização tem suas vantagens e desvantagens, a maior vantagem é a de necessitarmos manipular, somente um *flag*, porém se faz necessário ter ciência de que valor representa cada um dos estados. Deve-se ter o conhecimento, no momento de elaborar a aplicação, de que o valor "1" representa a situação de "Não Solicitado", de que o valor "3" representa os documentos no estado de "Para Aprovação" e assim sucessivamente.

TABELA 2 - Exemplo de um conjunto de *flags* para o fluxo

Flag	**Valor**	**Estado do documento**
<nao_solicitado>	"1"	Não Solicitado
<solicitado>	"1"	Solicitado
<para_aprovacao>	"1"	Para Aprovação
<aprovado>	"1"	Aprovados
<finalizado>	"1"	Finalizados

Na tabela 2 temos uma representação onde utilizamos um conjunto de *flags* (<nao_solicitado>, <solicitado>, <para_aprovacao>, <aprovado>, <finalizado>) para representar o estado do documento. Para este caso a indicação do estado do documento se dá de acordo com o *flag* que estiver com valor de "1", ou seja, o *flag* "ativo" indica o estado ao qual o documento se encontra. Neste caso temos a vantagem de que o *flag* ativo representa diretamente o seu estado, a desvantagem está no momento em que se for realizar as condições para a manipulação dos documentos. Uma vez tendo-se vários *flags* para a manipulação do fluxo torna-se necessário implementar condições baseadas em um número maior de *flags*, para cada situação específica o valor dos *flags* é verificado, de acordo com o valor específico do *flag* que representa a situação.

5.2 Problemas existentes durante a manutenção dos fluxos

O problema existente na etapa de manutenção dos fluxos é o problema referente ao trabalho associado às modificações necessárias sobre os processos já realizados, juntamente com a repercussão sobre as aplicações neles baseadas.

A grande proposta da arquitetura baseada em um motor de *workflow* está na possibilidade de mantermos as aplicações nas plataformas e linguagens originais. Porém, a dependência existente causa um grande re-trabalho no momento das atualizações dos fluxos. Fazendo assim, com que esta proposta inicial perca um pouco do seu sentido original, pois apesar de estarmos trabalhando em ambientes ou linguagens distintas, estamos sempre atrelados a ambos os contextos (aplicação e *workflow*). Este re-trabalhado pode vir a ser potencializado, uma vez que o contexto de desenvolvimento pode ser um ambiente em que as aplicações estão sendo desenvolvidas em diversas linguagens de programação.

No momento de se efetuar as modificações sobre o processo, no seu módulo de especificação de fluxo não existem problemas, pois a modificação é realizada facilmente e muitas vezes de forma gráfica e, além disso, estamos somente em um contexto, o do *workflow*. Porém no momento em que iremos atualizar essas modificações sobre as aplicações, ocorre a necessidade de reescrever código. Assim sendo, é necessário pessoal especializado na linguagem em que foi implementada a aplicação, mobilizando mão-de-obra para atualizar o conhecimento das aplicações sobre o processo modelado. De acordo com a alteração realizada, pode-se um trabalho enorme, pois muitas vezes uma simples modificação sobre o caminho e ser seguido por um documento acarreta modificações em grande parte do código da aplicação.

Estas modificações sobre o código das aplicações podem ser minimizadas utilizando-se padrões para o seu desenvolvimento, porém não se conseguirá evitar o re-trabalho, uma vez que esta dependência está baseada na arquitetura dos motores de *workflow*. Desta forma as modificações acabam se tornando pouco eficientes em relação à produtividade.

Deve-se perceber que este problema não é apenas referente a uma falta de padronização da modelagem, ou alguma característica que poderia ser contornada, adequando a análise realizada sobre os sistemas no momento das definições e implementações. O problema é referente ao modo como a arquitetura dos sistemas está estruturada, acarretando assim, adicionar uma camada capaz de realizar uma expansão das funcionalidades do motor.

Isso pode não parecer muito negativo, porém quando se for analisar uma das principais vantagens desta arquitetura, a possibilidade de termos várias aplicações utilizando o mesmo fluxo, corresse o risco de comprometermos demais os sistemas de *workflow*, uma vez que uma modificação na modelagem do processo pode repercutir em várias aplicações. Devemos lembrar que estas aplicações podem ser desenvolvidas em diferentes linguagens de programação ou mesmo diferentes contextos e ambientes. Com isso podem ser comprometidas diferentes equipes de desenvolvimento.

Outra característica importante, exposta anteriormente, é a dinamicidade com que os sistemas baseados em *workflow* são desenvolvidos, pois são sistemas que exigem constantes modificações e atualizações, por implementarem fluxos dinâmicos. Essa característica agrava ainda mais a dependência entre o processo modelado e a aplicação que dele se servirá, pois torna necessário controlar este “sincronismo” entre o que está modelado e o que é compreendido pela aplicação.

Outro ponto negativo para a dependência, se dá em relação ao conceito de reuso dos processos modelados. Podemos realizar a modelagem dos processos de forma genérica, de modo a possibilitar a reutilização de processos comuns a mais de um fluxo existente. Isso é de extrema importância, porém pode ser igualmente arriscado do ponto de vista do re-trabalho que possa acarretar, caso um fluxo genérico necessite modificações. No diagrama da Figura 9, é proposto um modelo exemplificando um caso em este contexto é encontrado.

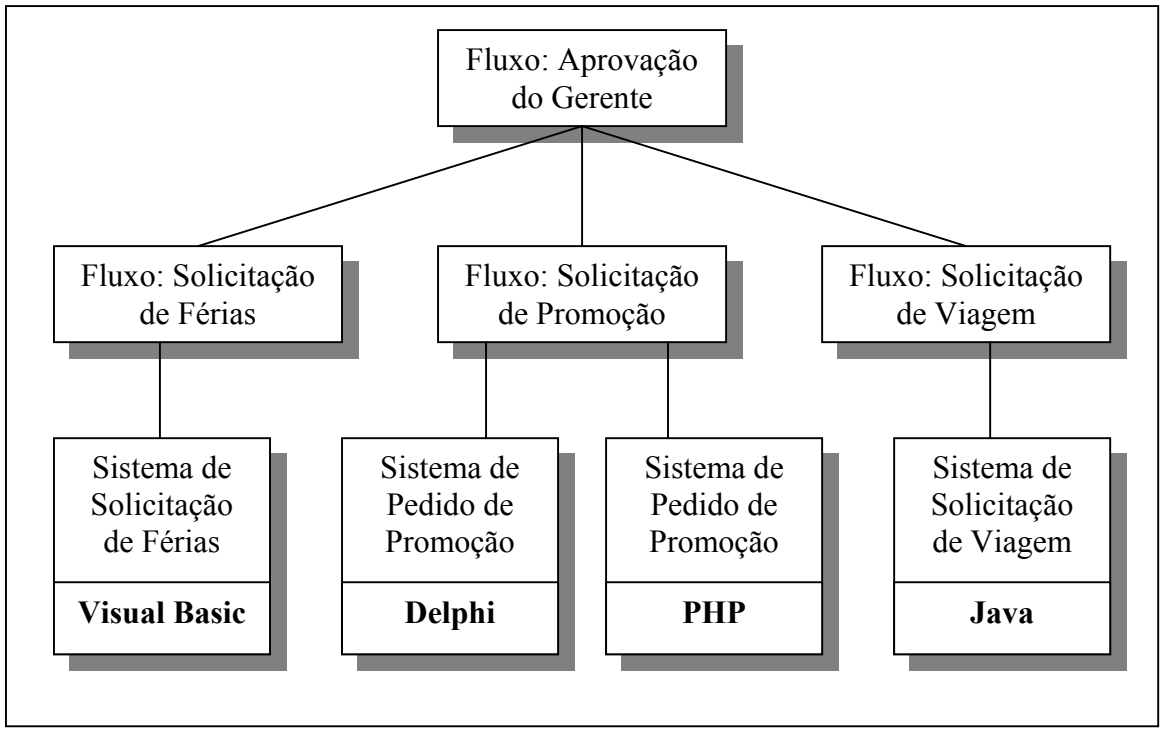


FIGURA 9 - Exemplo de uma reutilização de processos

Na Figura 9 temos a representação de uma modelagem de fluxos em que foram utilizados os conceitos de reaproveitamento, baseando a análise no mesmo paradigma da programação orientada a objetos. Temos um fluxo genérico (Aprovação do Gerente) que é utilizado por outros três (Solicitação de Férias, Solicitação de Promoção e Solicitação de Viagem). Para estes três fluxos existe uma etapa de aprovação da gerência, dessa forma, ao invés de repetir o fluxo já modelado, está sendo utilizado o processo modelado previamente para os demais processos. Nota-se também as aplicações que se baseiam nestes três fluxos, cada qual com sua respectiva linguagem de programação.

Caso ocorra alguma modificação do fluxo de Aprovação do Gerente, isso irá repercutir diretamente em cada uma das quatro aplicações existentes (Sistema de Solicitação de Férias, Sistema de Solicitação de Promoção – Cliente/Servidor e Web e Sistema de Solicitação de Viagem). Como, neste exemplo, cada sistema foi implementado em uma linguagem específica; respectivamente: Visual Basic, Delphi, PHP e Java; será necessário atualizar o código para estas quatro linguagens. Este exemplo demonstra como uma simples modificação em apenas um fluxo pode repercutir em modificações em vários outros contextos.

6 Proposta de Solução para o Problema da Dependência

Neste capítulo será apresentada a proposta, da qual está baseado todo o estudo, para a Camada Intermediária. São apresentados os objetivos, características e estruturas que compõem esta camada. A Camada Intermediária pode ser definida como uma forma de expansão do motor de *workflow*, de forma a comportar a independência necessária entre aplicações e processos.

A Camada Intermediária será analisada separadamente sob a divisão de Camada de Independência e Camada de Integração. Juntamente com a análise sobre cada uma das camadas, serão abordados aspectos do protótipo desenvolvido e do motor utilizado no contexto de *workflow* utilizado.

6.1 Introdução

Como incentivo para o desenvolvimento de uma solução para esta dependência, entre aplicações e fluxo, temos todos os argumentos expostos no capítulo anterior, visando um novo enfoque para uma arquitetura que agregue facilidades ao desenvolvimento de aplicações baseadas em *workflow*. Esta arquitetura pode ser utilizada de melhor forma, no momento em que obtivermos a clara separação entre os processos e as aplicações.

Enfatizando uma crítica à arquitetura de *workflow* baseada em ambientes distintos para modelagem e execução dos fluxos, podemos abordar uma característica importante: a utilização de diferentes linguagens de programação. Esta característica causa uma facilidade na programação, agregando grande flexibilidade ao desenvolvimento das soluções. Porém, esta característica, que nos possibilita não programar em um ambiente específico de *workflow*, acaba se tornando aos poucos um ponto negativo em função da dependência existente. Essa dependência agrega um grande trabalho com modificações, que deverá ser realizado em diferentes ambientes.

Com esta arquitetura, as aplicações não são desenvolvidas em um ambiente específico de *workflow*, muitas vezes proprietário, porém as aplicações permanecem com um forte vínculo com o ambiente de *workflow*. Este vínculo acaba por gerar esforço que pode chegar a ser comparado com as aplicações em que temos uma arquitetura única de desenvolvimento e execução. No momento em que isso ocorre, a arquitetura de ambientes distintos acaba por se tornar sub-utilizada; pois estará agregando apenas a vantagem de termos um modelo de forma gráfica para os fluxos modelados, porém o código continuará complexo e confuso no que diz respeito a futuras modificações.

6.2 Objetivos

Basicamente o objetivo desta proposta é eliminar totalmente a dependência existente entre as aplicações, independente da linguagem utilizada, e o fluxo ao qual ela se referencie, modelado em um ambiente de *workflow*. De forma que, qualquer

modificação realizada sobre o fluxo não repercuta, de forma alguma, em modificações sobre o código das aplicações.

Esta independência resultará, por parte da equipe de desenvolvimento, em uma padronização das aplicações, pois ao utilizar os processos, através das funções disponibilizadas pela camada de API's da Camada Intermediária, existirá uma seqüência de passos a serem seguidos que sempre serão repetidos. O resultado deste padrão será, uma forma simples e clara de utilizar os processos através das funções disponibilizadas por esta camada. As aplicações servirão apenas como uma interface, quando se estiver referenciando os processos nelas existentes.

Na Figura 10 temos um diagrama representando a lógica das regras de negócio para das aplicações e as regras para a lógica dos fluxos. O diagrama está baseado na proposta para a separação entre as duas lógicas envolvidas nos sistemas.

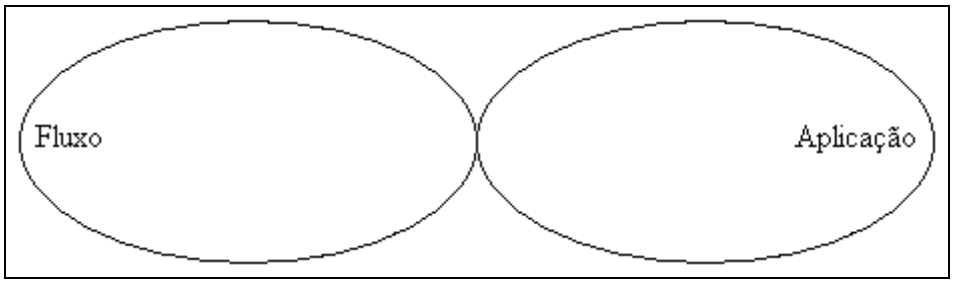


FIGURA 10 - Diagrama da arquitetura baseada na proposta para independência

Na Figura 10 vemos que não existem pontos em comum para qualquer uma das elipses, ou seja, ficam totalmente separadas: a lógica dos fluxos modelados, representada pela elipse da esquerda e a lógica das aplicações, representada pela elipse da direita. Se compararmos este diagrama com os diagramas das Figuras 7 e 8, percebe-se que o enfoque principal está na separação total das regras e controles existentes, uma vez que não existem pontos de união entre as elipses.

Uma vez eliminada a dependência existente, as aplicações terão a responsabilidade de controlar apenas os controles referentes a sua própria lógica do negócio, não tendo assim, qualquer envolvimento com etapas do fluxo.

6.3 Características

A proposta está baseada no desenvolvimento de uma Camada Intermediária entre as aplicações e o motor de *workflow* (sua camada de API's). O papel desta camada é o de atender as aplicações sobre todas as requisições referentes ao processo e as instâncias existentes, e se necessário, repassá-las ao motor de *workflow*, através das API's do motor, e após retornar a requisição às aplicações.

Em momento algum a Camada Intermediária será responsável pelo controle da execução dos processos ou documentos. Este controle permanece sendo do motor de *workflow*, a Camada Intermediária pode ser vista como uma extensão para o motor de

workflow visando complementar a sua estrutura, de modo a comportar a lógica do fluxo dos processos.

No diagrama da Figura 11 está ilustrado do relacionamento entre a Camada Intermediária e o Motor de *Workflow*, temos a camada, com suas estruturas e elementos, interagindo com o motor como um ambiente completo.

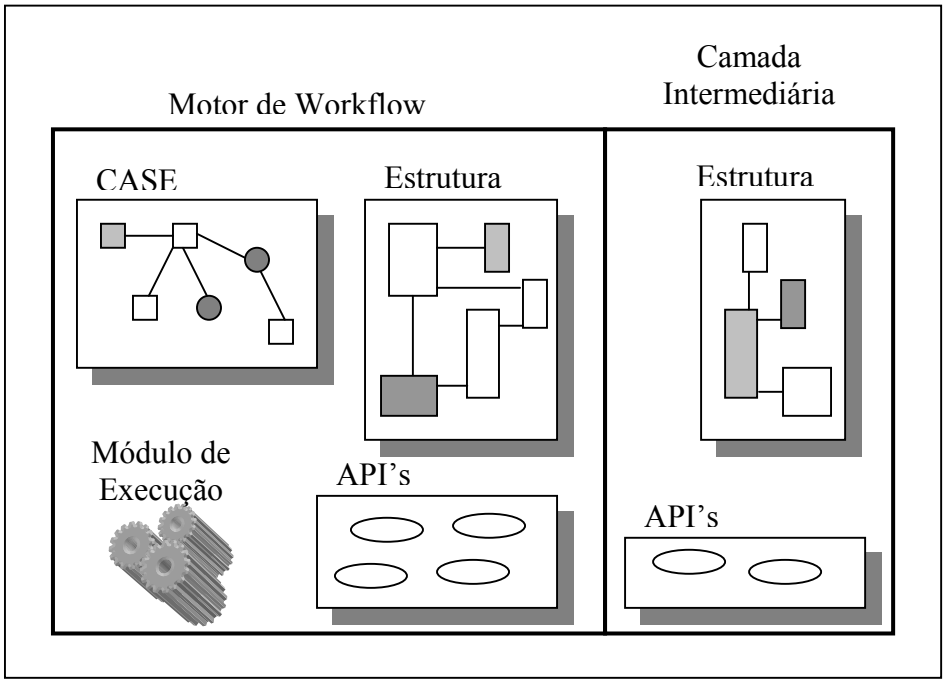


FIGURA 11 - Camada Intermediária com o Motor de *Workflow*

Pode-se perceber, conforme e Figura 11, que ocorre uma expansão das estruturas e funcionalidade (API's) do motor de *workflow*. Não há, em momento algum, uma concorrência entre os papéis exercidos pelo Motor e pela Camada Intermediária. A execução e gerência dos processos estão sempre controladas pelo motor; pois como ilustrado, não existe nenhum módulo de execução da Camada Intermediária. Na Camada Intermediária estão somente estruturas gerando subsídios para a complementação das informações necessárias para o modelo apresentado.

6.4 Estrutura

A proposta para o problema da falta de independência está na utilização de uma camada que realiza a expansão da ferramenta de modelagem dos fluxos e do próprio motor. Com esta expansão seriam considerados os aspectos necessários para a total independência entre motor e aplicação. Seriam adicionadas funcionalidades tanto na ferramenta de modelagem quanto nas funções disponibilizadas pelo motor.

Para o motor, referenciando a sua camada de API's, seriam adicionadas funções para o retorno das especificações da modelagem realizada. O papel destas funções é justamente evitar que o "conhecimento" da lógica do fluxo esteja no código das aplicações, e que este "conhecimento" seja informado pelo motor às aplicações. Algumas das funções existentes nesta camada que podem ser citadas seriam:

- Retorno de todas as possíveis situações (estados) do processo: esta função retorna todos os estados existentes, uma vez que para a aplicação os estados do fluxo não são de seu conhecimento, é necessário que sejam informados; após isso, podem ser informados aos usuários a fim de que eles selecionem o(s) desejado(s);

- Retorno de todas as instâncias pendentes de um usuário (documentos sob sua responsabilidade): Esta função retorna todos os documentos que estão aguardando alguma ação do usuário para terem seu prosseguimento. Com esta função pode-se facilmente listar todos os documentos que estão pendentes para o usuário, facilitando assim as consultas às bases de documentos;

- Retorno de todas as instâncias possíveis de visualização por determinado usuário: para o usuário visualizar todos os documentos existentes para determinada situação, deve ter sido anteriormente selecionada qual situação deseja consultar, por exemplo: visualizar todos os documentos de Normas que estão Aprovados e que podem ser visualizados por todos os funcionários;

- Retorno das possíveis ações de um determinado usuário para uma determinada instância: situação em que o usuário selecionar um documento específico. E, a partir deste ponto, a aplicação deve ser informada das ações que podem ser tomadas sobre o mesmo, exibindo assim através de botões ou link's as ações que o usuário tem acesso de executar;

- Retorno do controle de acesso (tanto leitura, quanto edição) do usuário para uma específica instância: no momento em que o usuário for alterar algum dado referente ao processo, será informado à aplicação se estes podem ser alterados, somente lidos, ou se o usuário não tem acesso algum sobre os dados.

Esta lista de funções representa somente uma parte das informações que seriam retornadas pelas API's da Camada Intermediária. O objetivo destas funções é oferecer meios para que a aplicação requisite as informações sobre a estrutura do fluxo que está trabalhando.

Para a ferramenta de modelagem são adicionadas, para todos os elementos do fluxo (atividades, estados, regras, papéis, condições, e demais elementos), as informações necessárias, de modo a ter-se toda a informação necessária ao fluxo neste modelo. Deste modo, não será necessário qualquer outro complemento ou trecho de código para representar qualquer informação sobre qualquer dos elementos constituintes do fluxo. Estes complementos estão representados nas próprias aplicações do modo como hoje são estruturadas as soluções de *workflow*.

Na Figura 12 temos uma idéia de como está enquadrada esta Camada Intermediária na arquitetura do motor. O diagrama representa a Camada Intermediária como um elemento único, posteriormente será realizada uma análise aprofundada especificamente sobre a Camada Intermediária.

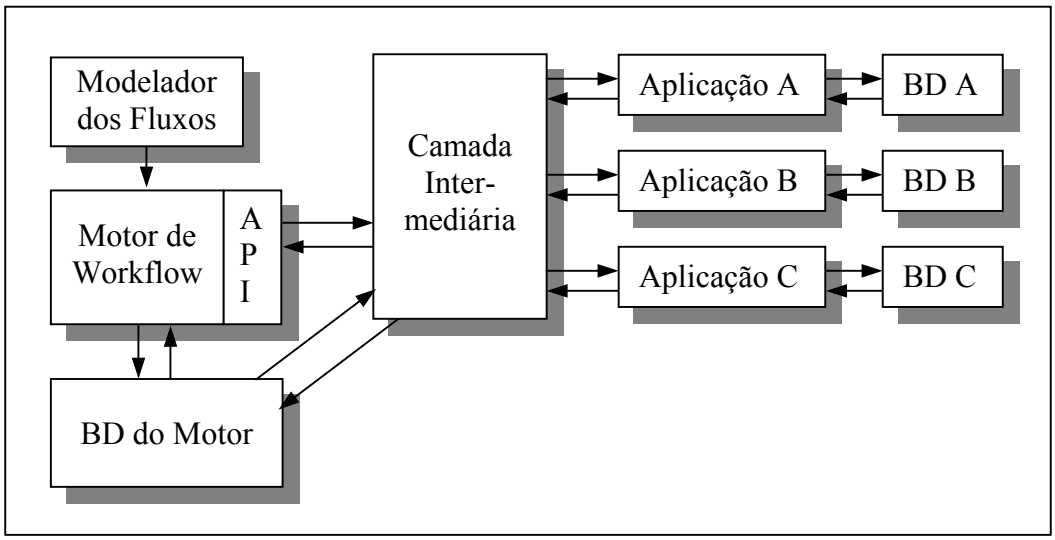


FIGURA 12 - Arquitetura da Camada Intermediária

O diagrama na Figura 12 está representando o modo como é estruturada a arquitetura desta Camada Intermediária. É importante destacar que existe, nesta Camada Intermediária, comunicação além de com a aplicação, também com o motor de *workflow* (através das API's). A partir do diagrama torna-se mais clara a visualização da comunicação existente entre cada um dos elementos, juntamente com as aplicações. Vale lembrar que o banco de dados das aplicações (representados por BD A, BD B e BD C) não se comunica com nenhum outro elemento, além de sua própria aplicação.

Os banco de dados: das aplicações, da Camada Intermediária e do motor de *workflow* não têm necessidade de estarem em um mesmo contexto ou ambiente, sua utilização é totalmente desvinculada de qualquer dependência. Cada banco de dados é responsável por um papel específico: o banco de dados das aplicações fica responsável pelo armazenamento dos dados referentes às regras da aplicação e o banco de dados do motor fica encarregado de controlar as instâncias dos processos.

Fazendo uma análise detalhada desta Camada Intermediária, podemos dividí-la em três partes: um Banco de Dados da camada, uma Camada de Integração e uma Camada de Independência. Esta separação é feita com o objetivo de delimitar o escopo e separar as suas funções, pois poderiam ser definidas como um módulo único.

Cada uma dessas novas camadas fica responsável por uma etapa no processo de atendimento às requisições feitas pelas aplicações, sempre baseada no primeiro elemento, o banco de dados da Camada Intermediária. Assim temos uma forma de separar os procedimentos e ter, de forma mais organizada e modular, as estruturas responsáveis pelo retorno das informações solicitadas. Analisando de forma mais específica: a Camada de Integração, a Camada de Independência e o Banco de Dados responsável pela armazenagem do fluxo; temos como demonstrado na Figura 13, dentro da arquitetura do motor, a seguinte representação de suas estruturas.

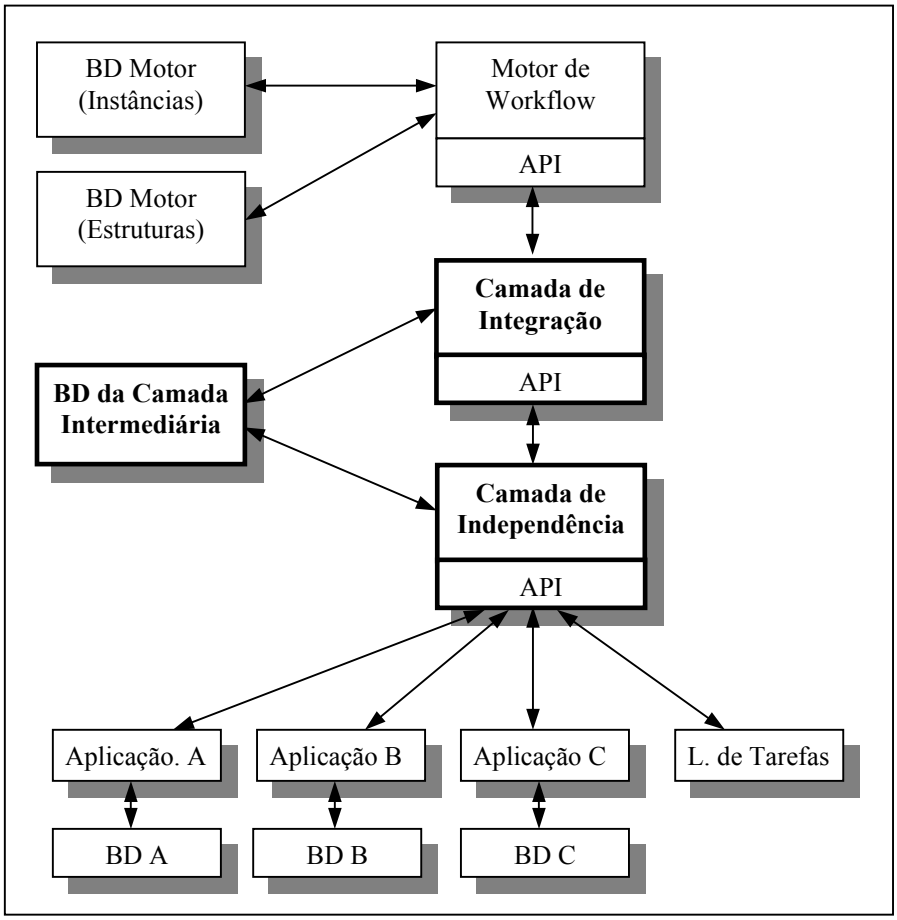


FIGURA 13 - Camada Intermediária e Camada de Integração

Como mostrado na Figura 13, os três elementos (Camada de Integração, Camada de Independência e BD da Camada Intermediária) estão relacionados entre si, e interagindo com os demais. Temos assim: o BD da Camada Intermediária como a fonte de informações sobre as estruturas dos fluxos e instâncias existentes e, as duas camadas implementando uma interface para a manipulação e recuperação destas informações.

Deve ser dito, ainda para a Figura 13, que todas as aplicações, com seus bancos de dados (BD A, BD B, BD C), armazenam as informações específicas sobre a lógica do negócio. Para a lista de tarefas não existe um banco de dados; pois a lista de tarefas trata-se de uma aplicação com o objetivo de informar aos usuários sobre suas tarefas pendentes, não necessitando assim um banco de dados próprio, uma vez que não existe lógica de negócio para realizar esta função, ela se servirá somente das informações disponibilizadas pelo motor de *workflow*, através de suas API's.

No banco de dados da Camada Intermediária está o núcleo da proposta, pois será criado um banco de dados baseado no modelo relacional, onde está armazenada toda a lógica dos fluxos. Este banco de dados existe internamente na arquitetura atual dos ambientes de *workflow*, porém, nestes ambientes, sua estrutura não contempla a totalidade dos processos modelados, e em alguns casos, nem mesmo este é seu objetivo.

A Camada de Independência será a porta à qual as aplicações solicitarão e receberão todas as informações sobre as instâncias e os fluxos. O papel da Camada de Independência é o de atender os pedidos referentes às estruturas dos fluxos, retornando as ações possíveis a serem realizadas, a seqüência dos estados e tudo que for relativo ao fluxo. Todas as solicitações, referentes às instâncias ou processos existentes serão repassadas para a Camada de Integração; pois estas informações dizem respeito à execução dos processos e não à estrutura dos fluxos.

Na Camada de Integração serão resolvidas as solicitações referentes às instâncias dos processos, neste ponto existe a integração entre o modelo existente no banco de dados da Camada Intermediária e o banco de dados do ambiente de *workflow*. O papel da Camada de Integração é realizar uma conversação entre as duas partes: a Camada de Independência e as API's do motor. Para atender estas solicitações serão chamadas as funções disponibilizas pelo ambiente de *workflow*, através da camada de API's, para a manipulação das instâncias. Por exemplo, a execução de uma ação é requisitada pela aplicação; ela é interpretada pela Camada de Integração e invocada, pelas API's do motor, a função referente a este pedido, sendo executada a ação. A Camada de Integração não executa as ações apenas as dispara no ambiente do *workflow*. Nas próximas seções a Camada Intermediária será analisada detalhadamente.

6.5 Banco de Dados da Camada intermediária

Ao ser analisado o banco de dados da Camada Intermediária, deve-se estar ciente que, nele está estruturada toda a lógica modelada sobre o fluxo. Motivo pelo qual, pode-se denominá-lo o núcleo da proposta de independência. Este é o escopo e o foco principal da Camada Intermediária, todas as suas funções e estruturas estão calcadas na idéia de gerar subsídios para que, a lógica dos processos, esteja desvinculada da aplicação e armazenada em seu banco.

Será alcançada a independência necessária, pois as aplicações irão solicitar, através da Camada de Independência, todo o conhecimento necessário para a manipulação dos documentos ou processos. Assim, as aplicações tornam-se totalmente isentas de "responsabilidade" pelo fluxo do processo, pois esta tarefa é repassada ao banco de dados.

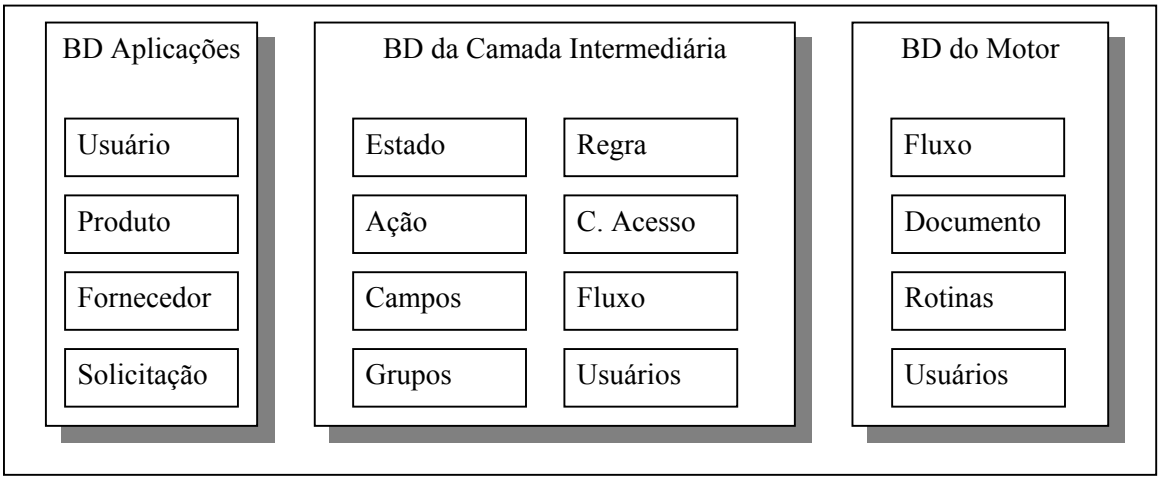


FIGURA 14 - Representação dos bancos de dados

Podemos perceber na Figura 14 a separação existente entre cada uma das estruturas existentes no banco de dados: das aplicações, da Camada Intermediária e do motor de *workflow*. Cada um destes bancos de dados tem um papel específico, e de acordo com suas estruturas de armazenamento, podemos perceber qual o tipo de informação será armazenada em cada um deles.

Para as aplicações, as informações necessárias dizem respeito única e exclusivamente às suas regras de negócios, temos tabelas como: Usuário, Produto, Fornecedor; ou seja, tabelas que contém os elementos que estarão sendo manipulados pela aplicação, independente do fluxo utilizado.

O banco de dados do motor terá as estruturas para armazenar as situações, os valores, as ações, as permissões; enfim, as informações referentes a cada umas das instâncias existentes. O armazenamento e controle sobre os processos em execução permanecem de posse do motor de *workflow*, não ocorre qualquer interferência por parte da Camada de Independência ou da Camada de Integração sobre o banco de dados do motor.

E por fim, no banco de dados da Camada Intermediária, as estruturas montadas têm como principal função representar o fluxo do processo, ou seja, todos os elementos que constituem um processo estão modelados neste banco de dados. Tendo os relacionamentos entre tabelas responsáveis pelas ligações existentes entre cada um destes elementos (Ações, Papéis, Grupos,...).

Podemos, assim como foi feito para a Camada Intermediária, dividir a estrutura do banco de dados, em duas partes: uma responsável por armazenar os dados referentes à Camada de Independência e outra responsável pelos dados a serem utilizados pela Camada de Integração. Assim sendo, temos duas estruturas de banco de dados, uma para cada camada, e cada uma com suas funções específicas.

O objetivo do banco de dados para a Camada de Independência é o de armazenar a lógica do processo modelado, ou seja, o conjunto de tabelas representando a modelagem de *workflow*.

Para o banco de dados da Camada de Integração temos um conjunto reduzido de estruturas, pois o objetivo destas tabelas não será o de armazenar toda a lógica de um sistema de *workflow*, mas sim realizar o mapeamento das estruturas existentes na Camada de Independência e o motor de *workflow*. Esse mapeamento se dá justamente pelo papel de um interpretador entre o motor e a Camada de Independência.

A seguir, será exposta a estrutura do banco de dados, para ambas as camadas, independência e integração, com seus detalhes e estruturas.

6.5.1 Banco de Dados da Camada de Independência

O banco de dados da Camada de Independência foi composto levando-se em consideração alguns aspectos básicos, que auxiliaram na definição do escopo a que a estrutura criada irá se propor a atuar. O objetivo principal foi o de permitir a definição dos fluxos de forma simples, após o início da elaboração das estruturas (tabelas e

relacionamentos), algumas características a mais foram modeladas, tais como: controle de acesso, definição de grupos, papéis, e usuários.

O modelo completo tornou-se extenso demais para ser analisado globalmente, em função disso serão analisadas separadamente cada uma das tabelas existentes para o modelo, e partes da estrutura completa, de acordo com o foco da análise.

Apesar de disponibilizar uma estrutura completa e complexa, para o armazenamento dos dados, podemos utilizar apenas parte desta estrutura para a implementação da solução da Camada Intermediária. Tornando-se assim, a estrutura flexível a ponto de não ser necessário exigir a utilização de todas as suas funcionalidades.

Será realizada uma análise, para o modelo gerado, sobre cada uma das tabelas implementadas e posteriormente os principais relacionamentos existentes entre as mesmas. Inicialmente são expostas as tabelas:

As estruturas das tabelas, representadas nas Figuras 15 a 23, foram extraídas utilizando-se a ferramenta visual do banco de dados SQL Server.

- **Ação**:

FIGURA 15 - Tabela de Ação da Camada de Independência

O objetivo da tabela de Ação é o de armazenar todos os dados referentes às ações que podem ser realizadas sobre o fluxo modelado. Como podemos ver pela Figura 15, temos as seguintes informações armazenadas:

- Descrição: atributo onde é descrito o nome da ação, é a informação disponibilizada aos usuários no momento em que é disposta a ação;
- Condição: atributo onde é descrita uma condição para que a ação seja permitida ou não, esta condição diz respeito a alguma informação específica da instância em questão, não é uma condição genérica para todos os documentos. Um exemplo pode ser: um valor existente para um campo referente ao fluxo, como o valor de um aprovador, em que a ação só deve ser disponibilizada para o aprovador informado em um determinado campo. Para o protótipo implementado esta funcionalidade não foi utilizada;
- Arquivo: atributo em que é descrito o caminho e o arquivo que deve ser executado quando a ação é realizada. Esta informação a princípio não é de responsabilidade da Camada de Independência; pois fica a cargo do

motor de *workflow* realizar a ação, porém foi criado este atributo visando uma funcionalidade futura, em que possam ser expandidas as ações realizadas pela Camada de Independência;
- Comentário: atributo onde são descritas informações a um de nível de ajuda para o usuário.

Os demais atributos têm como função servir de chaves para os relacionamentos ou para a própria tabela de Ação.

- **Acesso**:

FIGURA 16 - Tabela de Acesso da Camada de Independência

A tabela de Acesso é uma tabela que serve de base para todos os acessos existentes no modelo. Seus registros contêm valores para informar o acesso tanto para o controle de permissões sobre o fluxo, quanto para a execução das ações e mesmo para a visualização ou edição dos campos existentes no modelo. Como pode ser visto pela Figura 16, além do atributo de Tipo, temos, o atributo CodAcesso, que serve como chave para a tabela:

- Tipo: atributo onde serão descritas informações do tipo que está sendo armazenado. Serão armazenadas apenas descrições sobre o acesso existente, as aplicações serão responsáveis por interpretar a informação obtida desta tabela. Em função desta característica é importante existir um padrão para definir o acesso, como por exemplo: Consultar, Editar, Excluir, Criar e demais controles que se necessite exercer sobre os objetos do fluxo.

- **Campo**:

FIGURA 17 - Tabela de Campo da Camada de Independência

A tabela de Campo é uma tabela que serve de base para todos os campos existentes no modelo. Seus registros contêm valores para identificar cada um dos campos que compõem a estrutura do fluxo. Os campos armazenados são os campos que contém informações relevantes para o fluxo. Estes campos não são referentes à regra do negócio, mas referentes à regra do processo que é modelado. Como exposto na Figura 17, temos as seguintes informações armazenadas:

- Nome: atributo onde estão descritas informações sobre o nome que será atribuído ao campo, ou seja, o modo como ele será representado;
- Obrigatório: atributo onde é informado para a aplicação se o preenchimento deste campo se faz de forma obrigatória ou opcional, este atributo aceita somente dois valores: verdadeiro ou falso, sendo verdadeiro tornando obrigatório o preenchimento e falso opcional;
- Editável: atributo onde será informado se o valor do campo é preenchido pelo usuário (sendo possível de edição) ou somente de leitura, sendo preenchido de forma automática baseado em alguma condição, a cargo da aplicação;
- Visível: atributo informa se o campo é visível para os usuários, ou trata-se de um campo destinado a controles internos para o fluxo;
- Valor Default: atributo informa qual é o valor padrão para este campo, caso exista algum valor pré-definido para o seu preenchimento;
- Comentário: atributo onde são descritas informações em um de nível de ajuda para o usuário. Alguma orientação como o tipo de informação que deve ser preenchida;
- Label: atributo onde é informado o texto a ser exibido pelas aplicações quando o campo é exibido aos usuários, diferentemente do seu nome, o label serve somente para a interface das aplicações.

Os demais atributos têm como função servir de chaves para os relacionamentos ou para a própria tabela de Campo.

- **Estado**:

Estado

Nome da coluna	Tipo condensado	Permitir nulos
CodEstado	int	☐
CodFluxo	int	☑
Descricao	nvarchar(100)	☑
TempoLimite	int	☑
CodAcaoLimite	int	☑
TempoAviso	int	☑
MsgAviso	nvarchar(100)	☑
Comentario	nvarchar(100)	☑
		☐

FIGURA 18 - Tabela de Estado da Camada de Independência

A tabela de Estado é uma tabela que serve para armazenar todos os estados existentes no fluxo, todas as etapas às quais os documentos ou processos

poderão percorrer estarão descritas nesta tabela. Como exposto na Figura 18, temos as seguintes informações armazenadas:

- Descrição: atributo onde é descrito o nome do estado, esta é a informação disponibilizada aos usuários no momento em que é exibido o estado dos documentos;
- Tempo Limite: atributo onde é informado um tempo limite para a permanência neste estado, ao alcançar este tempo limite o documento deve executar a ação limite correspondente. Esta informação não é de responsabilidade da Camada de Independência; pois fica a cargo do motor de *workflow* realizar este controle, porém foi criado este atributo visando uma funcionalidade futura em que venham a ser expandidas as ações realizadas pela Camada de Independência;
- Tempo Aviso: atributo onde é informado um tempo limite para que seja enviada uma mensagem ao usuário responsável pela ação que deve ser tomada para este estado. Este aviso acontece anteriormente à expiração do prazo limite de permanência do documento no estado, assim como o tempo limite este controle fica a cargo do motor de *workflow*;
- Mensagem Aviso: atributo onde é informado o aviso que é enviado ao usuário, quando o tempo limite para o envio da mensagem for alcançado. Assim como os dois atributos anteriores, este controle não foi implementado, pois é parte das tarefas realizadas pelo motor de *workflow*, podendo futuramente ser ampliadas as funções da Camada de Independência, e assim exercendo também esta tarefa;
- Comentário: atributo onde são descritas informações em um de nível de ajuda para o usuário, informações referentes a características deste estado ou orientação gerais aos usuários sobre o estado que o documento se encontra.

Os demais atributos têm como função servir de chaves para os relacionamentos ou para a própria tabela de Estado, representando não somente os relacionamentos, mas também as próximas ações a serem executadas, como é o caso do atributo CodAcaoLimite.

- **Fluxo**:

Fluxo

	Nome da coluna	Tipo condensado	Permitir nulos
🔑	CodFluxo	int	☐
	Descricao	nvarchar(50)	☑
	Sigla	nvarchar(50)	☑
	Comentario	nvarchar(50)	☑
			☐

FIGURA 19 - Tabela de Fluxo da Camada de Independência

A tabela de Fluxo é uma tabela que serve como chave para agregar todos os demais elementos que formam, em conjunto, a estrutura completa do processo. Existirá um controle de acesso também relacionado a este

elemento. Como exposto na Figura 19, temos as seguintes informações armazenadas:

- Descrição: atributo onde é informado o nome do fluxo, o modo como é referenciado pelas aplicações que deve se servirão. A descrição é um atributo com o objetivo mais informativo do que funcional;
- Sigla: atributo onde é informada a sigla do processo que está sendo modelado. Assim como o atributo de descrição seu objetivo é mais informativo do que funcional;
- Comentário: atributo onde são descritas informações em um de nível de ajuda para o usuário, informações referentes a características deste fluxo, o que ele se propõe ou mesmo informações complementares sobre sua utilização ou algum procedimento a ser seguido.

Os demais atributos têm como função servir de chaves para os relacionamentos ou para a própria tabela de Fluxo.

- **Grupo**:

FIGURA 20 - Tabela de Grupo da Camada de Independência

A tabela de Grupo é uma tabela que tem como função o armazenamento de todos os grupos existentes para os processos e mesmo para a organização. Podem existir outras formas de representação de grupos para a organização, como por exemplo, a utilização do padrão LDAP, porém é necessário que seja de conhecimento do motor a estrutura de grupos e usuários associados aos mesmos. Foi implementado um nível de recursão sobre os grupos, de forma que possamos utilizar grupos pertencentes a outros grupos, porém somente em um nível. Como exposto na Figura 20, temos as seguintes informações armazenadas:

- Descrição: atributo onde é informado o nome do grupo, o modo como é referenciado pelas aplicações que dele se servem. Ao associarmos um usuário ou mesmo outro grupo a este grupo, esta é a informação disponibilizada aos usuários para este procedimento;

- Comentário: atributo onde são descritas informações em um de nível de ajuda para o usuário, informações referentes a características deste grupo, o tipo de pessoas associadas ou mesmo alguma informação específica para o grupo.

Os demais atributos têm como função servir de chaves para os relacionamentos ou para a própria tabela de Grupo. Os componentes do

grupo estão descritos a partir dos relacionamentos existentes, mas estes relacionamentos são vistos mais adiante.

- **Papel**:

FIGURA 21 - Tabela de Papel da Camada de Independência

A tabela de Papel é a tabela que tem como função o armazenamento de todos os papéis existentes para os processos. Estes papéis, ou regras, representam o modo como são relacionados os usuários e grupos com suas atribuições para o processo modelado. Como exposto na Figura 21, temos as seguintes informações armazenadas:

- Descrição: atributo onde é informado o nome do papel, o modo como é referenciado pelas aplicações que dele se servem. Ao associarmos um usuário ou mesmo um grupo a este papel, esta é a informação disponibilizada aos usuários para este procedimento;
- Comentário: atributo onde são descritas informações em um de nível de ajuda para o usuário, informações referentes a características deste papel, o tipo de pessoas ou grupos associados ou mesmo alguma informação específica para o papel.

Os demais atributos têm como função servir de chaves para os relacionamentos ou para a própria tabela de Papel. Os componentes do papel estão descritos a partir dos relacionamentos existentes, mas estes relacionamentos são vistos mais adiante.

- **Tipo**:

FIGURA 22 - Tabela de Tipo da Camada de Independência

A tabela de Tipo é a tabela que tem como função o armazenamento de todos os tipos permitidos para os campos existentes. Portanto a tabela de Tipo está relacionada à tabela de Campo, sua função é, para etapa de implementação, o controle de valores coerentes com os tipos de dados armazenados, que é realizado pelas aplicações. Como exposto na Figura 22, temos a seguinte informação armazenada:

- Tipo: atributo onde é informado o nome do tipo de dados. Deve ser especificado, juntamente com as aplicações, um padrão para a definição dos tipos, um exemplo: String, Inteiro, Real, Data, Moeda, e outros que venham a ser necessários.

Os demais atributos têm como função servir de chaves para os relacionamentos ou para a própria tabela de Tipo.

- **Usuário**:

Usuario

	Nome da coluna	Tipo condensado	Permitir nulos
🔑	CodUsuario	int	☐
	Nome	nvarchar(50)	☑
	Matricula	nvarchar(50)	☑
	Sexo	bit	☐
	Login	nvarchar(50)	☑
	Senha	nvarchar(50)	☑
			☐

FIGURA 23 - Tabela de Usuário da Camada de Independência

A tabela de Usuário é a tabela que tem como função o armazenamento de todos os usuários que podem interagir com o processo modelado. Para a tabela de usuários foi implementada uma tabela com um número limitado de informações, uma vez que o objetivo não era o de ter uma modelagem completa de cada um dos usuários, mas apenas dados básicos sobre eles. Possivelmente a organização já contenha os dados completos dos usuários em alguma outra estrutura de armazenamento. Como exposto na Figura 23, temos a seguinte informação armazenada:

- Nome: atributo onde será informado o nome do usuário;
- Matrícula: atributo onde será informada a matrícula do usuário;
- Sexo: atributo onde será informado o sexo do usuário, caso verdadeiro será considerado masculino, caso falso será considerado feminino;
- *Login*: atributo onde será informado o *login* do usuário, será o modo como ele irá se identificar para o processo que for trabalhar;
- Senha: atributo onde será informada a senha do usuário, associada ao seu *login*.

Os demais atributos têm como função servir de chaves para os relacionamentos ou para a própria tabela de Usuário.

Com esta definição mostrada acima, foram apresentadas as tabelas responsáveis por representar todas as estruturas existentes na modelagem utilizada. Este modelo pode ser expandido para termos mais funcionalidade ou características importantes. Para o presente estudo este foi o escopo definido, quanto mais informações forem armazenadas no banco de dados da Camada de Independência mais isenta será a aplicação da lógica de negócios do fluxo. Para a implementação do protótipo este nível de informações foi o suficiente, limitando assim a este número de tabelas, para trabalhos futuros esta

estrutura deve ser expandida, possibilitando uma infra-estrutura mais completa para o desenvolvimento e modelagem de processos sobre *workflow*.

Como já citado, os relacionamentos existentes entre as tabelas implementam os controles existentes no modelo. Relacionamentos estes, que podem ser: controles de acessos, troca de estados, ações correspondentes para os estados, grupos, papéis e demais estruturas do processo modelado. A seguir serão analisados os principais relacionamentos implementados, os relacionamentos não serão vistos de uma forma única, porém serão agrupados de acordo com as estruturas específicas que implementam, os relacionamentos abordados foram gerados, assim como as a tabelas apresentadas, pelo de definição de tabelas e relacionamentos do banco de dados SQL Server:

- **Controle de Acesso para os Estados:**
  Relaciona as tabelas de modo a gerar o controle de acesso para cada um dos estados existentes no processo. O acesso será definido de acordo com a tabela de Acesso. Informando qual o controle de acesso permitido aos usuários para cada etapa do processo. Permite que sejam controlados quais os estados ou situações, de modo que possam ser informados quais usuários, papéis ou grupos de usuários terão acesso sobre os determinados estados no fluxo. Este relacionamento é representado na Figura 24.

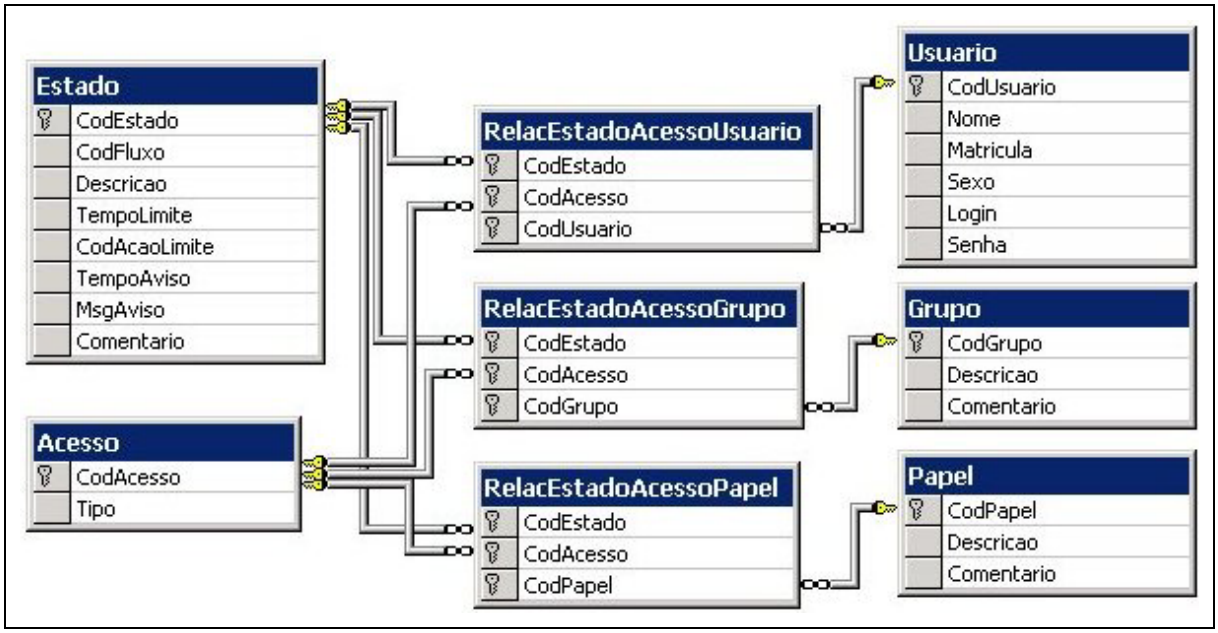


FIGURA 24 - Relacionamentos do Acesso para os estados

- **Controle de Acesso para o Fluxo:**
  Informa, para os fluxos qual o tipo de acesso será atribuído aos usuários, grupos e papéis existentes. Como exemplo, pode ser citado: incluir documentos, excluir documentos, apenas consultar, nem mesmo consultar. O controle de acesso para o fluxo é um controle de acesso em um nível mais elevado, não faz parte do seu escopo detalhes da estrutura. Este relacionamento é representado na Figura 25.

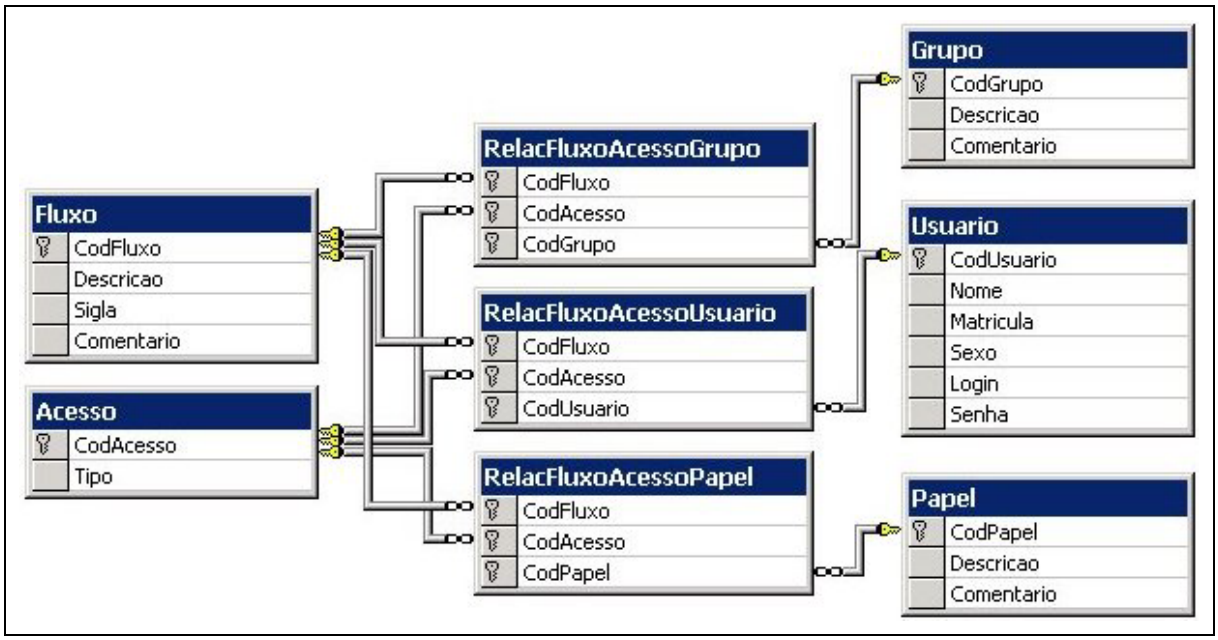


FIGURA 25 - Relacionamentos do Acesso para o Fluxo

- **Usuários e suas Ações:**
  Incluindo o controle de acesso, todas as ações que podem ser realizadas pelos usuários, de acordo com o estado em que o documento ou processo se encontra. As condições, além do estado do documento, para a execução das ações, estão armazenadas na tabela de ações. Todas as ações referentes ao fluxo serão controladas por estes relacionamentos, relacionando usuário, grupos e papéis. Este relacionamento é representado na Figura 26.

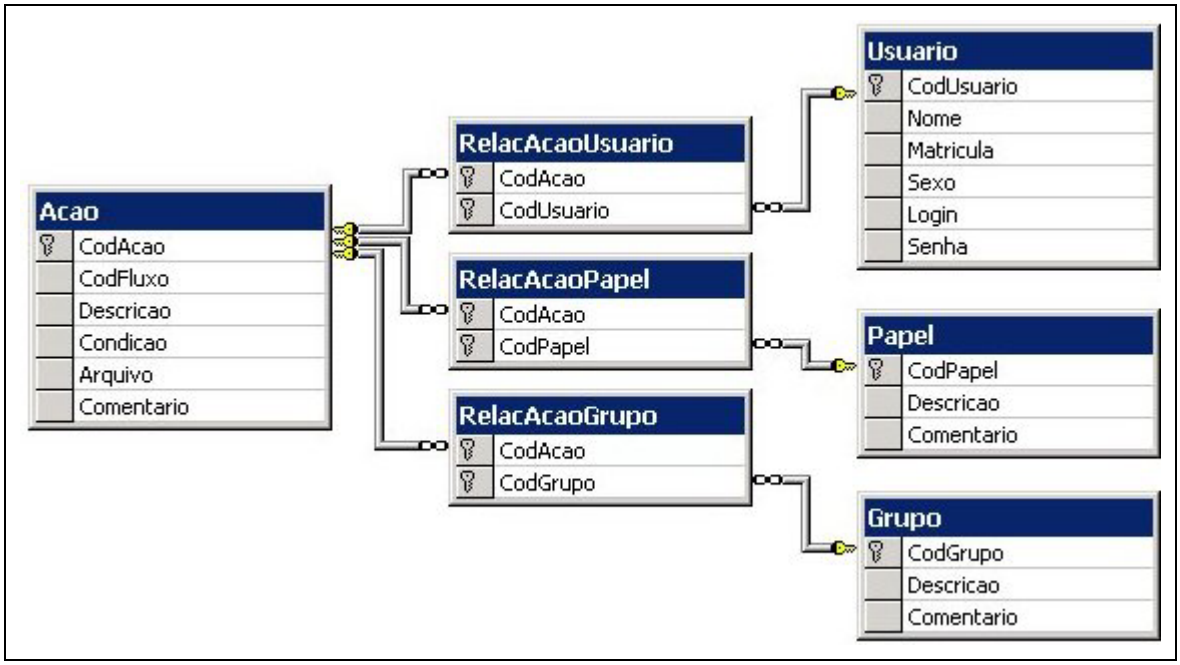


FIGURA 26 - Relacionamentos para as Ações e Usuários

- **Os Campos existentes para o Processo:**

  Todos os campos que forem de relevância ao processo estão armazenados nesta estrutura. Não são armazenados os valores contidos nos campos, pois isso é tarefa do banco de dados do motor, uma vez que está sob sua responsabilidade o controle das instâncias. O controle dos valores dos campos diz respeito ao motor de *workflow*, pois os campos fazem referência a uma instância específica. Este relacionamento é representado na Figura 27.

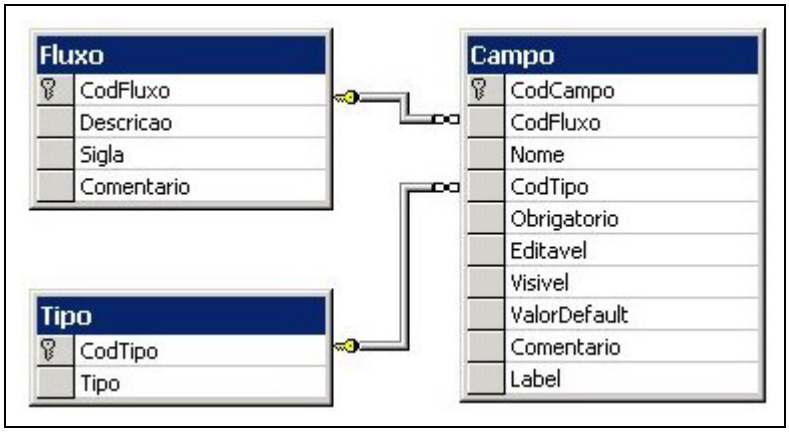


FIGURA 27 - Relacionamentos para os Campos

- **Troca de Estados e as Ações e elas relacionadas:**
  Indica qual a ação irá desencadear a mudança entre os estados. Este relacionamento é representado na Figura 28.

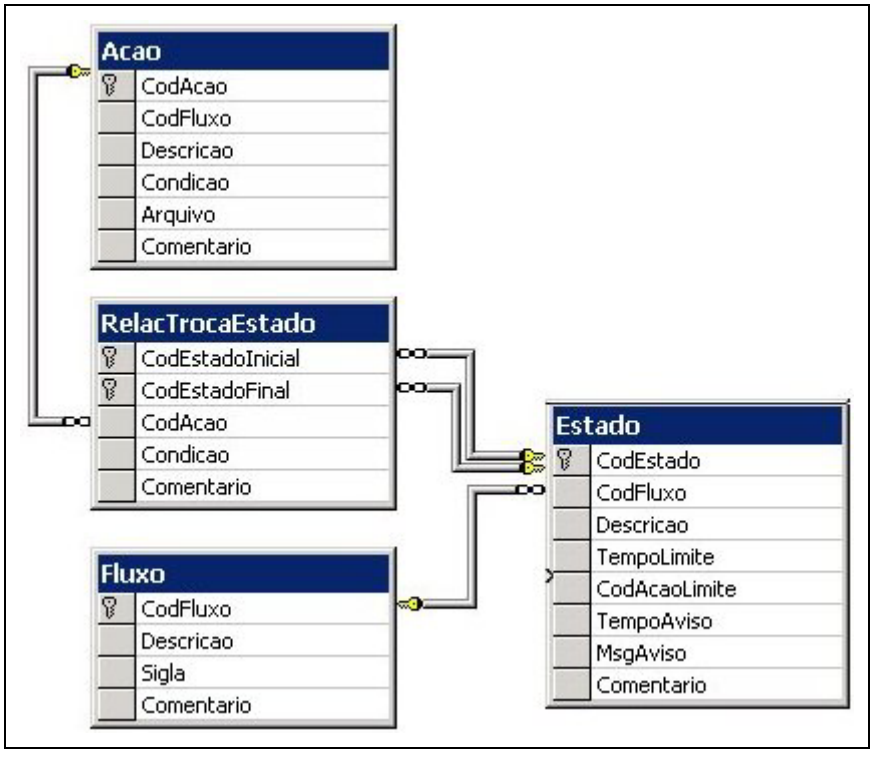


FIGURA 28 - Relacionamentos em entre os Estados, Ações e Fluxo.

- **Grupos relacionados aos usuários:**
  Implementa a organização dos usuários e o(s) grupos(s) ao(s) qual(is) pertence(m). Este relacionamento é representado na Figura 29.

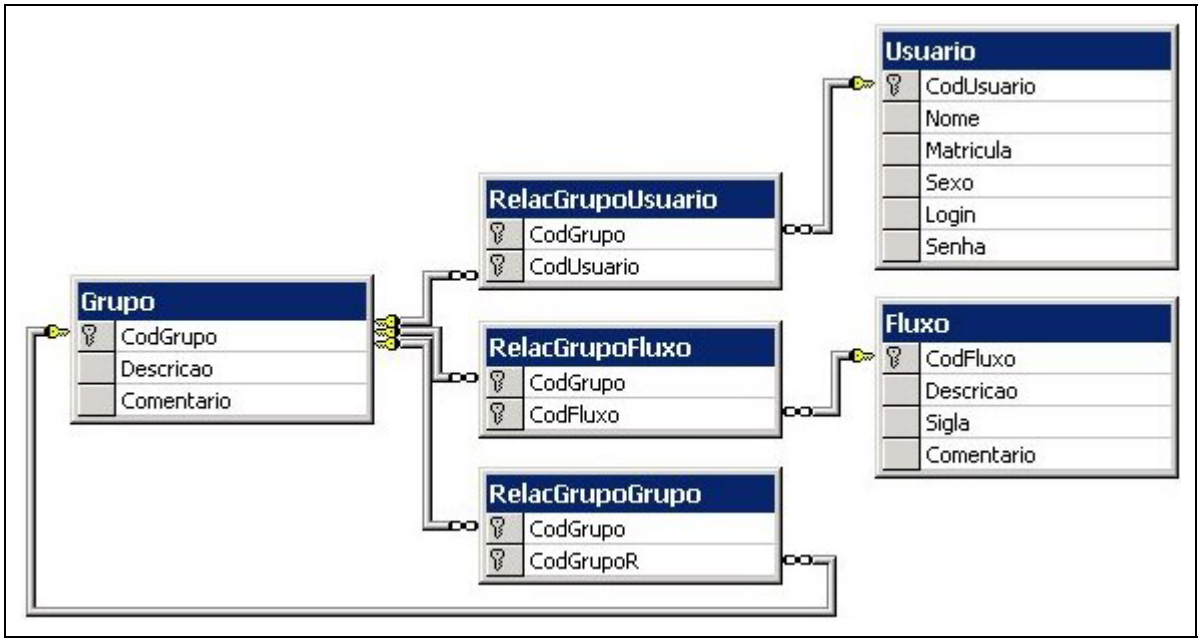


FIGURA 29 - Relacionamentos entre Grupo e demais tabelas

- **Papéis relacionados aos usuários:**
  Implementa a organização dos usuários e o(s) papel(éis) ao(s) qual(is) pertence(m). Este relacionamento é representado na Figura 30.

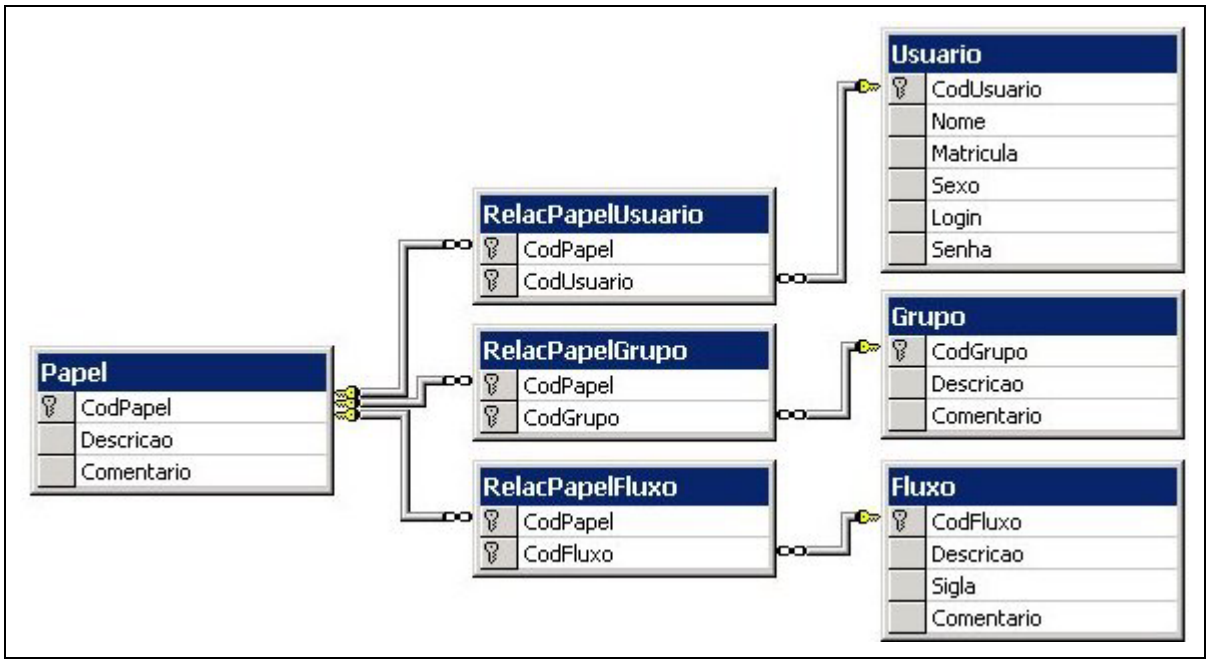


FIGURA 30 - Relacionamentos para as Regras ou Papéis

6.5.2 Banco de Dados da Camada de Integração

Para a Camada de Integração temos uma estrutura de tabelas mais simples, pois a característica deste banco de dados não está na quantidade de tabelas e relacionamentos, mas sim na complexidade das informações armazenadas. Uma vez que sua função é relacionar os elementos existentes na Camada de Independência com os elementos existentes no motor de *workflow*.

Para algumas tabelas utilizadas pelo motor de *workflow*, existe uma correspondência direta entre as tabelas da Camada de Independência, não havendo assim, necessidade de uma tabela que mapeie esta relação, para os demais casos foi necessário criar novas tabelas para realizar este mapeamento. Sempre ser lembrado que essa organização se deve exclusivamente às características estruturais do ambiente de execução do W4.

As tabelas que necessitaram este mapeamento foram as seguintes: tabela de Fluxo, tabela de Campo e tabela de Ação. Posteriormente, quando for descrita a estrutura completa da Camada de Integração serão expostas as razões para a criação de cada uma delas. Neste momento apenas serão descritas as estruturas criadas. Foram criadas três tabelas para armazenar as relações existentes entre os diferentes conceitos, que são:

- **Fluxo_Procedure**:

FIGURA 31 - Tabela de Fluxo_Procedure da Camada de Integração

A tabela de Fluxo_Procedure é a tabela que tem como função armazenar o relacionamento entre a tabela de Fluxo, da Camada de Independência, e a tabela Procedure, do banco de dados do motor de *workflow*. Poderia-se denominá-la como a tabela de maior nível, pois define qual o processo será modelado, todas as demais estarão relacionadas a esta direta ou indiretamente. Traçou-se uma relação entre estes dois elementos que representam o fluxo que está sendo modelado. Como exposto na Figura 31, temos as seguintes informações armazenadas:

- CodFluxo: atributo que representa o código para a tabela de Fluxo existente na Camada de Independência;
- Procedure_ID: atributo que representa o código do fluxo que está sendo modelado para o motor, neste caso para a tabela Procedures.

- **Campo_Variable**:

FIGURA 32 - Tabela de Campo_Variable da Camada de Integração

A tabela de Campo_Variable é a tabela que tem como função armazenar o relacionamento entre a tabela de Campo, da Camada de Independência, e as tabelas de Procedure_Variable e de Activity_Variable do banco de dados do motor de *workflow*. Nesta tabela existem duas tabelas do motor envolvidas, pois em ambas as tabelas existem o conceito de Campo, comparando com os conceitos da Camada de Independência. Assim sendo, como exposto na Figura 32, temos as seguintes informações armazenadas:

- CodFluxo: atributo que representa o código para a tabela de Fluxo existente na Camada de Independência;
- CodCampo: atributo que representa o código para a tabela de Campo existente na Camada de Independência;
- Procedure_ID: atributo que representa o código do fluxo que está sendo modelado para o motor, neste caso para a tabela Procedures;
- Procedure_Variable: atributo que representa o código para a tabela de mesmo nome - Procedure_Variable - pertencente ao banco de dados do motor;
- Activity_Variable: atributo que representa o código para a tabela de mesmo nome - Activity_Variable - pertencente ao banco de dados do motor.

- **Acao_Activity**:

FIGURA 33 - Tabela de Acao_Activity da Camada de Integração

A tabela de Acao_Activity é a tabela que tem como função armazenar o relacionamento entre a tabela de Ação, da Camada de Independência, e a tabela de Activity do motor de *workflow*. Assim sendo, como exposto na Figura 33, temos as seguintes informações armazenadas:

- CodFluxo: atributo que representa o código para a tabela de Fluxo existente na Camada de Independência;
- CodAcao: atributo que representa o código para a tabela de Ação existente na Camada de Independência;
- Activity_ID: atributo que representa o código para a tabela Activity pertencente ao banco de dados do motor.

Foram relacionadas às ações criadas na Camada de Independência, na tabela de Ações, às atividades existentes na a tabela de Activity do motor de *workflow*. Deste modo, os eventos destinados ao disparo de ações executadas pelas aplicações, podem ser repassados da Camada de Independência para a Camada de Integração, e esta por sua vez efetivar a ação do motor, através de suas API's.

6.6 Camada de Independência

A Camada de Independência tem como papel retornar, para as aplicações, todas as informações referentes ao fluxo utilizado pelas mesmas, o objetivo desta camada é o objetivo principal deste trabalho, ou seja, fornecer apoio de forma que a aplicação não tenha necessidade de ter informações referentes à lógica do fluxo.

Através da API possibilitará às aplicações o retorno de todas as informações modeladas no módulo de definição do fluxo. O seu papel é na realidade uma expansão das estruturas de armazenamento do motor de *workflow*, de modo a permitir a completa especificação de cada uma das: tarefas, ações, papéis, permissões, trechos de códigos a serem executados. Todas as informações que atualmente ficam dissociadas na implementação da aplicação estarão armazenadas no banco de dados desta camada e, a partir de funções específicas, serão retornadas às aplicações.

Normalmente o que acontece é a interação direta, das aplicações com o motor, e o retorno das informações sobre a instância ou documento em questão, porém o motor não contempla todo o conhecimento necessário, e com isso o restante da informação necessária para o processo, fica a cargo da aplicação. Este restante refere-se à lógica do fluxo, ou seja, as condições de execução das ações, as permissões de leitura e modificação de valores de campos, e demais informações referentes ao modelo que é feito na etapa de definição. É neste momento que temos a independência, devido a separação criada entre as regras de negócios e a as regras do *workflow*.

6.6.1 API's da Camada de Independência

Podemos enfocar a camada de API's como sendo um facilitador, para as aplicações, visando simplificar a utilização das estruturas de armazenamento. Poderíamos utilizar as estruturas idealizadas sem termos uma camada de API's, porém com isso seria exigido das aplicações o conhecimento necessário para percorrer as tabelas e os relacionamentos de modo a obter as informações necessárias. Temos a Figura 34 como exemplo do objetivo das API's:

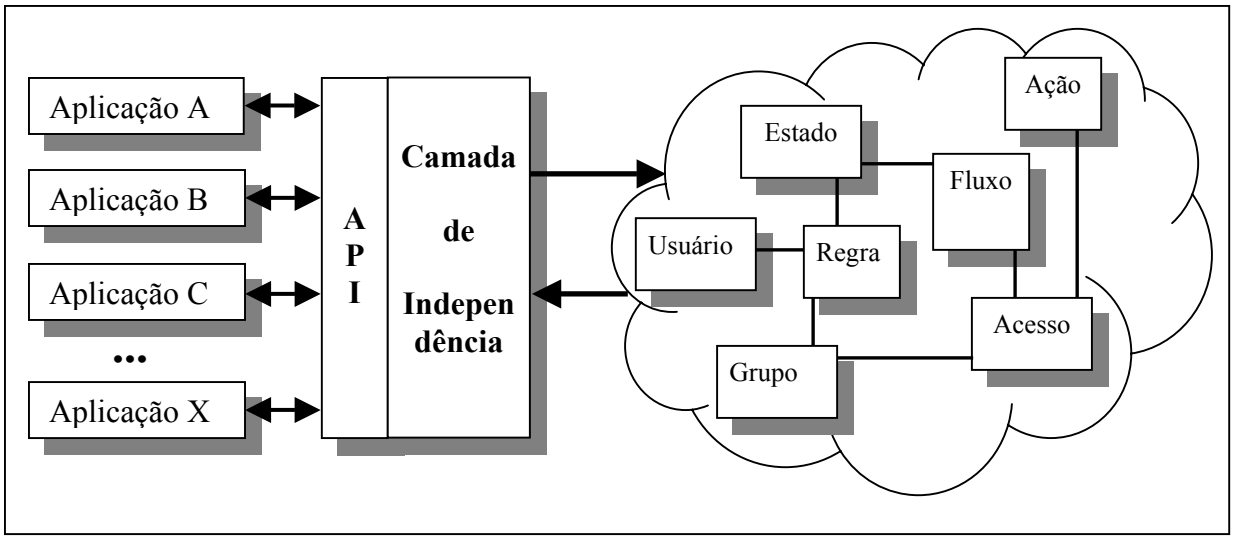


FIGURA 34 - API's da Camada de Independência

Temos na Figura 34 as aplicações acessando somente a camada de API's. As tabelas e relacionamentos da Camada de Independência estão sendo vistas como uma estrutura desconhecida por parte das aplicações, uma vez que para ela apenas o que importa são as funções de manipulação da estrutura que serão utilizadas.

A camada de API's é de extrema importância e deve ser bem extensa e de simples utilização, pois dependerá dela toda a comunicação das aplicações, não só com a lógica do fluxo, mas também com o motor de *workflow*. Não se deve esquecer que ao utilizar a Camada Intermediária não mais haverá comunicação direta com o motor de *workflow*, somente com as API's disponibilizadas pela camada.

Foram implementadas diversas classes para a manipulação dos dados. No total foram desenvolvidas nove classes para a manipulação da lógica dos processos, armazenada no banco de dados. Cada classe representa um elemento existente no modelo.

A linguagem adotada foi o Visual Basic em função de sua praticidade no desenvolvimento das interfaces em aplicações. Para o desenvolvimento das classes não é necessário interface, porém para o desenvolvimento do protótipo foi importante ter uma linguagem que possibilitasse facilidade na criação das telas e demais elementos.

São expostas as classes com algumas características existentes para os seus métodos e funcionalidades; porém a definição completa para cada classe, com suas propriedades e métodos estará descrita somente no ANEXO.

Os dois métodos: "Carrega" e "Consiste", estão presentes em todas as classes, eles correspondem respectivamente os métodos construtores e destrutores das classes. Foi necessário implementar estes dois métodos em função do banco utilizado - SQL Server - ser um banco de dados relacional. Se tivesse sido utilizado um banco de dados orientado a objetos, não teria se feito necessário implementar estes métodos; pois, ao utilizar um banco de dados orientado a objetos, a camada de persistência torna-se transparente às aplicações. Após a criação dos objetos – da Camada de Integração -

devemos chamar o método Carrega, e conseqüentemente antes de liberar o objeto em memória devemos chamar o método Consiste, para que sejam salvos os seus dados.

- **Ação:**

Na classe de ação estão representadas todas ações que podem ser executadas pelos usuários, seu controle de acesso, e demais controles e elementos relacionados às ações existentes. Todas as informações relacionadas são carregadas ao se iniciar o objeto, este sempre relacionado a um documento.

Detalhando um pouco o método "Executa", cabe salientar que o método está relacionado com a classe da Camada de Integração, uma vez que a execução da ação envolve a ativação do motor de *workflow*. Sendo assim, a chamada deste método repercute na passagem desta requisição à Camada de Integração, que irá através do seu mapeamento invocar a respectiva ação do motor de *workflow*.

- **Campo:**

A classe de campo representará as informações referentes ao fluxo que está sendo modelado. Esta classe não contém métodos específicos para o tratamento de elementos relacionados à lógica do fluxo. Seus métodos referem-se somente às suas próprias informações, ou sejam, manipulam cada uma de suas propriedades. O controle de acesso é pressuposto da classe de campo.

- **Documento:**

A classe documento possui uma grande quantidade de elementos associados, é uma das mais completas classes implementadas. Temos diretamente associadas à classe de documento, as seguintes classes: Campo, Ação, Estado e Acesso.

Ao se instanciar um novo objeto deve-se informar, no método "Carrega", qual a chave do documento, e a partir desta chave, todos as demais informações serão carregadas. Alguns dos métodos existentes são:

- getCampos
  Retorna uma coleção de objetos do tipo campo. Estes campos são todos os dados existentes para esta instância de documento ou processo. Para o retorno dessas informações é chamada, a partir, da Camada de Integração os valores atuais para cada um dos campos;
- getAcoes
  Retorna uma coleção de objetos do tipo ação. Estas ações são todas as ações que poderão ser executadas pelo usuário atual, informado a partir da classe de sessão. Para o retorno dessas informações não é necessária a Camada de Integração, uma vez que ações dizem respeito ao fluxo do processo, ou seja, está armazenada no banco de dados da própria Camada de Independência;
- getAcesso
  Retorna o controle de acesso atual para este documento para o usuário informado pela atual sessão.

- **Estado:**

A classe de estado, assim como a classe documento, é uma classe de grande importância; pois nela estará representado todo o fluxo do processo. Algumas de suas funcionalidade foram previstas no modelo de dados, porém, por questão de tempo não foram utilizadas no protótipo. Entre os seus métodos podemos citar

  - getTempoLimite
    Retorna o tempo máximo admitido para o documento permanecer em determinada situação, uma vez alcançado este período o documento será encaminhado para o próximo estado indicado no processo. Essa funcionalidade foi prevista, mas para este protótipo, não foi implementada;
  - getCodAcaoLim
    Retorna o código para o estado que o documento deve ser encaminhado ao final do seu tempo limite.
  - getTempoAviso
    Retorna um tempo limite, que uma vez decorrido, enviará uma mensagem especificada para um usuário ou grupos de usuários, solicitando que alguma ação seja executada, este evento deve ser utilizado anteriormente ao tempo final permitido pelo documento para o estado em questão.

- **Fluxo:**

A classe de fluxo será responsável por comportar todas as demais, é a classe pela qual os documentos são recuperados juntamente com todos os estados existentes no processo. A classe fluxo será instanciada após ser devidamente iniciada uma sessão, a partir da identificação de um usuário. A classe fluxo contém vários métodos importantes para o fluxo, mas alguns dos principais são:

  - GetDocumentosPorEstado
    Retorna todos os documentos que estiverem no estado especificado pela passagem de parâmetros, uma coleção de documentos será retornada. Este método irá invocar a Camada de Integração para receber as informações sobre os documentos existentes;
  - GetDocumentosPendentes
    Este método foi previsto, mas para este protótipo não foi implementado. Sua função será retornar todos os documentos, para um fluxo específico, que estiver dependendo de alguma ação do usuário. Este método é de extrema utilidade para quando for ser implementada a caixa de pendência para os usuários. A caixa de pendências é o local onde são informados todos os documentos, para todos os fluxos aos quais o usuário interage, e que existam pendências;
  - GetDocumentos
    Este método retornará todos os documentos, independente do estado em que estejam, vale lembrar que todos estes métodos baseiam-se no controle de acesso dos usuários para o retorno das informações.

- **Grupo:**

Esta classe representa os grupos existentes na organização ou empresa, a informação dos grupos deverá estar replicada no banco de dados do motor ou integrada com o sistema de diretórios utilizado. Para este protótipo foi implementada uma hierarquia de modo a comportar um nível de recursão, ou seja, pode-se ter grupos

pertencentes a grupos; porém apenas em um nível. Para fins de uma utilização concreta será importante ter-se esta recursão não limitada a somente um único nível; porém como o objetivo inicial é somente servir a um protótipo, não foi destinado maior tempo e envolvimento para alguns detalhes.

- **Regra (ou Papel):**

Esta classe representa as regras ou papéis existentes no fluxo, assim como a classe de grupo foi implementada uma estrutura capaz de comportar a existência de regras pertencentes a outras regras, porém apenas em um nível. Um de seus métodos é:

  - getMembros
    Retorna todos os usuários pertencentes à regra. Serão informados todos, independentes se sua ligação é direta ou em função de pertencerem a algum grupo ou a alguma outra regra.

- **Sessão:**

A classe de sessão será o modo como os usuários, através das aplicações, iniciam a interação com os documentos e o processo. Deve existir um usuário com acesso mínimo para a manipulação do fluxo ao qual for manipular. Os métodos analisados são os métodos que são utilizados para o início desta interação, disponibilizando a partir destes métodos iniciais os demais elementos envolvidos, estes métodos são:

  - Login
    Após a correta identificação de um usuário e senha, com o mínimo de permissão necessária para o processo em questão, é criado um objeto sessão. A partir deste objeto de sessão todos os demais elementos são solicitados;
  - getFluxo
    Retornará um novo objeto do fluxo informado como parâmetro. O usuário atual da sessão deve ter acesso mínimo para manipular ou interagir no fluxo ao qual está solicitando a nova instância.

- **Usuário:**

A classe de usuário tem um papel mais significativo, em função do controle de acesso envolvido nos elementos existente do fluxo e o controle existente para a utilização dos processos. Alguns dos métodos disponibilizados são:

  - getRegras
    Retorna todas as regras às quais o usuário está associado, assim como no caso da classe de regra, independente de estar associado diretamente a regra ou em função de algum grupo ao qual pertença;
  - getGrupos
    Retorna todos os grupos aos quais o usuário está associado, sempre respeitando o máximo de um nível para a recursão entre grupos.

Temos a seguir, representado pela Figura 35, o diagrama de classes completo, segundo os padrões de UML, para a modelagem de objetos:

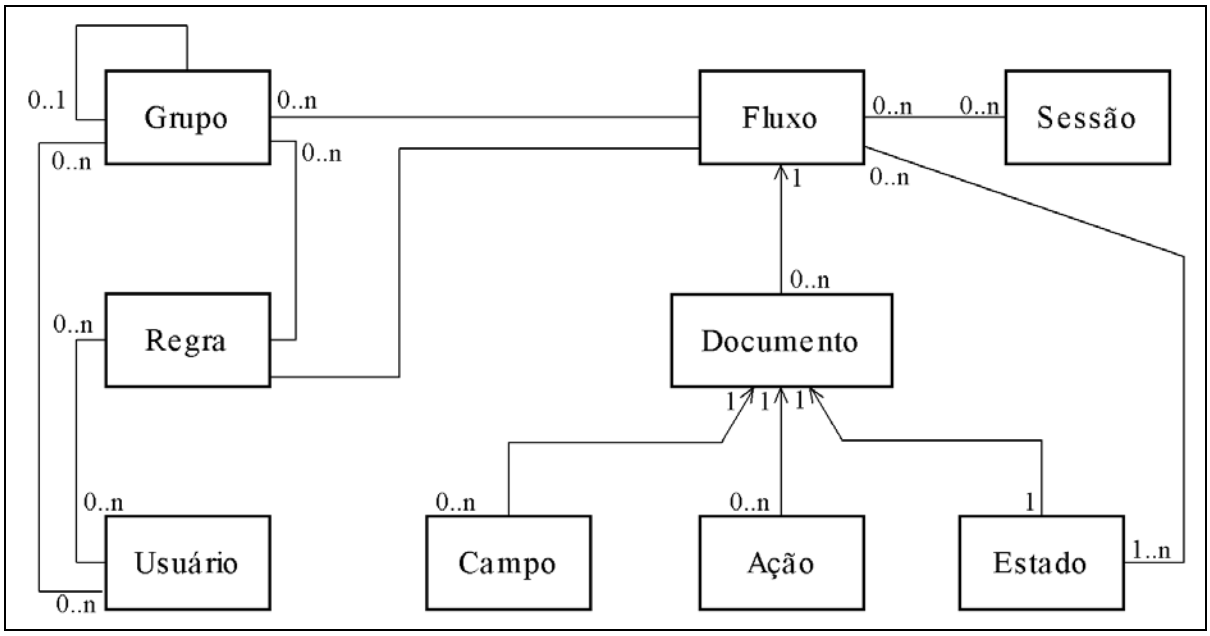


FIGURA 35 - Diagrama das classes da Camada de Independência

6.7 Camada de Integração

A Camada de Integração é a responsável pela conversação entre o motor de *workflow* e Camada de Independência, fazendo assim a integração entre a duas partes. Cabe salientar que esta camada tem suas características de integração especificamente para o motor que se estiver integrando, e não para qualquer motor de *workflow* existente. Para existir a integração com vários motores se teria que adaptar as suas funções (Camada de Integração) para cada motor envolvido. É possível realizar esta integração com vários motores, porém se faz necessário realizar um grande estudo para cada motor, de modo a adaptar as suas características de modelagem de fluxos e arquitetura de suas API's, adequando-as as estruturas de dados utilizadas pela Camada de Independência.

O seu papel é o de receber requisições da Camada de Independência (requisições estas referentes ao controle do fluxo das instâncias), encaminhar a requisição ao motor e retornar as informações obtidas do motor. A Camada de Integração se faz necessária uma vez que não é tarefa da Camada de Independência ter o controle das informações ou gerenciar as instâncias dos processos, mas sim ter "conhecimento" de todas as informações sobre a sua modelagem (estrutura do fluxo). Basicamente a Camada de Integração estará com o mapeamento, através do motor, das informações referentes ao controle sobre as instâncias existentes. Estas funções referenciam as API's do motor, realizando um meio de campo entre a Camada de Independência e o motor. As funções disponibilizadas pela Camada de Integração irão mapear as características de modelagem do motor para os conceitos adotados pela modelagem da Camada de Independência.

Um ponto de grande complexidade é identificar os diferentes conceitos e enquadrá-los, se possível, para as duas modelagens, uma vez que podemos ter motores com diferentes metodologias de modelagem para seus fluxos. Deve ser feito um extenso

estudo de modo a traçar as analogias entre os dois modelos, conseguindo assim, uma representação equivalente ao que se propõem na Camada de Independência.

A Camada Intermediária, proposta para a independência entre as aplicações e os motores de *workflow*, foi dividida em Camada de Independência e Camada de Integração visando à possibilidade de seu desenvolvimento de forma separada. A Camada de Integração foi projetada e desenvolvida em uma parceria com um colega da graduação, ela foi apresentada como proposta de Trabalho de Diplomação de Sandro Coutinho Emerim sobre a titulação de **Estudo para Implementação de uma Camada de Integração Aberta entre Aplicações e Motores de *Workflow***. Desde modo, foi possível uma dedicação maior à Camada de Independência e ao restante do trabalho.

A seguir são apresentados alguns dos aspectos importantes sobre o desenvolvimento da Camada de Integração, serão apenas expostos os tópicos e uma breve análise sobre cada um dos itens. O enfoque que será dado à análise será sucinto, uma vez que não é o escopo deste trabalho uma completa explanação sobre esta camada.

- **Definir a forma como serão trocadas as informações entre a Camada de Independência e a Camada de Integração:**
  Este ponto é fundamental para que a Camada de Independência trabalhe de modo eficiente e uniforme para quaisquer definições e implementações realizadas na Camada de Integração. Desta forma estaremos garantindo a portabilidade do modelo para todo motor de *workflow* para o qual seja necessário implementar a integração;

- **Traçar paralelos entre as estruturas utilizadas no armazenamento dos registros dos dados no motor de *workflow* e as estruturas utilizadas no modelo proposto:**
  Para que a Camada de Integração seja funcional precisamos gerar equivalências (ou eventualmente traduções) de um conceito para outro, na forma das estruturas de armazenamento (tabelas). Por exemplo, é necessário que se mapeie "Tabela A no modelo do motor corresponde a uma tradução de Tabela X do modelo proposto";

- **Diferenciar o tratamento das Instâncias de documentos do tratamento das estruturas definidas no fluxo:**
  Para as aplicações que acessam os documentos (instâncias), em muitas situações não é importante o conhecimento da estrutura do fluxo que se está utilizando, como exemplo, podemos citar a execução de uma ação. Para estes casos, é suficiente que sejam retornadas informações sobre as instâncias. As funções de conversação entre as camadas internas devem ser consideravelmente diferenciadas para que não gere complexidade durante a sua utilização;

- **Definir as tabelas de integração com o motor designado:**
  Como a integração deve ser específica para cada motor utilizado, é necessário customizar a Camada de Integração de forma que acesse as informações de instâncias e estruturas do fluxo disponibilizadas pelo motor. Nesta adaptação

incluem-se tabelas de armazenamento e mapeamento das funções disponibilizadas (API's);

- **Estabelecer uma "carga inicial" da estrutura do motor para a estrutura da Camada de Independência (importação dos dados):**
  Deve existir uma forte integração entre a Camada de Integração e o motor de *workflow*, inicialmente estabelecida pela importação das estruturas armazenadas no banco de dados do motor para que seja disponibilizada pela Camada de Independência. Posteriormente deverá existir um processo constante de sincronização das duas estruturas.

6.7.1 API's da Camada de Integração

O conjunto de funções disponibilizadas por sua API é restrito, se comparado com as funções existentes na API da Camada de Independência. Além do limitado número de funções disponibilizadas, como diferença entre as camadas, temos a função realizada pela API da Camada de Integração, que é um mapeamento entre as funções disponibilizadas pela Camada de Independência e a API do motor de *workflow*.

Podemos ilustrar o objetivo das funções disponibilizadas de acordo com o diagrama a seguir, onde temos, ilustrada a relação que existe entre cada uma das funções das API's envolvidas.

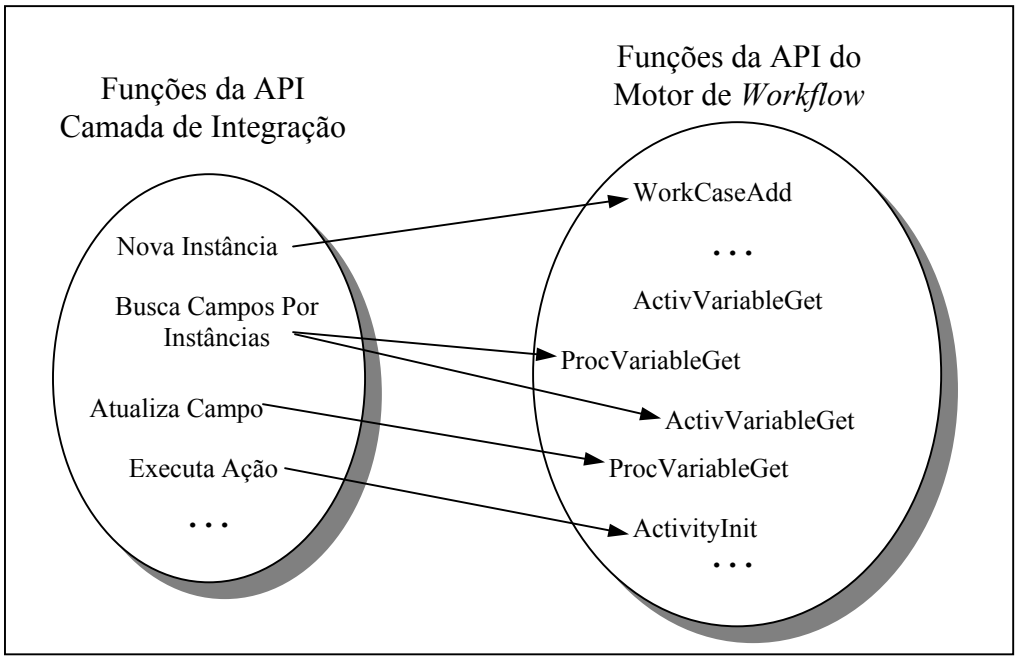


FIGURA 36 - Representação do mapeamento da API's da Camada de Integração

Conforme a Figura 36 temos um mapeamento entre as funções disponibilizadas pelo motor de *workflow* e pela API, para cada função existente na Camada de Integração deve existir uma função associada na API do motor. Para alguns casos pode haver mais de uma função associada no motor de *workflow*, quando isso ocorrer será em

função do diferente enfoque existente entre os conceitos, para o tratamento dos documentos e processos, entre o motor analisado e a Camada Intermediária.

Partindo das funções definidas pela Camada de Independência (disponibilizadas para as aplicações para o controle das instâncias) é necessário mapear quais as correspondências na API do motor de *workflow*.

- NovaInstancia (string fluxo)
  A função do motor **WorkCaseAdd,** criará uma nova instância do fluxo;

- BuscaTodasInstancias ()
  Para esta função da Camada de Independência não existe correspondência, tornando-se necessário um mapeamento para as tabelas do motor de *workflow*;

- BuscaCamposPorInstancia (int docID)
  As funções: **ProcVariableGet** e **ActivVariableGet**, pertencentes ao motor, são as responsáveis por retornar todos os valores de campos relativos à instância especificada;

- BuscaTodasInstanciasPorEstado (string estado, string fluxo)
  Para esta função da Camada de Independência não existe correspondência, tornando-se necessário um mapeamento direto a partir das tabelas;

- AtualizaCampo (string campo, string valor)
  As funções: **ProcVariableSet** e **ActivVariableSet,** pertencentes ao motor, são as responsáveis por atualizar o valor do campo especificado na chamada desta função.

- ExecutaAcao (string ação, int docID)
  A função do motor **ActivityInit** executará a atividade relacionada à ação da Camada de Independência, especificada na chamada desta função.

No próximo capítulo, serão abordadas as características encontradas no motor de *workflow*, adotado para o desenvolvimento da Camada de Integração. Não serão abordados em sua totalidade os aspectos e características encontradas no motor; pois este trabalho já foi definido no estudo citado anteriormente. São expostos os aspectos referentes à contextualização da Camada de Integração para este trabalho.

6.7.2 Motor de "Workflow" - W4 (*World Wide Web Workflow*)

O W4 é um produto voltado para o mercado corporativo, que atende as necessidades das organizações no que diz respeito à utilização de *workflow*, e tem uma filosofia completamente voltada para a Internet / Intranet (Web). Foi concebido e estruturado para que toda sua navegabilidade e usabilidade fossem portadas para os navegadores, a fim de facilitar a integração dos mais diversos ambientes operacionais e a sua utilização em diversas plataformas.

Funcionalmente podemos enquadrar o W4 na arquitetura descrita no capítulo 2, onde temos modelagem e execução em ambientes distintos. Possui um Motor (*engine*)

responsável pela criação e manipulação das instâncias dos fluxos, uma ferramenta de autoria, onde são definidos os fluxos e procedimentos, e uma API, através da qual as aplicações acessam as instâncias de forma dinâmica.

A modelagem dos fluxos se dá através de uma ferramenta denominada W4Author, disponível para a plataforma Windows, que trabalha basicamente em dois modos:

- Standalone, sem conexão com o banco de dados do Motor W4. Desta forma os fluxos e procedimentos são salvos apenas localmente, em arquivos de arquitetura proprietária, podendo ser posteriormente enviados ao servidor;
- Conectado ao servidor, onde é possível salvar os fluxos diretamente no banco de dados do Motor.

É possível verificar, em ambos os modos, se existe alguma inconsistência no fluxo, sendo que para o armazenamento no servidor é estritamente obrigatório que não exista qualquer erro na estrutura. Uma característica interessante do envio para o servidor é que o arquivo pode ser gravado como uma modificação em um fluxo já existente, ou podem ser criadas novas versões, denominadas “Revisions”, que facilitam a manutenção de cópias dos fluxos dentro do banco de dados do W4.

Como toda ferramenta de modelagem o W4Author possui uma simbologia própria e uma notação simbólica para cada elemento dentro do fluxo. Não cabe aqui analisar profundamente cada um destes símbolos, sendo necessário apenas uma leve explanação sobre cada um dos seus elementos. As suas tabelas de armazenamento serão expostas mais adiante quando forem demonstradas as estruturas de armazenamento do motor. Um exemplo da estrutura de modelagem utilizada está ilustrado na Figura 37.

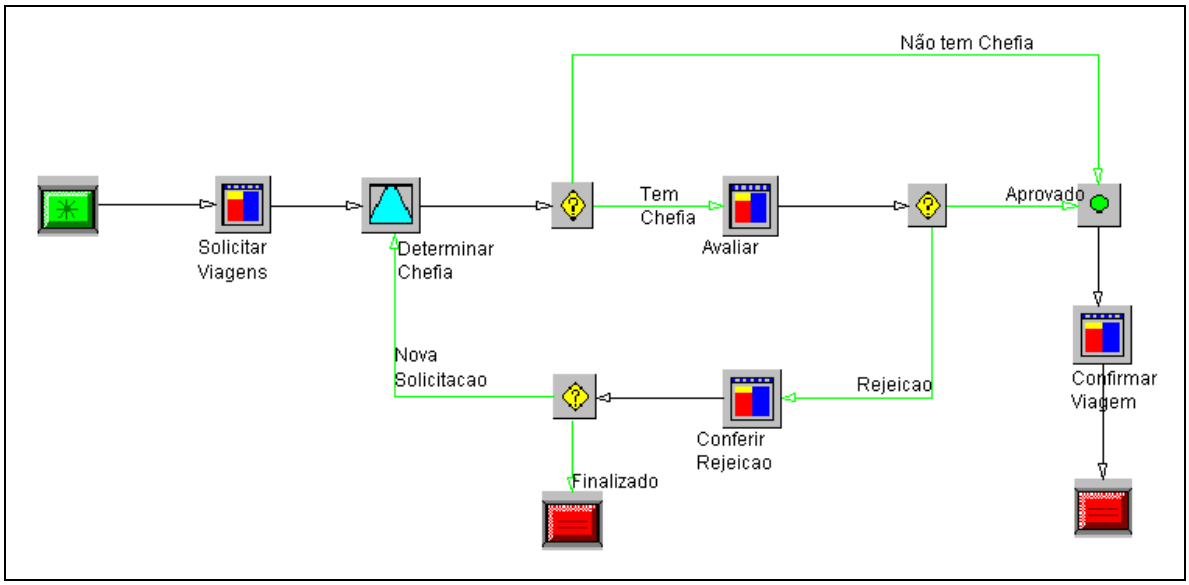


FIGURA 37 - Exemplo de estrutura utilizada para modelagem no W4

Como visto na Figura 37, temos um grande conjunto de elementos, componentes da modelagem do processo. Neste momento podemos analisar a forma como a ferramenta CASE deste ambiente se comporta, quais os elementos são disponíveis, e o

modo como devemos modelar os processos. Para este ambiente, a modelagem é orientada a atividades, temos, para cada um dos passos a serem realizados, etapas a serem executadas pelos usuários. A seguir será apresentado cada um dos elementos existentes no modelo apresentado:

- **Start Node:**
  Ponto inicial (Partida) da Procedure ou fluxo. Este elemento é único em todo o fluxo, ou seja, não devem existir dois pontos de partida para os processos modelados. Suas propriedades são: Nome e Descrição. Sua representação é mostrada na Figura 38.

FIGURA 38 - Start Node

- **Stop Node:**
  Ponto final da Procedure. Este elemento pode estar presente em mais de um momento na modelagem, ou seja, é previsto que possa existir um processo com mais de um ponto de final. Suas propriedades são: Nome e Descrição. Sua representação é mostrada na Figura 39.

FIGURA 39 - Stop Node

- **Basic Activity:**
  Denota uma atividade básica dentro da Procedure. Pode ser traçada uma analogia com os estados utilizados na modelagem adotada pela Camada Intermediária. Suas propriedades são: Nome, Atividade Correspondente, Variáveis Envolvidas, Responsáveis, Duração e Prioridade. Sua representação é mostrada na Figura 40.

FIGURA 40 - Basic Activity

- **Condition:**
  Seu papel é apenas indicativo (meramente ilustrativo), suas propriedades são: Nome e Descrição. Utilizada para ilustrar uma condição dentro da Procedure Sua representação é mostrada na Figura 41.

FIGURA 41 - Condition

- **Automated Activity:**
  Atividade automatizada dentro do fluxo, será executada sem a interação dos usuários, suas propriedades são: Nome, Atividade Correspondente, Variáveis Envolvidas, Responsáveis, Duração e Prioridade. Sua representação é mostrada na Figura 42.

FIGURA 42 - Automated Activity

- **Join:**
  Junção para que duas ou mais condições levem a determinado ponto do fluxo. Suas propriedades são: Nome e Descrição. Sua representação é mostrada na Figura 43.

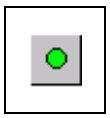


FIGURA 43 - Join

- **Conditional Transition:**
  Ligação entre um nodo e outro, baseada em uma condição lógica (Verdadeiro ou Falso). Suas propriedades são: Nome e Condição (para que a transição ocorra entre os dois nodos conectados). Sua representação é mostrada na Figura 44.

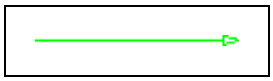


FIGURA 44 - Conditional Transition

- **Transition:**
  Ligação simples entre um nodo e outro. Sua propriedade é somente Nome. Sua representação é mostrada na Figura 45

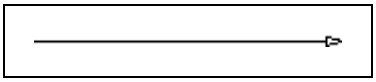


FIGURA 45 - Transition

6.8 Arquitetura Completa Detalhada

Re-avaliando a arquitetura de forma completa temos todos os elementos que compõem este estudo, desde as aplicações até o motor de *workflow*. Foi acrescentado um elemento que, em função de não interferir no estudo apresentado, até então não havia sido citado. Este elemento é o módulo que realiza a modelagem dos fluxos, ou simplesmente o CASE dos motores de *workflow*. Nele estará disposta a metodologia utilizada por cada fabricante para modelar seus processos ou atividades, bem como as funcionalidades existentes e demais características do produto.

Foi adicionado ao diagrama um módulo de CASE também para a Camada de Independência, este, porém está apenas tracejado; pois se trata de um trabalho futuro. Não foi abordado, como parte do escopo deste trabalho, o estudo de uma ferramenta de modelagem para os processos. Uma pequena explanação é realizada sobre o CASE da Camada de Independência como parte do capítulo 9, onde são propostos trabalhos futuros.

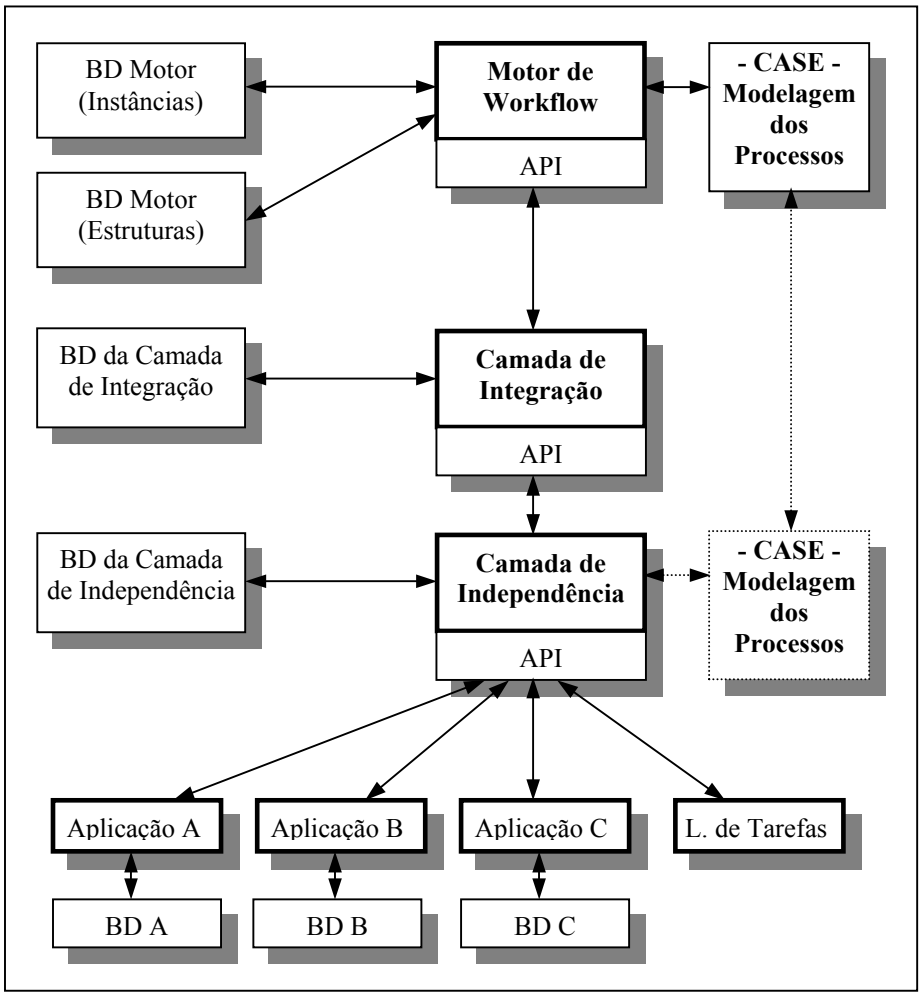


FIGURA 46 - Arquitetura completa do motor de *workflow*

Conforme o diagrama, apresentado pela Figura 46, podem ser analisadas algumas das características importantes:

- **BD Motor:**
  Temos no banco de dados do motor a separação entre Instâncias e Estruturas. Esta separação existe porque são dois os papéis do motor de *workflow*: Primeiro, para as instâncias – documentos ou processos – armazenar as informações referentes ao controle e execução; em segundo, temos o papel para as estruturas – lógica dos processos - modeladas no CASE, com a função de armazenar o conhecimento das etapas existentes e a seqüência das atividades e tarefas, enfim os elementos que definem o fluxo. Como exemplo, podemos citar, os valores dos campos que forem referentes ao processo modelado;

- **BD da Camada de Integração:**
  O objetivo deste banco de dados é manter os dados referentes às equivalências existentes entre os conceitos do motor de *Workflow* e a Camada de Independência. Tem a responsabilidade de armazenar o mapeamento capaz de subsidiar informações, de modo a permitir, para a Camada de Integração realizar o seu papel de um *gateway* entre Motor e Camada de Independência;

- **BD da Camada de Independência:**
  A estrutura implementada no banco de dados da Camada de Independência é o centro de todo o estudo, o objetivo deste banco de dados é armazenar todo o conhecimento existente sobre os papéis, atividades, procedimentos, permissões, enfim, todos os elementos do processo. A lógica dos processos, que agora fica a cargo deste banco de dados, era o ponto que gerava a dependência entre as aplicações e os fluxos; pois, a lógica dos processos, anteriormente, estava dissociada no código das aplicações;

- **BD das aplicações (A, B, C):**
  O banco de dados das aplicações não mantém vínculo algum com qualquer tipo de estrutura utilizada, nem mesmo, com qualquer elemento componente do ambiente do Motor de *Workflow*. São de responsabilidade única e exclusiva das próprias aplicações;

- **Motor de *Workflow*:**
  O Motor de *Workflow* é o responsável pela verificação dos processos, ativação de eventos; é ele quem, de poder das informações dispostas em seu banco de dados, irá gerenciar a tramitação dos processos e demais controles existentes no ambiente. A sua comunicação com a Camada de Integração acontece através de suas API's

- **Camada de Integração:**
  Esta camada é responsável pela conversação com os motores que venham a ser utilizados. Seu papel é realizar o mapeamento e atender a todas as requisições feitas pela Camada de Independência. Seu escopo está delimitado entre o Motor de *Workflow* e a Camada de Independência;

- **Camada de Independência:**
  Esta camada é responsável por atender todas requisições feitas pelas aplicações; é ela que, através de seu banco de dados, irá retornar às aplicações todos os controles que se fazem necessários para a utilização dos processos.

As requisições referentes ao controle dos processos e instâncias serão repassadas à Camada de Integração. As requisições às quais terá controle, por si só, serão as requisições referentes à estrutura dos processos;

- **Aplicações (A, B, C):**
  As aplicações mudam totalmente o enfoque que vinham seguindo até então; pois passam a requisitar qualquer informação sobre os processos, tornando-se isentas de qualquer conhecimento sobre os mesmos. As aplicações permanecem, em função da arquitetura de motor utilizada, podendo ser desenvolvidas em qualquer linguagem de programação que seja capaz de utilizar as API's disponibilizadas, agora, pela Camada de Independência;

- **Lista de Tarefas:**
  Esta é uma aplicação atípica; pois seu papel é o de informar aos usuários sobre os documento que estão sobre sua responsabilidade, funciona como um gerenciador de atividade a serem feitas. Como todo o seu conhecimento está baseado nas informações contidas no banco de dados do motor, não necessita de um banco de dados próprio;

- **CASE do Motor de *Workflow*:**
  É a interface onde são modelados todos os processos, é disponibilizada pelo motor, contém um padrão próprio para a modelagem dos fluxos. Esta ferramenta é de utilização apenas da equipe e desenvolvimento, não sendo de conhecimento dos usuários finais do sistema;

- **CASE da Camada de Independência:**
  A diferença básica existente entre o CASE do motor de *workflow* e o CASE da Camada de Independência baseia-se no escopo de cada um deles. O objetivo deste CASE será de comportar toda e qualquer informação existente sobre o fluxo modelado, ao contrário do que existe hoje para o CASE do motor, onde nem toda a lógica dos processos é modelada, acarretando este ônus às aplicações.

7 Enfoque das aplicações para este novo modelo proposto

O enfoque que temos para as aplicações, desenvolvidas utilizando esta arquitetura, é totalmente diferente do que utilizávamos até então. A mudança é bastante grande, se comparado com a maneira como os sistemas são desenvolvidos atualmente, sem o contexto da independência. Apesar de existir esta mudança, e ela não ser pequena, ela é positiva; pois, ela vem com o objetivo de facilitar o desenvolvimento das aplicações que utilizam a arquitetura do motor de *workflow*, uma vez que irá padronizar o desenvolvimento destes sistemas.

As aplicações ao utilizarem esta arquitetura estarão calcadas a um molde, onde a única tarefa será controlar a interface que é disponibilizada aos usuários. A parte da aplicação referente aos fluxos será uma série de passos que sempre deverão ser seguidos, realizando assim, uma rotina baseada em uma seqüência pré-definida de procedimentos. Deste modo, temos as aplicações como sendo, quando analisadas enfocando o fluxo dos processos, muito semelhantes uma das outras.

Esta semelhança no código referente ao tratamento dos fluxos poderá, caso seja bem estruturada a sua utilização, chegar a um nível de poder compartilhar o mesmo código de procedimentos, implementados no seu código. Deste modo, temos o tratamento do fluxo, como um simples conjunto de rotinas dispostas em determinados pontos da aplicação. Cada conjunto de rotinas com sua função específica, padronizando a inserção do código, para o tratamento do fluxo, em todas as aplicações.

Estes passos consistem em consultas e requisições que deverão ser submetidas ao motor, de modo a obter a informação sobre quais ações, acessos e direitos o usuário poderá executar sobre o fluxo que estiver interagindo. Um aspecto importante a lembrar é que a aplicação em momento algum terá “conhecimento” sobre o que está sendo realizado ou qual o significado de cada uma das ações ou tarefas que estão sendo disponibilizadas pelo motor através das suas consultas. O motor passa a ser a fonte de toda a informação referente ao fluxo do processo, e com isso a aplicação torna-se um simples intermediador entre a interação dos usuários com o fluxo, não se envolvendo com execução de nenhum código além da invocação de procedimentos.

O diagrama a seguir demonstra os passos existentes, de modo a uma aplicação utilizar corretamente o conceito desta arquitetura. São demonstrados os passos de execução, para uma aplicação onde serão expostos todos os documentos (ou instâncias) existentes, e posteriormente um documento será selecionado. Os documentos, para este caso, não foram selecionados pelo estado ou condição que se encontram, simplesmente todos os documentos aos quais o usuário tem acesso de visualização foram retornados.

Após a seleção, são disponibilizadas todas as informações contidas no documento, juntamente com todas as ações que podem ser executadas pelo usuário da sessão, para este documento específico. Uma ação, dentre as permitidas pelo usuário, será executada e após a sua execução será atualizado o banco de dados do motor, assim como o banco de dados da Camada Intermediária, realizando deste modo, o ciclo completo de interação entre usuário, aplicação e motor de *workflow*.

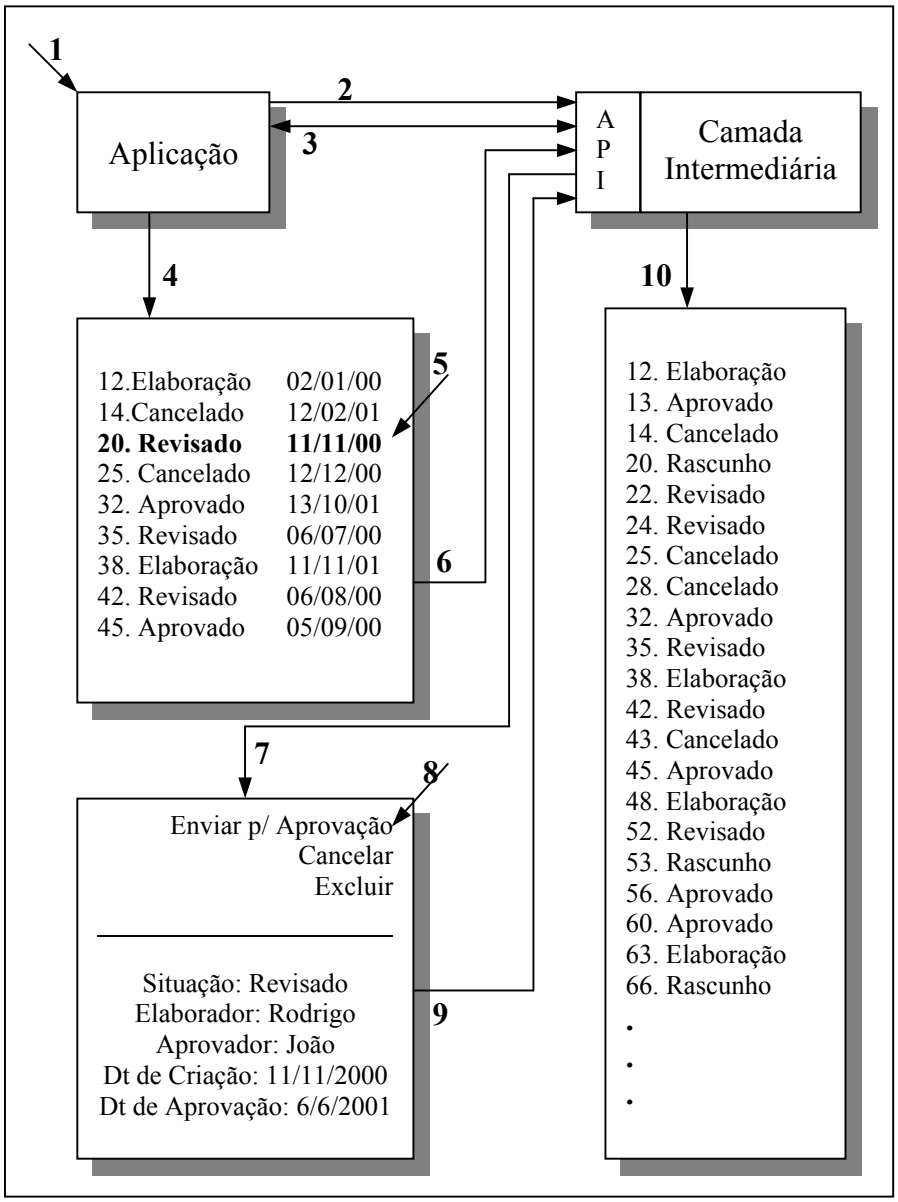


FIGURA 47 - Execução das aplicações ao utilizar a Camada Intermediária

Abaixo são descritos os passos referentes à Figura 47, onde são representadas as ações que devem ser realizadas pelas aplicações, ao utilizar o modelo proposto pela Camada Intermediária. Está representada, no diagrama da Figura 47: a interação do usuário com a aplicação e a interação da aplicação com a Camada Intermediária. Os passos estão descritos de forma sucinta, de forma a transmitir o núcleo do funcionamento dos passos que são utilizados.

1. **Usuário se identifica na aplicação:**
   O usuário, através das aplicações, solicitará os documentos aos quais tem acesso, de acordo com um padrão da aplicação, podem ser solicitadas todos os estados existentes ou apenas algum estado (ou situação) específico. Após a solicitação, são retornados todos os documentos que obedecem ao critério informado pelo usuário. A aplicação não tem conhecimento dos estados existentes no fluxo, ela simplesmente solicita ao motor de *workflow* o retorno

de todos os estados possíveis para os documentos. De posse destes estados, a aplicação os disponibiliza ao usuário para que um seja selecionado ou então, sem ser selecionado nenhum estado, simplesmente são retornados todos os documentos aos quais o usuário tiver acesso. Para este exemplo, todos os documentos foram retornados;

2. **A aplicação solicita à Camada Intermediária todas as instâncias do fluxo (documentos) às quais o usuário tem direito de visualização:**
   É passado como parâmetro apenas a chave do usuário, pode-se informar além da chave do usuário algum critério de seleção para os documentos que se deseja pesquisar, como por exemplo: Somente os documentos que estiverem na situação de "Documentos Aprovados". Neste caso, teria sido apresentado o estado de "Documentos Aprovados" como um dos existentes para este fluxo, de modo a permití-lo selecionar o estado específico ao qual tem interesse para sua pesquisa;

3. **A Camada Intermediária retorna à aplicação as instâncias existentes:**
   A partir da solicitação da aplicação, a Camada Intermediária irá consultar seu banco de dados e verificar quais os documentos o usuário tem permissão de consultar e retornará para a aplicação um cabeçalho (definido em função da economia de tráfego de dados) com apenas parte das informações contidas no documento. O motivo de não retornar todas as informações para cada um dos documentos é por uma questão de desempenho e não desperdício do canal de comunicação. Esta preocupação com o desempenho se dá uma vez que o usuário irá interagir, possivelmente, com apenas alguns documentos, e não com a totalidade a que possivelmente tenha acesso. Assim deixa-se o retorno de todas as informações somente para o momento em que o usuário solicitar, para um documento específico o restante de seus dados. Para facilitar o desenvolvimento do protótipo, todas as informações sobre os documentos foram retornadas a cada solicitação, não tendo assim o conceito de cabeçalho de informações;

4. **Com base nestes dados a aplicação exibe as informações, fornecidas pela Camada Intermediária, ao usuário:**
   A aplicação simplesmente exibe as informações fornecidas, ela não tem conhecimento do seu significado, apenas segue seus padrões de interface, exibindo assim, ao usuário o conteúdo dos documentos (instâncias). Deve ficar bem claro que em momento algum a Camada Intermediária tem o controle da interface, a Camada Intermediária sempre estará aparecendo somente no *back-end*, ou seja, sempre trabalhando em um segundo plano. A aplicação em todos os momentos é a responsável pela interface com os usuários. A aplicação sempre "perguntará" à Camada Intermediária o que poderá e quando poderá exibir (documentos, campos, ações, visualizações, e demais elementos);

5. **O usuário seleciona um documento para ser expandido:**
   A partir da lista de documentos possíveis de serem visualizados o usuário irá selecionar o documento com o qual irá interagir. Ele poderá interagir com quantos documentos forem necessários, porém visando facilitar o

desenvolvimento do protótipo implementado, apenas um documento por vez será manipulado;

6. **A aplicação irá realizar uma nova requisição à Camada Intermediária:** Desta vez será informada a chave do documento, ou dos documentos, em que se necessita o restante das informações. Com as chaves informadas (parâmetros para a chamada da função da camada de API), são retornadas todas as informações (campos) referentes aos documentos. Cabe lembrar que estes campos são os campos referentes ao fluxo do *workflow*, campos que terão informações relevantes na tomada de decisão no fluxo. A idéia é manter sob controle do motor de *workflow* somente os campos que tiverem relevância sob o fluxo em questão. Dependendo de como for realizada a análise, poderão ser modelados campos que não tenham relação direta com o fluxo. Isso deve ser evitado, de forma a manter, como mencionado anteriormente, somente os campos que tenham relação direta com o fluxo, a importância de manter esta separação se deve ao isolamento que deve existir entre a lógica do fluxo e a lógica das aplicações;

7. **O documento é exibido:** Com base das informações obtidas na consulta realizada, a aplicação exibe ao usuário, de acordo com o seu padrão de interface, o documento com todas suas ações, campos e demais elementos. Neste momento alguns aspectos são muito importantes, como: (1) Os campos serão exibidos de acordo com o controle de acesso do fluxo, em função disso não somente serão informados, pela Camada Intermediária, os campos e valores, mas também se eles podem ser editados ou apenas visualizados, ou seja, o controle de acesso associado a cada um deles. A aplicação fica responsável em exibir o campo de modo que permita modificações ou não. (2) As ações são montadas de acordo com as informações que forem retornadas pela Camada Intermediária, a aplicação não toma conhecimento do que implica a realização destas ações, ela receberá um vetor de "strings" que serão as possíveis ações e disponibilizará de acordo com seu padrão de interface (botões, links, objetos de seleção, enfim o seu padrão de interação com os usuários). (3) Outro aspecto importante diz respeito aos campos exibidos, a aplicação terá campos aos quais será responsável (campos que estarão no seu banco de dados) e campos referentes ao fluxo (os que são retornados pela Camada Intermediária). É interessante criar um padrão para visualização destes diferentes campos, isso irá repercutir juntamente na análise das aplicações. Um exemplo pode ser: Para um pedido de aquisição de materiais onde fosse informado o valor e dependendo do valor do pedido o documento seria encaminhado para um setor ou para outro. Neste caso o campo de <valor do pedido> seria um campo com relevância para o fluxo, desta forma deve fazer parte da modelagem do processo. De acordo com o modo como for modelada a aplicação ela pode nem mesmo tomar conhecimento deste campo e ele ser de responsabilidade única do motor, ou então estar em ambos os lados (tanto no motor quanto na aplicação, porém desta forma surge a necessidade de mantê-los sincronizados). (4) Outra verificação necessária para a aplicação será para cada um dos campos que forem retornados pela Camada Intermediária adicionar consistências quanto a sua obrigatoriedade no preenchimento, podendo ser realizada também sua validação de tipo. Essas informações são retornadas como atributos para cada

um dos campos, estas informações sobre cada um dos campos são retornadas a partir de uma estrutura a ser definida que contém, para todos os campos, os atributos modelados na etapa de especificação do fluxo;

8. **O usuário realiza alguma de suas possíveis ações:**
   Seja clicando em algum botão, link ou selecionando de uma lista de ações. De acordo com a interface da aplicação, é disponibilizada ao usuário a lista de ações que podem ser executadas;

9. **Execução da ação:**
   No momento em que o usuário selecionar a ação a aplicação irá indicar para a Camada Intermediária que, para aquele documento, uma ação foi tomada. A aplicação irá, através de chamadas à API da Camada Intermediária, informar a chave do documento (instância no motor), o usuário e a ação que foi realizada. As informações de chave e ação estão armazenadas em variáveis controladas pela aplicação, porém não existe o conhecimento, por parte da aplicação, do que elas significam; pois dizem respeito exclusivo para o controle interno do fluxo pelo motor. Com base destes parâmetros será chamada, pela Camada Intermediária, a função responsável pela atualização do estado deste documento no motor, bem como o código associado à execução da ação. Este código está armazenado em algum repositório específico no motor de *workflow*, a execução do código é feita pelo motor; pois o controle da execução e gerência dos eventos para todas as instâncias do processo, sempre está sob comando do ambiente do *workflow*, a Camada Intermediária apenas irá disparar o processo através de requisições. A Camada Intermediária tem conhecimento, em função do mapeamento realizado anteriormente, qual ação deve ser realizada pelo motor de *workflow*, para cada uma das ações disponibilizadas pela Camada Intermediária;

10. **Atualização do banco de dados:**
    A última ação da seqüência será o motor realizar o código associado à ação e atualizar o seu banco de dados. Após este momento o documento já está em sua nova situação. Terá sido modificado seu estado no fluxo envolvido, todos os controles de acesso e de ações são atualizados e o ciclo novamente pode ser acionado.

Podemos ilustrar como ficaria o código das aplicações conforme o diagrama da Figura 48, temos nele representado, um exemplo de como o código envolvido no fluxo dos processos fica pontual no interior das aplicações. Ocasionando assim, uma maior facilidade na identificação dos trechos de código referentes à lógica da aplicação e os referentes à lógica dos processos. Assim conseguimos, além da independência entre as lógicas empregadas, uma clara visão de separação no código das aplicações, facilitando a identificação de erros ou problemas na construção da aplicação. Esta representação é exposta no diagrama apresentado pela Figura 48.

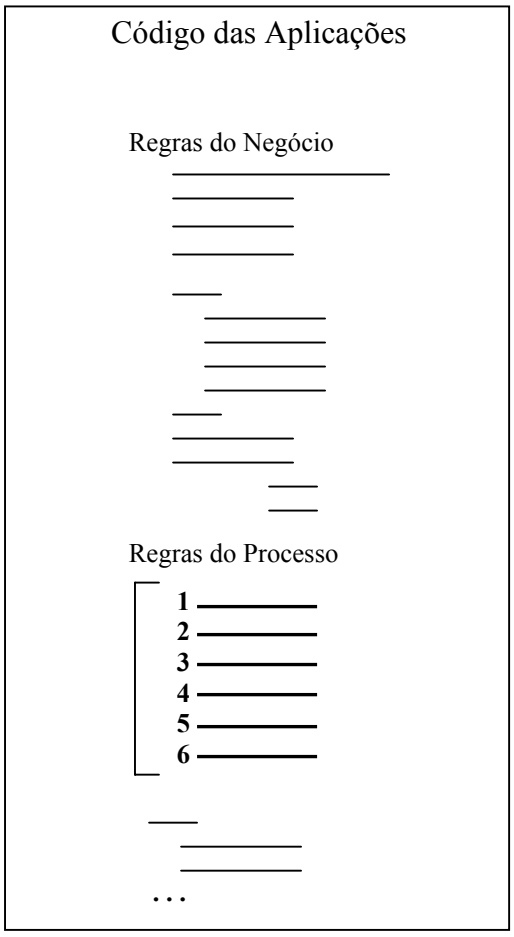


FIGURA 48 - Código das aplicações utilizando a Camada Intermediária

Cabe ser relembrado que as ações descritas anteriormente são apenas uma parte do que se realizará para as aplicações, uma série que questões ainda devem ser discutidas para termos um modelo completo, porém uma vez criado este modelo, todas as aplicações seguirão a mesma estrutura para a utilização dos fluxos.

Deverão ser definidas metodologias para o desenvolvimento das aplicações, de acordo com as tecnologias e infraestruturas já existentes na empresa ou organização. Essa etapa de determinar o modo como serão estruturadas as aplicações é de extrema importância, podendo repercutir tanto em desempenho para as aplicações como também em dificuldades ou facilidades para os desenvolvedores e mesmo para os analistas. Esta repercussão acontece; pois, o modo como forem estruturadas as aplicações, refletirá diretamente sobre o código a ser gerado e, mesmo sobre a modelagem a ser realizada.

No momento em que desenvolvemos as aplicações com este novo enfoque, tendo por trás a infra-estrutura capaz de recuperar as informações do fluxo dos processos, conseguimos alcançar a independência entre as aplicações e os fluxos modelados. Conseguimos atingir a independência; pois, a parte do código das aplicações referente ao fluxo será apenas uma série de etapas e, em função disso, qualquer modificação ocorrida no fluxo não repercutirá no trecho de código referente aos processos.

Esta desvinculação é atingida em sua totalidade, de tal forma que para exemplificar a idéia, podemos utilizar como exemplo, a mesma implementação da interface para uma aplicação e simplesmente referenciar um outro processo. Ao se definir uma interface suficientemente padronizada para os fluxos, se torna possível utilizá-la para mais de um fluxo. Isso foi demonstrado pelo protótipo implementado, onde temos mais de um processo sendo utilizado pela mesma aplicação. Esta flexibilidade fica demonstrada no início da identificação do usuário, onde é selecionado o processo ao qual será realizado o ser trabalho.

No caso do protótipo implementado, salvas as devidas limitações em função de ser apenas um protótipo, foi desenvolvida uma interface de forma que no início da interação se informe qual o processo será utilizado. Este é um caso extremo, onde não somente modificações teriam sido realizadas, mas sim, uma modificação total da modelagem, uma vez que estamos selecionando um novo processo, e mesmo assim para o protótipo esta mudança aconteceu de forma transparente; ou seja, sem nenhuma necessidade de re-compilação ou alteração de código.

Temos, nas Figuras 49 a 53, a representação a partir de algumas telas desenvolvidas, da interface utilizada para a implementação do protótipo:

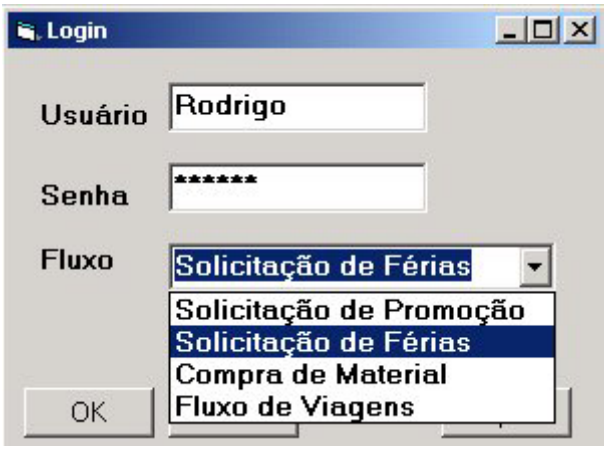


FIGURA 49 - Tela inicial do protótipo

Na Figura 49 temos a identificação do usuário e, juntamente, a escolha do processo ao qual irá interagir, a aplicação mantém sempre a sua interface, modificando apenas informações referentes ao processo que for selecionado. Com este exemplo demonstra-se a flexibilidade que podemos ter utilizando este modelo, pois estamos dinamicamente especificando qual o processo a aplicação irá estar baseada. Para este caso, poderíamos escolher qualquer um dos quatro processos: Solicitação de Promoção, Solicitação de Férias, Compra de Material ou Fluxo de Viagens.

A próxima etapa, após a identificação e especificação do processo utilizado é a tela principal do sistema, para o padrão utilizado foi implementada uma tela inicial onde todos os estados são dispostos ao usuário. Deste modo, fica a cargo do usuário a seleção de um estado ou situação específica, à qual deseje realizar a manipulação aos documentos. Temos as próximas imagens representando dois processos modelados: Solicitação de Promoção e Fluxo de Viagens, cada um com suas respectivas situações.

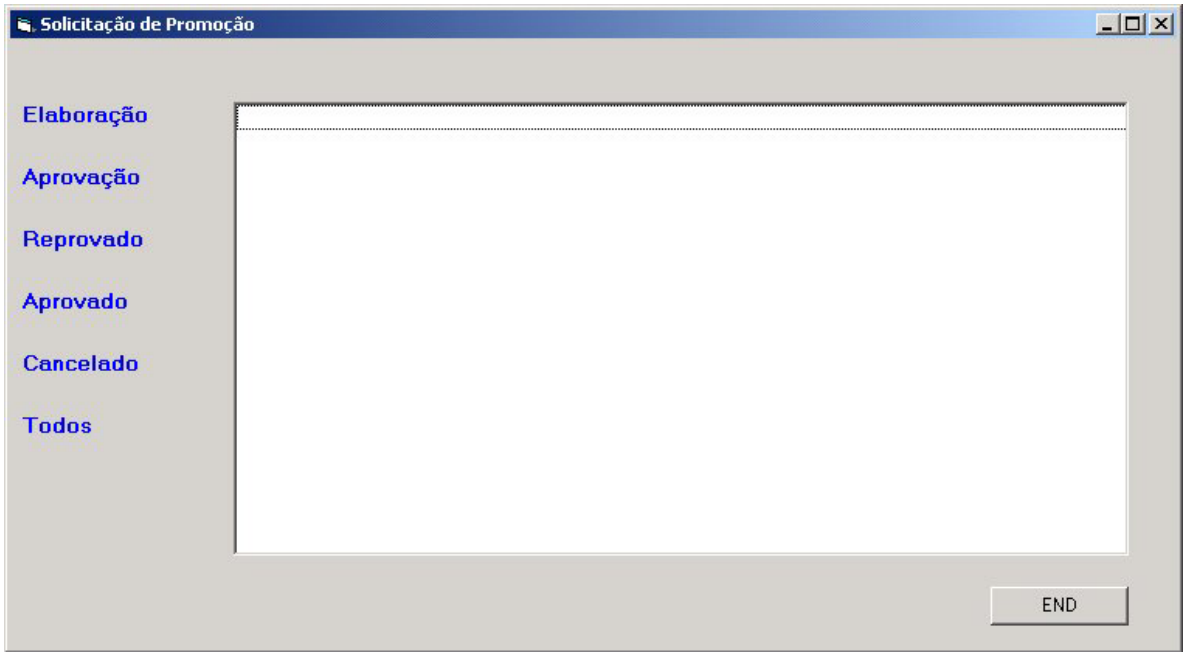


FIGURA 50 - Tela principal para o fluxo de solicitação de promoção

Ao selecionarmos o fluxo de Solicitação de Promoção, como mostrado na Figura 50, são retornados todos os estados existentes para este processo, neste caso temos 5 estados (Elaboração, Aprovação, Reprovado, Aprovado e Cancelado) e uma opção em que todos os documentos, independente de estado, são exibidos. A informação sobre cada um dos estados foi montada dinamicamente, a partir, de requisições a informações do fluxo, neste caso o fluxo de Solicitação de Promoção.

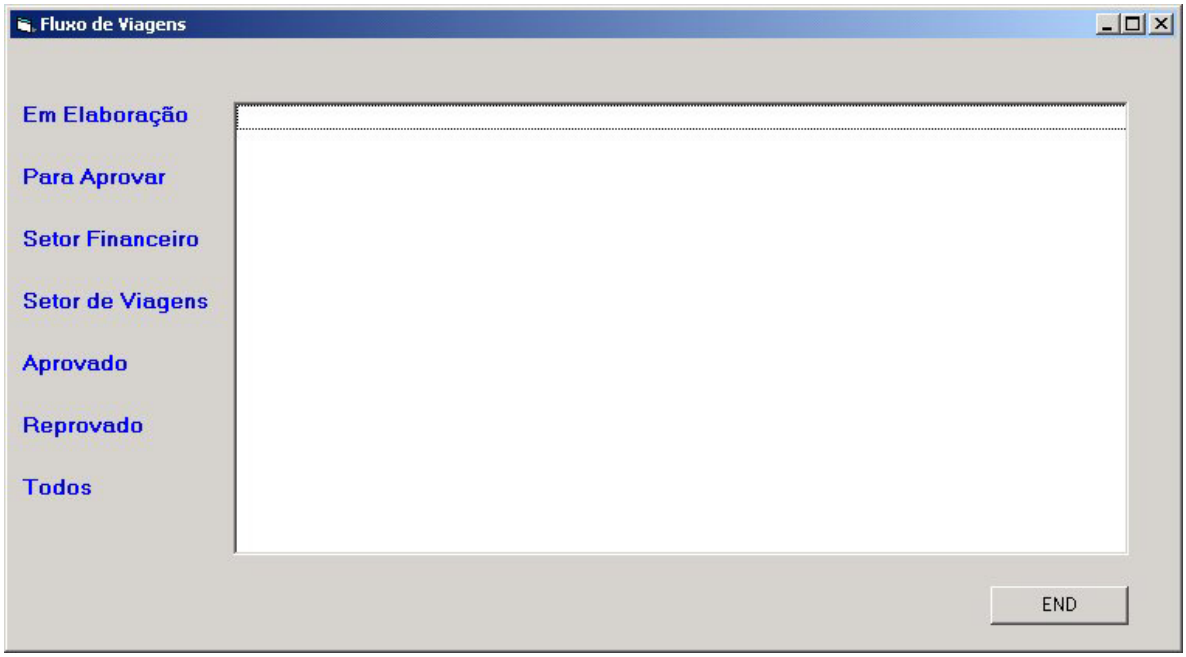


FIGURA 51 - Tela principal para o fluxo de viagens

Ao selecionarmos o fluxo de Viagens, como mostrado na Figura 51, são retornados todos os estados existentes para este processo, neste caso temos 6 estados

(Em Elaboração, Para Aprovar, Setor Financeiro, Setor de Viagens, Aprovado e Reprovado) e uma opção em que todos os documentos, independente de estado, são exibidos. Temos aqui a diferença, se compararmos a Figura 50, entre os dois processos utilizados onde tínhamos outros possíveis estados para os documentos.

As próximas telas do sistema, assim como as iniciais, serão definidas em tempo de execução de acordo com as definições retornadas pela Camada Intermediária. Para as demais interações realizadas pelo usuário seguiremos com a utilização do processo de Fluxo de Viagens. O próximo passo será informar, a partir de qual dos estados, se deseja manipular os documentos.

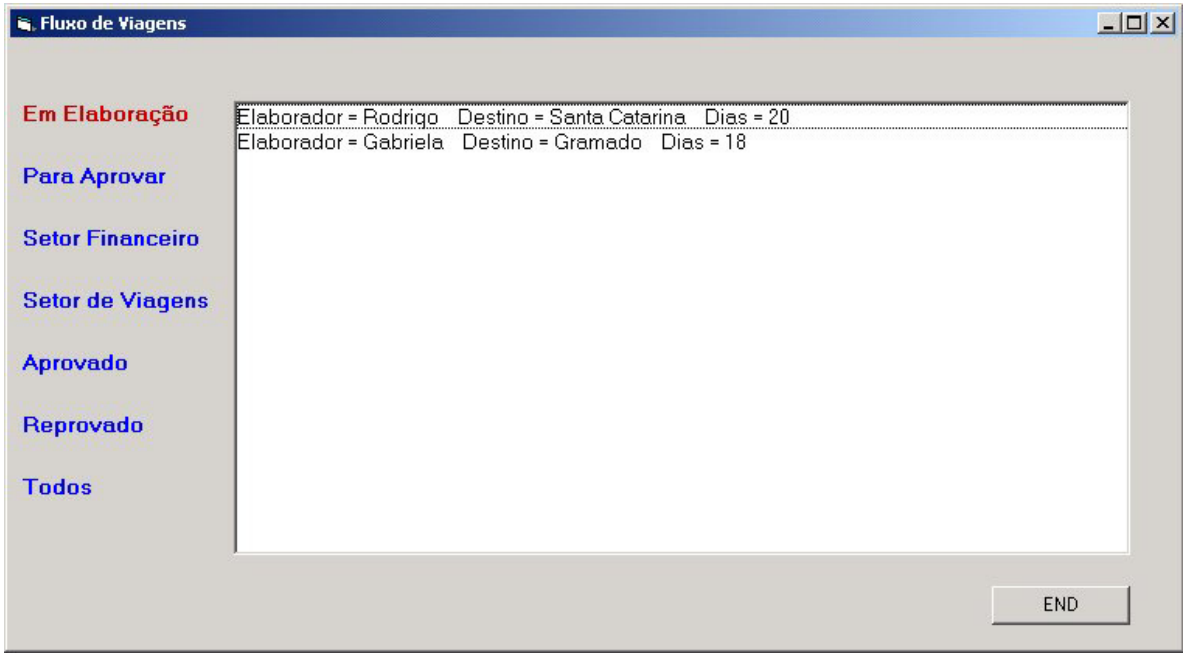


FIGURA 52 - Visualização dos documentos no estado de “Em Elaboração”

Conforme a Figura 52, foi selecionada a situação de “Em Elaboração” e todos os documentos que estavam nesta situação foram exibidos, neste caso temos apenas dois documentos neste estado. Nesta tela temos então, todos os modos de visualização disponibilizados aos usuários. Para o padrão adotado, foi seguida a seguinte regra, disponibilizar visões para cada um dos estados possíveis dos documentos e uma visão final que reunisse todos os documentos, independente de situação. Porém, este padrão pode se adequar de acordo com os padrões definidos para o desenvolvimento das aplicações. Poderíamos definir, por exemplo, campos em que fossem informados atributos específicos para uma consulta mais detalhada, retornando assim os documentos que satisfizessem somente aos critérios informados pelos usuários.

A próxima etapa, após selecionar a visão e receber a totalização dos documentos, consiste na interação com um documento específico. Neste momento todas as informações referentes ao documento que se está trabalhando serão dispostas. Parte das informações contidas nos documentos é referente à própria lógica de negócio, e parte das informações é de domínio do motor de *workflow*. As informações que são de controle das aplicações são disposta diretamente pela aplicação, e as demais, informações sobre o fluxo, serão requisitadas ao motor, através da Camada

Intermediária. Na tela representada na Figura 53 é exposta a interface à qual os usuários utilizarão, para manipular os documentos.

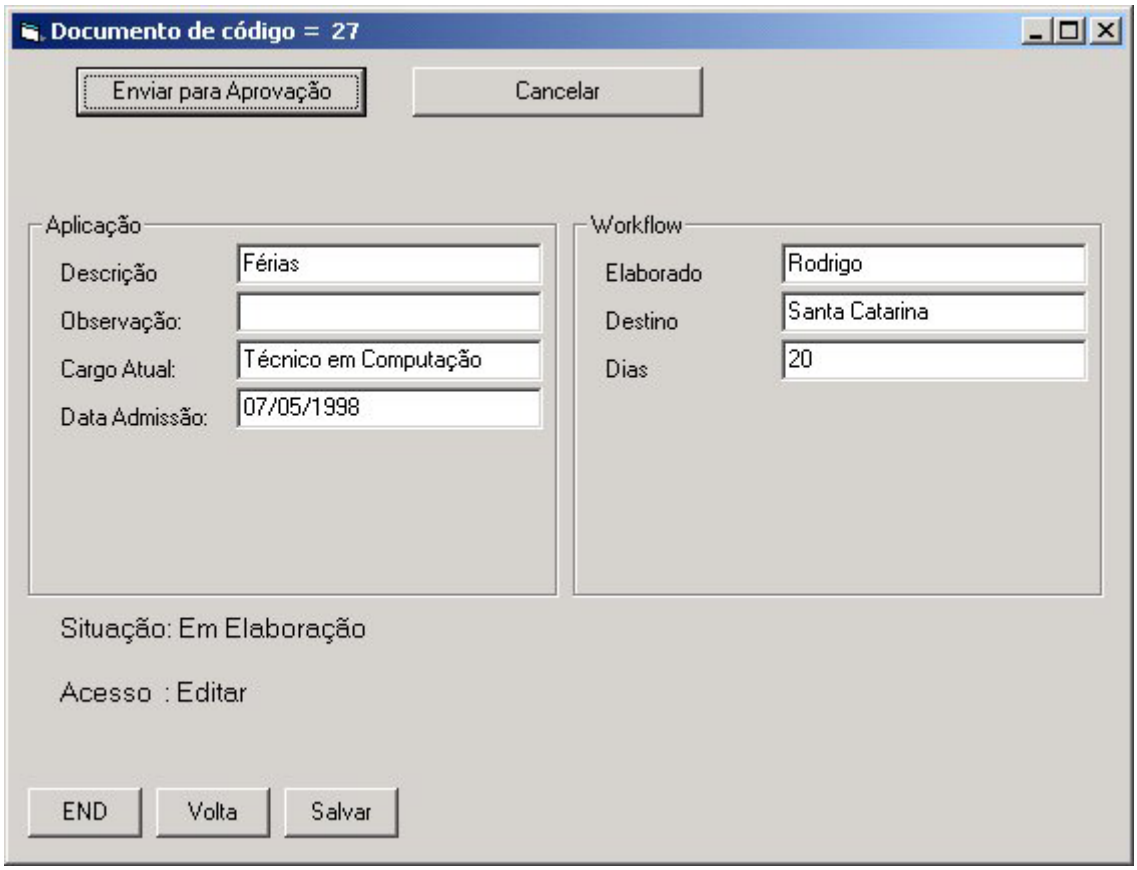


FIGURA 53 - Visualização de um do documento específico

Podemos ver, através da Figura 53, que todos os controles, informações, ações e elementos estão dispostos para o documento. A aplicação não teve necessidade alguma de conhecimento sobre o processo à qual é baseada.

Os elementos, para este padrão de interface, foram dispostos do seguinte modo:

- **Ações:**
  As ações foram dispostas na forma de botões colocados na parte superior do formulário, todos os botões representam as ações que podem ser realizadas, pelo usuário de sessão, para o documento específico. Para a disponibilidade das ações são levados em consideração: o estado do documento, o usuário com suas regras e atribuições e demais controles e definições do fluxo;

- **Campos referentes à regra do negócio:**
  Foram separados os campos que contém informações referentes à regra de negócio e os campos que dizem respeito exclusivo ao fluxo. Os campos com as informações sobre as regras do negócio foram agrupados no lado esquerdo do formulário. Esta disposição facilita o entendimento sobre a divisão entre informações sobre a aplicação e sobre o processo. Outra facilidade é no

momento da montagem dinâmica do formulário, é bastante interessante que tenhamos esta separação formal entre os dois contextos;

- **Campos referentes ao fluxo:**
  Assim como os campos referentes às regras, os campos referentes ao fluxo foram agrupados. Estão agrupados à direita do formulário e irão conter valores e informações de importância para o trâmite dos documentos pelo processo;

- **Informações sobre o estado e acesso:**
  Foram dispostas de forma ilustrativa a situação ou estado do documento, e o controle de acesso atribuído ao documento. Estas informações estão subentendidas pelo contexto do documento, mas foram repetidas para fins didáticos;

- **Botões de navegação no sistema:**
  São os botões referentes única e exclusivamente às aplicações, neste ponto nada referente ao fluxo irá interferir sobre estes elementos.

Não fez parte deste trabalho definir uma interface que melhor se adequasse às necessidades encontradas em sistemas baseados em *workflow*, ou mesmo definir padrões para a utilização da Camada Intermediária. O escopo do protótipo é ilustrar a viabilidade de desenvolver sistemas baseados nesta arquitetura de forma simples e clara. Assim sendo, não se deve avaliar o protótipo como uma solução final ou modelo completo de aplicação para o contexto apresentado.

Esta é a realidade das aplicações dentro do contexto da arquitetura baseada em uma camada responsável em gerar subsídios para permitir a total independência entre os processos modelados e as aplicações por ela servidas.

8 Conclusões

Uma vez tendo, por parte dos fabricantes de ferramentas de *workflow*, o principal enfoque, baseado na modelagem dos processos, e em um segundo plano o desenvolvimento das aplicações, torna-se complexo o desenvolvimento das aplicações; pois, a utilização destes processos modelados se dá de forma desestruturada e não padronizada.

Em função desta carência, não fornecida pela infraestrutura dos ambientes de *workflow*, se faz necessária a utilização de uma camada capaz de gerar subsídios para o desenvolvimento menos complexo e mais produtivo das aplicações. A utilização da Camada Intermediária vem contribuir para o modelo de desenvolvimento das aplicações de forma a tornar o seu desenvolvimento procedimentos simples e padronizados; pois, parte do princípio que toda a lógica dos processos deve estar modelada no próprio ambiente do *workflow*, e em momento algum sob responsabilidade das aplicações.

A utilização da Camada Intermediária, além das vantagens quanto ao desenvolvimento, agrega às aplicações, uma clara separação entre a lógica empregada sobre o fluxo dos processos e a lógica empregada nas regras de negócio. Com esta separação formalizada, a modelagem das aplicações e processos pode ser feita de forma separada, deixando cada etapa de modelagem com um escopo específico.

O enfoque das aplicações será totalmente diferenciado do modo como é realizado atualmente; pois as aplicações serão baseadas em seqüências de procedimentos a serem seguidos. Desta forma, a sua manutenção, em caso de modificações nos processos, se dará de forma transparente, não sendo necessária qualquer modificação em seu código inicial. Este é um dos grandes benefícios agregados com a utilização da Camada Intermediária proposta.

No próximo capítulo serão abordados trabalhos futuros, que surgem a partir da abordagem realizada neste estudo.

9 Trabalhos Futuros

Ao iniciar este estudo foi necessário definir o seu escopo, algumas das idéias iniciais não puderam fazer parte do estudo, por correr o risco de ampliar o trabalho de tal forma que fosse inviável realiza-lo de forma completa. Em função disso, algumas idéias iniciais e também novas visões, originadas no decorrer do estudo, podem ser citadas para trabalhos futuros.

Neste capítulo serão apresentadas algumas propostas que podem ser fruto de um novo trabalho, fazendo assim a complementação, da idéia inicial ou mesmo ampliando a abrangência da proposta apresentada. São propostos dois novos enfoques, que vêm complementar este estudo: Suporte a Modelagem dos Fluxos (CASE) e Padronização visando integração entre motores.

Nas próximas sessões cada um deles será abordado, fazendo uma análise sobre o contexto destes futuros trabalhos, sua abrangência e escopo a se definir.

9.1 Suporte a Modelagem dos Fluxos (CASE)

Como citado anteriormente, uma próxima etapa da Camada Intermediária seria o desenvolvimento de um módulo responsável pela definição e modelagem dos fluxos. Este módulo inclusive chegou a ser cogitado sobre o escopo deste trabalho; porém, como poderia se extender demais a análise a ser analisada, foi decidido citar como trabalho futuro e não incluí-lo no trabalho realizado. Apesar de não ter sido o escopo do trabalho, algumas idéias surgiram ao longo do seu desenvolvimento, e estas idéias poderiam ser utilizadas como fonte inicial de um estudo que viesse a ser baseado nesta proposta.

Temos então, como premissas iniciais alguns tópicos, para o desenvolvimento do módulo de modelagem de *workflow*: Estudo dos padrões utilizados para modelagem de fluxos, definição de uma metodologia de modelagem, adequação aos conceitos utilizados pela Camada de Independência e implementação da interface de utilização. Cada um destes tópicos será brevemente comentado a seguir.

9.1.1 Estudo dos padrões utilizados para modelagem de fluxos

Devem ser analisadas as formas existentes de modelagem de fluxos, seja a partir dos ambientes existentes de *workflow*, seja a partir de padrões de modelagem, como por exemplo, especificações baseadas em UML ou mesmo o modelo de Casati/Ceri [CAS 95], para modelagem de fluxos.

Este estudo deve ser bem criterioso e extenso, em função da quantidade de propostas existentes, principalmente quando se está analisando a modelagem utilizada pelas ferramentas de *workflow* existentes no mercado.

Uma proposta interessante neste estudo seria o levantamento das melhores características de cada modelo analisado, de modo a poder gerar um modelo que agregue as melhores qualidades de cada um dos modelos.

9.1.2 Definição de uma metodologia de modelagem

A princípio se poderia adotar duas maneiras de desenvolver a metodologia e, conseqüentemente, a implementação da ferramenta de modelagem. Uma possibilidade seria não desenvolver um novo modelo e adotar o próprio modelo utilizado pelo motor de *workflow*, ao qual estejamos baseados; a outra possibilidade seria estruturar uma nova modelagem para os processos, deste modo não ficando fixado a nenhum modelo de representação existente.

No primeiro caso, o de adotarmos um modelo já existente, teríamos a necessidade de se extender sua estrutura; pois, o principal problema encontrado nos modelos estudados é não dispormos de estruturas capazes de comportar todo o conhecimento que necessitamos armazenar, a respeito dos fluxos. Sendo assim ou seria implementado um módulo separado da ferramenta disponível pelo ambiente de *workflow*, fazendo a complementação dos processos em um segundo momento (após a modelagem realizada na ferramenta do ambiente), ou então uma integração completa entre a ferramenta existente, de modo a conseguir o funcionamento de forma transparente para os analistas e desenvolvedores do fluxo, tanto da ferramenta CASE do motor quanto da extensão desenvolvida.

No segundo caso, o de desenvolvermos uma nova modelagem, temos a vantagem de não existir dependência de um ambiente específico de *workflow*, e em função disso não agregar qualquer tipo de limitação existente no modelo ao qual viéssemos a seguir. Esta proposta para trabalhos futuros está baseada neste segundo caso, onde o objetivo é desenvolver um novo módulo para a criação das representações dos processos a serem utilizados.

O principal objetivo deste módulo é completar as informações que não estão inseridas na ferramenta de modelagem do motor. No momento em que conseguimos colocar todas as informações referentes aos fluxos, teremos um diferencial das ferramentas de modelagem existentes. Nenhuma informação poderá deixar de ser inserida; pois, caso isso ocorra se estará repassando a lógica dos processos novamente para as aplicações, acabando assim com a independência. Dentre algumas das informações necessárias para termos este modelo completo, em suas especificações, podemos citar:

- **As situações ou estados do processo:**
  A modelagem está orientada a estados, e deve-se ter informado todos os estados possíveis aos documentos ou processos, como exemplo: EM ELABORAÇÃO, PARA APROVAR, CANCELADOS, CONCLUÍDOS;

- **Cadastro de usuários:**
  Deve-se criar para o ambiente uma forma de representar os usuários, ou então realizar a integração com o cadastro de usuários que se estiver utilizando para o ambiente de desenvolvimento;

- **Estrutura de grupos existentes:**
  Uma estrutura de grupos é fundamental, pois em muitos casos os papéis, exercidos no fluxo, são associados a grupos de usuários. Geralmente os grupos de usuários estão interligados com o modo como os usuários estão cadastrados no ambiente, utilizando-se assim, a mesma estrutura;

- **Papéis:**
  Os papéis podem e devem ser associadas a grupos de usuários; caso a empresa possua alguma estrutura que represente a hierarquia e os papéis de cada pessoa, é interessante que a ferramenta de modelagem consiga integrar-se com esta estrutura a fim de reconhecer os usuários que estão associadas a cada grupo;

- **Controle de Acesso:**
  Associar para cada etapa do fluxo as regras/papéis (ou mesmo diretamente quais usuários) com suas respectivas ações (ações no nível alto, genéricas). As ações são referentes à aplicação e não a algum documento específico. Para fins de exemplo podemos citar ações como Criar, Excluir, Alterar. Estas ações não são vistas como ações do fluxo, mas como ações dos sistemas, com um enfoque mais transacional;

- **Campos e variáveis:**
  Modelagem de todos os campos (ou informações) relevantes para o fluxo, juntamente a isso, associar quais destas informações podem ser alteradas, em que momento e por quem (leitura e escrita dos campos). Os campos armazenarão as informações referentes ao fluxo, ou seja, dados que terão relevância no momento de conduzir os documentos na tramitação do seu fluxo;

- **Ações:**
  Cadastrar as ações do fluxo, ações que modificam o status do documento, a princípio estas ações estarão relacionadas com a troca de uma situação à outra no fluxo do documento. Deverão constar todas as ações que podem ser executadas pelos usuários, devendo ser informadas as demais condições para que os mesmos tenham permissão de realização de tal ação; estas condições podem envolver valores de campos do fluxo, associações a certas regras ou grupos ou mesmo qualquer tipo de informação referente ao usuário que possa ter acesso a respectiva ação;

- **Associação das ações:**
  Cadastrar as ações que podem ser executadas, para cada usuário. Estas ações devem ser associadas a usuários/grupos/papéis e às ações do fluxo (Aprovar, Liberar, Cancelar, Recuperar). Deve ser permitido colocar condições para a permissão ou não, para cada uma das ações.

Um exemplo de modelagem realizada, esboçada de modo básico, poderia ser apresentado como na Figura 54:

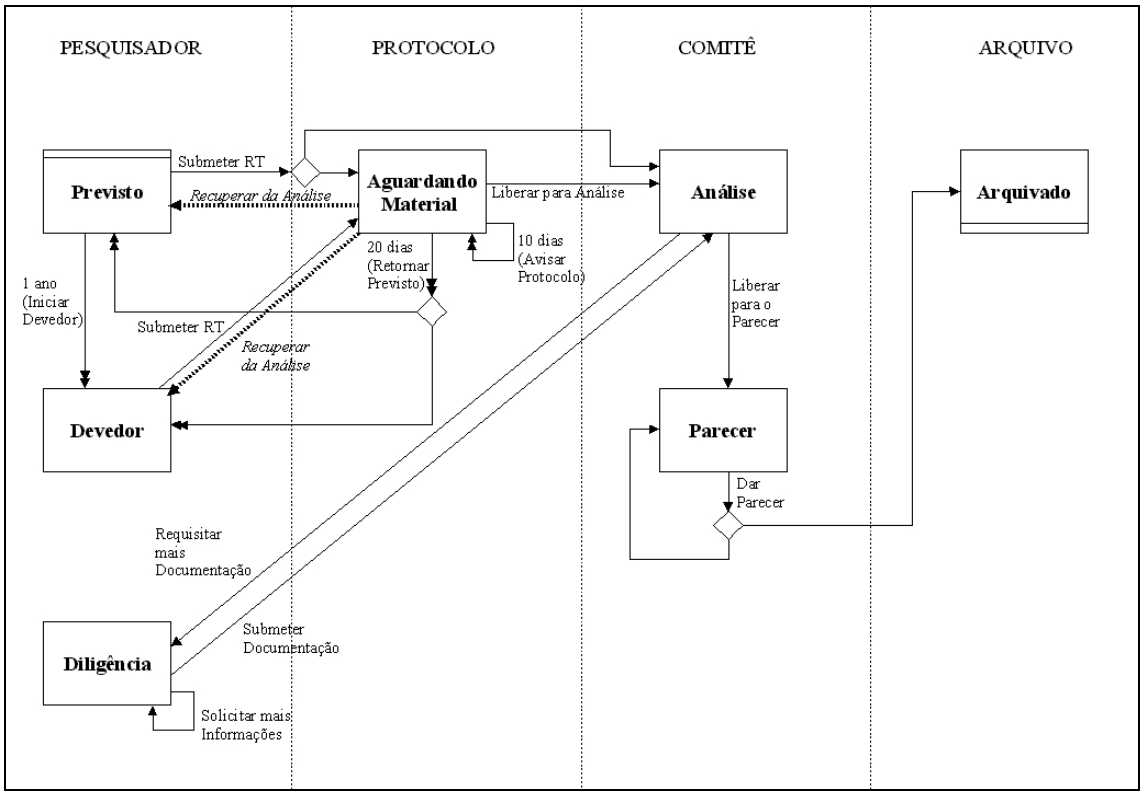


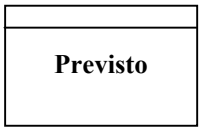
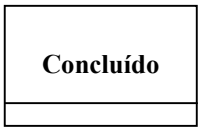
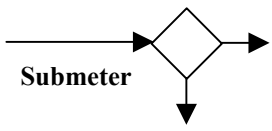
FIGURA 54 - Exemplo de modelagem de fluxo

Na Figura 54 temos um diagrama simplificado de um modelo de representação de fluxos. Temos nele representado um processo em que foram utilizados sete elementos, cada um destes sete elementos é descrito a seguir. Para cada um destes elementos existirão validações, regras de disparo, controles de acesso, código para alterações, limites de tempo, enfim, todos os controles necessários não somente para a modelagem do processo, mas também para sua realização e controle.

Como o modelo proposto não tem como objetivo ser um modelo completo, mas sim uma idéia inicial de uma representação completa, alguns aspectos não foram contemplados. Poderiam ser aqui reportados vários trabalhos que têm como exclusivo foco a modelagem de processos em *workflow*; vários destes trabalhos propõem melhorias e modificações sobre alguns modelos já conhecidos para modelagem de processos. Poderiam ser citados dois modelos clássicos que serviram de base para muitos trabalhos, que são: o modelo de gatilhos e o modelo de Casati/Ceri [CAS 95].

O diagrama apresentado está representado de acordo com os elementos expostos na Tabela 3, onde são abordadas: as características e significados para cada um dos elementos sugeridos nesta modelagem:

TABELA 3 - Representação dos símbolos da modelagem proposta

Representação	Significado
	O retângulo indica cada um dos estados existentes para o processo. A figura do retângulo com uma faixa na parte superior representa o estado inicial do fluxo, onde deve ser iniciado o fluxo, o estado ou situação inicial. Dentro do retângulo é informado o nome do estado, neste caso: "Previsto". Deve haver somente um estado inicial para o processo.
	O retângulo com a faixa no lado inferior está indica o estado final do fluxo. Pode haver vários estados finais para o processo modelado
	O retângulo sem nenhuma indicação indica um estado ou situação intermediária do fluxo, não sendo o estado inicial ou o final. Podendo existir quantos forem necessários, eles estão interligados por setas que indicam as ações.
	A seta simples informa a ação que será tomada para a troca de estados. O sentido da seta indica o estado que o documento estará, depois de realizada a ação, e a origem da seta indica qual o estado anterior à execução da ação. Abaixo da seta é indicado o seu nome, o qual deverá ser também sua referência nas aplicações.
	A seta com a ponta dupla representa uma ação que é executada em função de algum tempo limite. Não é uma ação executada pelos usuários, é informado o tempo (abaixo da seta) e, ao alcançar o tempo limite a ação é executada. O tempo limite refere-se ao tempo do documento sem ação sobre o estado ao qual a seta tem sua origem.
	A seta tracejada indica uma ação de Recuperação de documentos, esta ação se difere das demais; pois ela é executada pelo usuário que não mais detém o documento. O objetivo desta ação é o de representar as situações onde o documento necessita ser recuperado pelos usuários que já o enviaram para a etapa seguinte.
	A seta dividida com um losango indica uma ação que dependerá de uma condição para seguir um dos caminhos, podendo ser dividida em mais de duas opções.

Temos as situações ou estados possíveis, que, para este processo são os seguintes: Previsto, Aguardando Material, Devedor, Diligência, Análise, Parecer, Arquivado. Para cada ação será informado um conjunto de características destes elementos; porém, para fins de ilustração deste modelo não foi relevante mencioná-los.

Entre cada estado teremos as ações que realizam a troca de um estado para outro, novamente temos para cada ação um conjunto de informações ou características a serem inseridas. Um exemplo seria que na situação de “Análise” poderíamos executar a ação “Liberar para o Parecer” - o documento iria para a situação de “Parecer”, ou então executando a ação “Requisitar mais Documentação” - direcionando o documento para a situação de “Diligência”. Para um destes exemplos serão executados os códigos relacionados a cada uma das ações e serão atualizados os controles de acesso.

9.1.3 Adequação aos conceitos utilizados pela Camada de Independência

Independentemente da forma como estiver baseada a modelagem para os processos, deverá ser estruturado um mapeamento para os conceitos utilizados pela Camada de Independência, uma vez que o armazenamento se dará utilizando as suas estruturas e conceitos. Para o exemplo de modelagem citado anteriormente utilizamos uma representação semelhante ao modo como são organizadas as estruturas de armazenamentos – tabelas e relacionamentos – na Camada Intermediária.

No modo como foram estruturadas as tabelas da Camada Intermediária, temos um enfoque, sobre a modelagem dos processos, orientado a estados. Nada impede que a metodologia adotada parta de outro princípio, como uma orientação a processos, porém deve-se ter clara a necessidade de criarmos uma relação entre ambos os conceitos.

9.1.4 Implementação da interface de utilização

Após todas as etapas definidas, o desenvolvimento do módulo pode ser realizado. Este módulo deve ter uma interface completa e de fácil compreensão, de modo a disponibilizar todos os recursos existentes no motor. Desta forma, permitindo a modelagem completa de todos os elementos, fazendo com que não seja necessária qualquer complementação das informações.

9.2 Padronização visando integração entre motores

No momento em que temos fixado, como modelo para o armazenamento das informações dos fluxos, a estrutura proposta pela Camada Intermediária, podemos realizar uma abordagem visando a integração entre motores de diferentes fabricantes e com diferentes conceitos. Esta integração se torna possível no momento em que dispormos, através da Camada de Integração, de um mapeamento para os diversos motores definidos como participantes desta comunicação.

Partindo do princípio que o objetivo da Camada de Integração é possibilitar, para um motor específico, a comunicação entre a Camada de Independência e motor de *workflow*, por que não investir um maior esforço na Camada de Integração, visando a comunicação com mais de um motor de *workflow*? Uma vez estabelecido este mapeamento, para mais de um motor, poderemos ter a comunicação entre os motores através da Camada Intermediária, subsidiada pela Camada de Integração.

Uma vez tendo a Camada de Independência responsável pela especificação dos processos, segue indiferente para o restante do ambiente qual o motor estará realizando a execução e o controle dos eventos, uma vez que, para a Camada de Independência ocorrem de forma transparente, pois, a execução é transferida para a Camada de Integração.

Ao realizarmos esta integração, estaremos resolvendo um dos grandes problemas existentes para os ambientes de *workflow* existentes – a impossibilidade de se realizar tarefas e procedimentos sobre diferentes plataformas de *workflow*. Neste momento não mais serão problema, processos em diferentes ambientes, realizando suas tarefas conjuntamente.

Anexo API's da Camada de Independência

Classe:
Acao

Propriedades:
Private Nome As String
Private Condicao As String
Private Comentário As String
Private Código As Integer
Private CodFluxo As Integer
Private Arquivo As String
Private DocAcao As Documento

Métodos:
Public Function Carrega(ID As Integer, auxDoc As Documento) As Boolean
Public Function Consiste() As Boolean
Public Function Executa() As Boolean
Public Function getNome() As String
Public Function getCondicao() As String
Public Function getComentario() As String
Public Function getCodigo() As Integer
Public Function getCodFluxo() As Integer
Public Function getArquivo() As String
Public Function setNome(pNome As String)
Public Function setCondicao(pCondicao As String)
Public Function setComentario(pComentario As String)
Public Function setCodigo(pCodigo As Integer)
Public Function setCodFluxo(pCodFluxo As Integer)
Public Function setArquivo(pArquivo As String)

Classe:
Campo

Propriedades:
Private Nome As String
Private Comentário As String
Private Tipo As Integer
Private Valor As String
Private Editável As Boolean
Private Obrigatório As Boolean
Private Visível As Boolean
Private Código As Integer
Private ValorDefault As String
Private CodFluxo As Integer
Private Documento As Documento

Métodos:
Public Function Carrega(ID As Integer, auxDoc As Documento) As Boolean
Public Function Consiste() As Boolean
Public Function getCodFluxo() As Integer
Public Function getNome() As String
Public Function getComentario() As String
Public Function getTipo() As Integer
Public Function getValor() As String
Public Function getEditavel() As Boolean
Public Function getObrigatorio() As Boolean
Public Function getVisivel() As Boolean
Public Function getCodigo() As Integer
Public Function getValorDefault() As String
Public Function setCodFluxo(pCodFluxo As Integer)
Public Function setNome(pNome As String)
Public Function setComentario(pComentario As String)
Public Function setTipo(pTipo As Integer)
Public Function setValor(pValor As String)
Public Function setEditavel(pEditavel As Boolean)
Public Function setObrigatorio(pObrigatorio As Boolean)
Public Function setVisivel(pVisivel As Boolean)
Public Function setCodigo(pCodigo As Integer)
Public Function setValorDefault(pValorD As String)

Classe:
Documento

Propriedades:
Private collCampos As Collection
Private collAcoes As Collection
Private collAcesso As Collection
Private corrEstado As Estado
Private corrUsuario As Usuario
Private Código As Integer
Private CodEstado As Integer
Private CodFluxo As Integer

Métodos:
Public Function Carrega(ID As Integer, pUsuario As Usuario) As Boolean
Public Function Consiste() As Boolean
Public Function getCampo(pOffSet As Integer) As Campo
Public Function getAcao(pOffSet As Integer) As Acao
Public Function getCampos() As Collection
Public Function getAcoes() As Collection
Public Function getAcesso() As Collection
Public Function getEstado() As Estado
Public Function getCodigo() As Integer
Public Function getCodEstado() As Integer
Public Function getCodFluxo() As Integer
Public Function setCodigo(pCodigo As Integer)
Public Function setCodEstado(pCodigo As Integer)
Public Function setCodFluxo(pCodFluxo As Integer)

Classe:
Estado

Propriedades:
Private Código As Integer
Private CodFluxo As Integer
Private Nome As String
Private TempoLimite As Integer
Private CodAcaoLim As Integer
Private TempoAviso As Integer
Private MsgAviso As String
Private Comentário As String

Métodos:
Public Function Carrega(ID As Integer) As Boolean
Public Function Consiste() As Boolean
Public Function getCodigo() As Integer
Public Function getCodFluxo() As Integer
Public Function getNome() As String
Public Function getTempoLimite() As Integer
Public Function getCodAcaoLim() As Integer
Public Function getTempoAviso() As Integer
Public Function getMsgAviso() As String
Public Function getComentario() As String
Public Function setCodigo(pCodigo As Integer)
Public Function setCodFluxo(pCodFluxo As Integer)
Public Function setNome(pNome As String)
Public Function setTempoLimite(pTempoLimite As Integer)
Public Function setCodAcaoLim(pCodAcaoLim As Integer)
Public Function setTempoAviso(pTempoAviso As Integer)
Public Function setMsgAviso(pMsgAviso As String)
Public Function setComentario(pComentario As String)

Classe:
Fluxo

Métodos:
Private UsuarioFlu As Usuario
Private collRegras As Collection
Private collGrupos As Collection
Private collEstados As Collection
Private collRegrasUsu As Collection
Private collGruposUsu As Collection
Private Código As Integer
Private Nome As String
Private Sigla As String
Private Comentário As String
Private FluxoCI As INTEGRA_Fluxo

Propriedades:
Public Function Carrega(pID As Integer, pUsuario As Usuario) As Boolean
Public Function Consiste() As Boolean
Public Function getDocumentosPorEstado(pEstado As CEstado) As Collection
Public Function getDocumentosPendentes() As Collection
Public Function getDocumentos() As Collection
Public Function getDocumento(pID As Integer) As Documento
Public Function NovoDocumento() As Documento
Public Function getCodigo() As Integer
Public Function getNome() As String
Public Function getSigla() As String
Public Function getComentario() As String
Public Function setCodigo(pCodigo As Integer)
Public Function setNome(pNome As String)
Public Function setSigla(pSigla As String)
Public Function setComentario(pComentario As String)
Public Function getRegrasFluxo(pUsuario As Usuario) As Collection
Public Function getGruposFluxo(pUsuario As Usuario) As Collection
Public Function getEstados(pUsuario As Usuario) As Collection
Public Function getUsuario() As Usuario

Classe:
Grupo

Propriedades:
Private Código As Integer
Private Nome As String
Private Comentario As String
Private collMembros As Collection

Métodos:
Public Function Carrega(ID As Integer) As Boolean
Public Function Consiste() As Boolean
Public Function getMembros() As Collection
Public Function getCodigo() As String
Public Function getNome() As String
Public Function getComentario() As String
Public Function setCodigo(pCodigo As String)
Public Function setNome(pNome As String)
Public Function setComentario(pComentario As String)

Classe:
Regra

Propriedades:
Private Nome As String
Private Código As Integer
Private Comentário As String
Private collMembros As Collection

Métodos:
Public Function Carrega(ID As Integer) As Boolean
Public Function Consiste() As Boolean
Public Function getMembros() As Collection
Public Function getNome() As String
Public Function getCodigo() As Integer
Public Function getComentario() As String
Public Function setNome(pNome As String)
Public Function setCodigo(pCodigo As Integer)
Public Function setComentario(pComentario As String)

Classe:
Sessao

Popriedades:
Private UsuarioSess As Usuario
Private FluxoSess As Fluxo

Métodos:
Public Function getFluxo(pFluxo As String) As Fluxo
Public Function Login(pUsuario As String, pSenha As String) As Boolean
Public Function getUsuario() As Usuario

Classe:
Usuario

Propriedades:
Private Código As Integer
Private Nome As String
Private Matricula As String
Private Sexo As Boolean
Private collRegras As Collection
Private collGrupos As Collection

Métodos:
Public Function Carrega(ID As Integer) As Boolean
Public Function Consiste() As Boolean
Public Function getCodigo() As Integer
Public Function getNome() As String
Public Function getMatricula() As String
Public Function getSexo() As Boolean
Public Function setCodigo(pCodigo As Integer)
Public Function setNome(pNome As String)
Public Function setMatricula(pMatricula As String)
Public Function setSexo(pSexo As Boolean)
Public Function getRegras() As Collection
Public Function getGrupos() As Collection

Bibliografia

[AAL 95] AALST, W.M.P. **Modelling and analysing workflow using a Petri-net aprroach**. Eindhoven: Eindhoven University of Technology, 1995. Relatório de Pesquisa.

[BAR 95] BARTHELMES, P.; WAINER, J. Workflow Modelling. In: CYTED-RITOS INTERNATIONAL WORKSHOP ON GROUPWARE, CRIWG, 1., 1995, Lisboa, Portugal. **Proceedings…** Lisboa: Junta Nacional de Investigação Científica e Tecnológica 1995.

[BER 97] BERTOLETTI, Ana Carolina. **SAGRES**: um sistema adaptável de informações e suporte à interação em grupo. 1997. 124f. Dissertação (Mestrado em Ciências da Computação) - Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[BOR 95] BORGES, Marcos Roberto da Silva; CAVALCANTI, Maria Cláudia Reis; CAMPOS, Luiza Machado. **Suporte por computador ao trabalho cooperativo**. Porto Alegre: Instituto de Informática da UFRGS, 1995. Curso ministrado no Congresso da Sociedade Brasileira de Computação, 15.; Jornada de Atualização em Informática, 14., 1995.

[CAS 95] CASATI, F. et al. Conceptual Modeling of Workflows. In: INTERNACIONAL CONFERENCE ON OBJECT-ORIENTED AND ENTITY – RELATIONSHIP MODELING, OOER, 14., 1995, Gold Coast. **Object oriented and entity – relationship modeling**: proceedings. Berlin: Springer-Verlag, 1995. (Lecture Notes in Computer Science, 1021).

[DUI 95] DUITSHOF, Matthijs. **Workflow Automation in Three Administrative Organizations**. 1995. Masther's Thesis. University of Twente/Departament of Computer Science – Section Information Systems, The Netherlands.

[ELL 91] ELLIS, C. A.et al. Groupware: some issues and experiences. **Communications of the ACM**, New York, v. 34, n.1, p. 38-58, Jan. 1991.

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

[GRU 94] GRUDIN, Jonathan. Computer Supported Cooperative Work: Its History and Focus. **Computer**, Los Alamitos, v. 27, n. 5, p. 19-26, May 1994.

[GRU 97] GRUDIN, Jonathan. **Computer Supported Cooperative Work**: Its History and Participation. Irvine: University of California, 1997.

[HIL 97] HILLS, Melaine. **Intranet as Groupware**. USA: Wiley Computer Publishing, 1997.

[JOO 94] JOOSTEN, S. Trigger. Modelling for Workflow Analysis. In: WORKFLOW MANAGEMENT, CHALENGES, PARADIGMS AND PRODUCTS, CON, 1994, Viena, Áustria. **Proceedings...** Viena: Oldenbourg, 1994. p. 236-247.

[MAR 94] MARTIN, James. **Princípios de análise e projeto baseados em objetos**. Rio de Janeiro: Campus, 1994.

[MAT 2000] MATSUDA, Kelcy Mayumi. **Análise de Problemas para a Implantação de Workflow**. 2000. Dissertação (Mestrado em Ciências da Computação) – Instituto de Informática, Universidade de Campinas (UNICAMP), São Paulo.

[MCC 92] McREADY, S. There is more than one kind of workflow software. **Computerworld**, [S.l.], n. 2, p. 151-155, Nov. 1992.

[NIC 96] NICOLAO, Mariano. **Um Estudo sobre Conceituação de Workflow**. 1996. Dissertação (Mestrado em Ciências da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[NUM 91] NUNAMAKER, Jay F. et al. Eletronic Meeting and Decision Rooms. **Communications of the ACM**, New York, v. 34, n.7, July 1991.

[WFM 96] WFMC. WORKFLOW MANAGEMENT COALITION. **Terminology & Glossary**, TC-10110. Bruxelas, 1996. 52p.