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

Modelagem de Workflow utilizando um Modelo de Dados Temporal Orientado a Objetos com Papéis

por

Mariano Nicolao

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

Prof. Dra. Nina Edelweiss
Orientadora

Porto Alegre, novembro de 1998.

Sumário

Lista de Abreviaturas

API	Application Program Interface
CSCW	Computer Cooperative Work
EBNF	Extended Backus Normal Form
HAD	Heterogeneous, Autonomous and Distribuied
SGBD	Sistema Gerenciador de Banco de Dados
TF-ORM	Temporal Functionality in Objects with Roles Model
WFA	Workflow Administrator
WfMC	Workflow Management Coalition
WFMS	Workflow Management System
WPDL	Workflow Process Definition Language

Lista de Figuras

Lista de Tabelas

Resumo

Um dos grandes problemas relacionados a modelagem de *workflow* consiste na utilização de técnicas de modelagem conceitual específicas a cada sistema de *workflow*, não havendo, dessa forma, um modelo aceito consensualmente. Esta situação, decorrência do ambiente competitivo neste mercado, leva à não inclusão de muitas características conceitualmente importantes relacionadas à modelagem nos técnicas geralmente utilizadas. Um importante aspecto a ser tratado nos modelos conceituais é a questão da modelagem formal do *workflow* e que constitui o tema central deste trabalho. Esta dissertação apresenta uma técnica de modelagem de *workflow* utilizando como modelo de dados referencial o TF-ORM (*Temporal Functionality in Objects with Roles Model*). Esta técnica desenvolve uma especificação rigorosa de *workflow* em um nível conceitual, formalizando com a utilização de um modelo único seu comportamento interno (cooperação e interação entre tarefas) e seu relacionamento para o ambiente (designação de tarefas de trabalho para executores). Neste modelo, construções são apresentadas para representar, de forma eficiente, a modularização e o paralelismo. Uma linguagem textual de definição de *workflow* é apresentada. Adicionalmente é apresentada a utilização de descrições formais do *workflow* para gerar o esquema de dados do *workflow* e o conjunto de regras para seu gerenciamento. Em adição, o paradigma de regras oferece um formalismo conveniente para expressar computações reativas influenciadas por eventos externos, gerados fora do WFMS (*Workflow Manager System*).

Finalmente é realizada uma analise sobre algumas ferramentas comerciais, procurando validar a praticidade dos modelos conceituais desenvolvidos. Os principais conceitos envolvidos em *workflow* são descritos e classificados de forma a possibilitar, a validação tanto dos conceitos quanto da modelagem através de um estudo de caso e a utilização de um sistema comercial.

Palavras-Chave: Modelagem de Workflow, Técnica de Modelagem, Modelagem Conceitual, Modelagem Formal de Workflow

Abstract

One of the greatest problems in workflow modelling is the use of specific conceptual modelling techniques associated to each workflow system; there is not a consensual accepted model. This situation, a consequence of the strong competitive environment in this market, leads to the non-inclusion of many important conceptual characteristics. This restriction is a consequence of the restricted modelling techniques closely related with implementation models. An important aspect to be considered, and the central subject of this work, is the formal workflow modelling. A modelling technique using the TF-ORM (Temporary Functionality in Objects with Rolls Model) data model is here presented.

The modelling technique develops a rigorous specification of workflow at the conceptual level, formalising in one model its internal behaviour (the co-operation and interaction among tasks) and its relationship with the environment (the designation of tasks). In this model, constructions where developed to represent, in an efficient form, the modularity and the parallelism of the activities. A formal language for the workflow definition is presented. Additionally, the use of formal workflow description is used to generate the data flow and rules set for its management. In addition, the rules paradigm offers a convenient formalism to express reactive computations influenced by external events generated outside the Workflow Manager System.

Finally a case study is accomplished using some commercial modelling tools, to validate the developed conceptual models practicality.

Keywords: Workflow Modelling, Modelling Technique, Conceptual Modelling, Formal Workflow Modelling

1. Introdução

Sistemas de manufatura e de automação de escritórios são conhecidos desde o início da industrialização e constituem o produto da busca do aumento da eficiência, concentrando-se nas características das atividades e do trabalho que é produzido ao longo do processo [GEO 95]. As mudanças constantes, que vêm exigindo das organizações maior produtividade, menores custos e melhor atendimento aos clientes, criam a necessidade de novas formas de gestão. A percepção comum a praticamente todas as técnicas propostas é de que estruturas organizacionais não podem mais orientar-se pela divisão tradicional em áreas específicas. Ao contrário, a estrutura da organização deve ser baseada em processos que ela realiza [BUR 93, HAM 94, DUI 95, ELL 91,96, NIC 96]. Dessa forma, os processos assumem um papel primordial na empresa e consequentemente, os sistemas de informação capazes de representar estes processos. O principal problema dos sistemas baseados em processos é que não se tem o controle das atividades que estão sendo executadas, tornando-se assim difícil monitorar quando (em que instante de tempo) determinada atividade está sendo executada e quem a está executando, dificultando dessa forma a evolução do fluxo de trabalho. Assim, sistemas de *workflow* aparecem como uma tecnologia capaz de solucionar estes problemas e implementar os conceitos relacionados a interação entre os processos.

Sistemas de *workflow* manipulam e monitoram a informação relativa ao fluxo de trabalho para gerenciar, coordenar e controlar o trabalho mais eficientemente, minimizando o problema da coordenação do trabalho nos processos de negócios [DUI 95]. Dessa maneira estes sistemas vêm ganhando força no mercado. Contudo, considerando que os sistemas de *workflow* são recentes na área de sistemas de informação, estes ainda apresentam questões a serem tratadas como os aspectos relacionados a conceituação, interoperabilidade e modelagem. Os aspectos relacionados a conceituação de *workflow* foram tratados em [NIM 96], a questão da interoperabilidade é tratada pela [WFM 96b] e a questão da modelagem passa ser tratada neste trabalho.

Um dos grandes problemas relacionados a modelagem de *workflow* é o de que praticamente cada sistema de *workflow* utiliza sua própria técnica de modelagem, não havendo dessa forma, um modelo aceito de forma consensual. Esta situação determina que muitas das características relacionadas a modelagem sejam negligenciadas. Um importante aspecto a ser tratado nos modelos é a questão da modelagem formal do *workflow* e que constitui o tema central deste trabalho. O formalismo garante uma interação correta dos participantes do processo, a consistência dos dados e a representação segura do processo. Assim, neste contexto, este trabalho apresenta uma técnica de modelagem de *workflow* utilizando como modelo de dados o TF-ORM (*Temporal Functionality in Objects with Roles Model*) [EDE 93, 94, 94a, OLI 95] como referência, devido ao seu formalismo, as suas características de orientação a objetos, lógica temporal e, também, devido ao significativo conjunto de atividades desenvolvidas com este modelo no CPGCC (Curso de Pós-Graduação em Ciência da Computação).

1.1 Proposta do Trabalho

Este trabalho tem como um de seus objetivos, revisar os artigos existentes e outras bibliografias sobre o tema *workflow*, procurando determinar o que é um *workflow*, suas

características e qual é a definição mais apropriada para este trabalho. Esta atividade é necessária pois este conceito, útil para a modelagem de sistemas de informação, é abordado por diferentes autores de formas complementares.

Por outro lado, como há bastante trabalho comercial desenvolvido na área, entre eles as aplicações de Lotus Notes, Microsoft Exchange, Oracle Office, Jet Form e Process Builder, é importante verificar os produtos existentes e, como *workflow* trata de um domínio aplicado, analisar estes produtos para validar a praticidade dos modelos conceituais desenvolvidos.

Finalmente, os principais conceitos envolvidos em *workflow* devem ser descritos e classificados de forma a possibilitar, posteriormente, a validação tanto dos conceitos quanto da modelagem através de um estudo de caso e a eventual utilização de um sistema comercial. Além disso, de maneira a tornar a modelagem efetiva, é necessário melhorar e fortalecer a especificação do *workflow* em um nível conceitual, formalizando com um modelo único seu comportamento interno (cooperação e interação entre tarefas) e seu relacionamento para o ambiente (designação de tarefas de trabalho para executores).

Assim, este trabalho apresenta uma técnica de modelagem conceitual que é a base para alcançar a especificação e implementação de *workflow*. Neste modelo, construções são apresentadas para obtermos a modularização e o paralelismo eficientemente. Uma linguagem de definição textual de *workflow* é apresentada para permitir a especificação do *workflow*. Ela mostra como descrições formais do *workflow* podem ser usadas como entradas para gerar o esquema de dados do *workflow* e o código de regras para seu gerenciamento. Em adição, o paradigma de regras providencia um formalismo conveniente para expressar computações reativas que normalmente são influenciadas por eventos externos, gerados fora do WFMS (*Workflow Manager System*), tal como exceções ou pré-condições de tarefas [EDJ 95, 98, HEI 98, STI 98].

1.2 Organização do Texto

Neste capítulo foi introduzido o conceito de sistemas de *workflow*, sua aplicação e necessidade nas empresas, bem como os objetivos deste trabalho. O capítulo dois relata de forma sucinta a evolução do *workflow*, tendências futuras, os principais conceitos relacionados e sua classificação em 5 níveis; o capítulo três identifica alguns dos tipos de sistemas de *workflow* existentes, o relacionamento entre estes tipos e o grau de estruturação e de complexidade das tarefas realizadas nos processos de negócios; o capítulo quatro classifica as metodologias de modelagem de *workflow* existentes e descreve algumas das possíveis formas de especificação que os modelos de *workflow* utilizam; no capítulo cinco são apresentados três modelos de *workflow*, a partir destes modelos é feita uma a análise e são determinados um conjunto de critérios básicos necessários à modelagem de *workflow*; o capítulo seis introduz alguns conceitos básicos relacionados ao TF-ORM necessários a compreensão e desenvolvimento deste trabalho; no capítulo sete são tratadas as extensões realizadas no TF-ORM; no capítulo oito, é realizada a representação de um estudo de caso através de ferramentas comerciais de *workflow* analisando e validando a modelagem realizada através do TF-ORM comparando-a com diferentes metodologias, no capítulo nove é realizada uma conparação entre o TF-ORM e as ferramentas comerciais segundo os critérios básicos de modelagem do *workflow*; o capítulo dez apresenta as conclusões e os trabalhos futuros.

2. Workflow

Esta seção relata de forma sucinta a evolução do *wokflow*, as tendências futuras e identifica os principais conceitos relacionados.

2.1 Evolução

A construção da maioria dos sistemas de *workflow* de primeira geração, durante a década de 70, partiu de abstrações que descrevem o processo, a partir do exame de um problema. Estas abstrações eram representadas através de algum formalismo, e este modelo era então interpretado, direcionando a seqüência de disparos de atividades e distribuição de tarefas aos agentes, o que constitui o fluxo de trabalho neste caso. Esta posição foi reforçada pela observação de que, em média, 90% dos processos de uma organização podem ser considerados rotineiros [AMA 93], passíveis a princípio de uma modelagem precisa. Porém, mesmo em processos aparentemente bem comportados, com alto grau de estruturação, um número significativo de situações não previstas, fará com que o modelo se torne inadequado [HIR 86]. Os sistemas de *workflow* pioneiros provaram ser excessivamente inflexíveis [KRE 91], e a maioria dos projetos foram abandonados até a década de 90, que presencia um ressurgimento da pesquisa na área.

2.1.1 Evolução entre 1970 e 1980

A partir de 1970 começaram a ser introduzidos os primeiros sistemas de automação de processos de negócios. Alguns desses sistemas continham especificações complexas das corporações, detalhando o fluxo de trabalho (seqüência de procedimentos que compõem um processo a serem executados de forma seqüencial ou não) e os dados que seriam utilizados neste fluxo. Os anos 70 foram anos de grande otimismo sobre o efeito benéfico em relação à produtividade e à eficiência desta nova tecnologia. Como quase sempre acontece, muito deste otimismo foi infundado. Foi observado que as organizações somente obtinham sucesso se as pessoas que trabalhavam nelas criativamente violassem e enganassem os procedimentos padrões implantados no software (descrições estruturadas das tarefas) para a solução de um determinado processo de negócio. Quando estes coordenadores eletrônicos (*softwares* que auxiliam a coordenação de tarefas simples relacionadas aos processos de negócios) foram introduzidos nas organizações, as pessoas não podiam desobedecer aos procedimentos da organização de forma aberta. Em muitos casos, estes sistemas tornaram as organizações ineficazes; levando à rejeição da tecnologia. Os sistemas rígidos dos anos 70 tinham tendência a interferir mais com as rotinas de trabalho do que auxiliar sua expedição (execução). *Workflow* também não obteve sucesso nos anos 70 por motivos como: a não aceitação social dos computadores nas organizações; a falta de informação dos requisitos e das armadilhas da tecnologia de grupo; e porque a tecnologia relacionada a redes de computadores não estava geralmente disponível, ou seja, não haviam nem profissionais qualificados e nem tecnologia suficiente disponíveis a para automação dos processos de negócios.

2.1.2 Evolução entre 1980 e 1990

A partir de 1980 foi dada maior ênfase à modelagem dos processos de negócios. Um exemplo são as Redes de Controle de Informação [ELL 96], um modelo de atividade organizacional com características similares às redes de Petri coloridas. Este modelo generaliza redes de Petri pela adição de um modelo de fluxo de dados complementar, controlando o fluxo primitivo e simplificando a semântica para que o

modelo seja intuitivo e útil para o projeto de automação de processos de negócios. As redes de Controle de Informação e suas variantes foram usadas tanto no contexto acadêmico quanto no contexto industrial, nos anos 80, para descrever o trabalho estruturado e para adicionar ao estudo das organizações, os processos de análise e reestruturação

Após 1980 mais ênfase foi dada a *groupware* e CSCW (*Computer Support Cooperative Work*). A maioria destes trabalhos tinha como foco central diferentes tipos de *groupwares* organizacionais (agendamento de conferências, correio eletrônico, vídeo conferência, etc.) nos quais *workflow* não estava incluído. De fato, durante este período, houve uma reação negativa sobre a automação dos processos de negócios e sistemas de gerenciamento de *workflow*.

2.1.3 Evolução a partir de 1990

Em 1990 começam a aparecer questões e estudos sobre *workflow* nas conferências de *CSCW*. Pesquisadores preocupam-se em explorar novos modelos acreditando que estes modelos podem ser particularmente úteis na compreensão e evolução de processos complexos. A partir destas pesquisas verificou-se a também a aplicabilidade do *workflow* no processo de reengenharia. Esta utilização provem do fato de que tanto a reengenharia quanto o *workflow* trabalham com a mesma unidade: o processo. Enquanto a reengenharia procura reconstruí-lo da melhor forma possível, o *workflow* busca automatizá-lo, transferindo para o computador uma série de responsabilidades outrora de funcionários, como o trâmite de documentos, transferência de trabalho, coordenação e registro, entre outros.

Observando os sucessos e as falhas no decorrer desta evolução verifica-se a necessidade de pesquisas em sistemas de *workflow* para as próximas gerações. Características importantes a serem analisadas:

- sistemas de *workflow* precisam ser sistemas abertos e precisam ter alta capacidade de interoperabilidade. Estes requerimentos cada vez mais estão sendo requisitados por vendedores e usuários de sistemas de *workflow* [ELL 96];
- o paradigma de *workflow*s assíncronos precisa ser integrado com o paradigma de tempo real de *groupware*, para que sistemas de *workflow* possam conectar grupos distribuídos, para tomada de decisão e solução de problemas [DEH 96] Apud [ELL 96];
- corporações trabalham atualmente de forma ativa com suas subsidiárias, com consultores e competidores. Estas corporações precisam interagir eficientemente e efetivamente enquanto mantém níveis de segurança. Esta visão em larga escala de arquiteturas de *workflows* que suportam atividades integradas entre empresas precisa ser desenvolvida. Um usuário precisa estar habilitado para mover-se de um ambiente de usuário único (*single user*) para um ambiente de grupo de trabalho multi-usuário (*multi-user*) e deste para um ambiente de uma multi-corporação altamente decentralizada, tudo isto de forma fácil;
- muita pesquisa precisa ainda ser feita em sistemas distribuídos, modelagem social e organizacional, níveis de segurança em protocolos de rede e tecnologia em larga escala em orientação a objetos distribuídos, antes das visões acima serem realizadas.

2.2 Conceitos relacionados a *workflow*

Vários conceitos são atribuídos a *workflow*, tanto em nível acadêmico como no nível comercial. Alguns desses conceitos podem ser encontrados em [BUR 93, SAI 94, DUI 95, GEO 95, JOO 95a,95b, MCK 94, MIL 95, SHE 95,97, WEI 95, WFM 96]. Em geral todos contemplam a idéia de que *workflow* é direcionado a processos de negócios.

Segundo a *Workflow Management Coalition*[1] (WfMC) *workflow* consiste da automação de um processo de negócios, na sua totalidade ou parte, durante o qual documentos, informações ou tarefas são passadas de um participante para outro por ações, de acordo com um conjunto de regras procedurais [WFM 96].

2.2.1 Atividade

Atividade é um conjunto de eventos que ocorrem sobre a responsabilidade de um ator. Esta definição permite que uma atividade seja realizada por muitas pessoas, contanto que um ator seja responsável pela atividade. Por exemplo, enviar uma carta pode envolver secretárias, serviços de entrega, etc., mas isto é considerado uma atividade quando aqueles atos são executados sobre a responsabilidade do remetente.

Uma atividade requer recursos humanos ou automatizados para suportar a execução do processo. Sempre que um recurso humano é necessário, uma atividade é alocada para um participante do *workflow*.

Dois tipos de atividades podem fazer parte de um *workflow*:

- atividades automatizadas – capazes de serem automatizadas por meio de um computador, usando um sistema de gerenciamento de *workflow* para gerenciar a sua execução.
- atividades manuais – capazes de automação e saem do escopo do sistema de gerenciamento do *workflow*. Tais atividades podem ser incluídas na definição do processo, por exemplo para suportar a modelagem do processo, mas não fazem parte do resultado de um *workflow*.

Uma atividade pode ser completamente automatizada ou pode ser designada para um agente. Se a tarefa de trabalho é completamente automatizada, isto é um sistema externo similar a um *software* ou uma máquina executa a tarefa de trabalho, então esta característica precisa ser especificada adicionando a qualificação automática para o nome da tarefa de trabalho. Se ao invés disso a tarefa for manual, então ela tem que ser designada para um agente.

A determinação de uma tarefa de trabalho para um agente pode ser pré definida na descrição do *workflow*, ou pode ser selecionada pelo WFMS a partir de uma lista definida com base na carga de trabalho dos agentes.

2.2.2 Sincronismo

Esta seção apresenta formas de sincronismo existente entre atividades conforme [JOO 95, CAS 95].

O fluxo de controle permite a sincronização do processo de execução de atividades. A interação dessas atividades através do fluxo de controle é representada por conexões. Dessa forma, a possibilidade de representação formal destas conexões é um

---

[1] A *Workflow Management Coalition* é uma entidade que tem como objetivo aumentar a utilização das tecnologias de *workflow* através do desenvolvimento de terminologias comuns e padrões.

fator fundamental que deve estar presente em um modelo de *workflow*. Há muitas conexões possíveis entre atividades; a seguir serão apresentadas, das possíveis conexões existentes, as que basicamente devem compor as atividades de um *workflow*.

- **Conexão direta entre atividades**

  Conexão direta entre duas atividades A e B significa que B inicia sua execução imediatamente após A terminar a sua (Tabela 2-1a).

- **Junções do tipo "E"**

  Uma junção do tipo "E" significa que ambas atividades A e B precisam estar prontas para disparar uma atividade C (Tabela 2-1b).

- **Junções do tipo "OU"**

  Uma junção do tipo "OU" significa que uma atividade C inicia quando uma atividade A ou uma atividade B terminam (Tabela 2-1c).

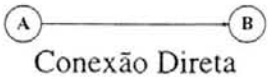
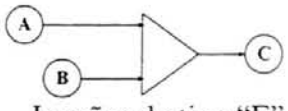
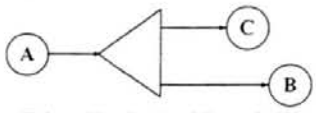
- **Distribuição Simultânea**

  Representa uma situação em que diferentes atividades iniciam simultaneamente. No exemplo daTabela 2-1d, ambas as atividades B e C podem ser iniciadas como conseqüência do término da atividade A.

- **Distribuição não Simultânea**

  Situação em que diferentes atividades são disparadas em diferentes momentos. Similar à junção do tipo "OU", este tipo de distribuição, não apresenta Tabela 2-1e).

Tabela 2-1: Conexão entre atividades

a)  Conexão Direta	**b)**  Junções do tipo "E"
c)  Junções do tipo "OU"	**d)**  Distribuição Simultânea
e)  Distribuição não Simultânea	

- **Atividades do tipo *Fork***

  ➢ Condicional (com exclusão mútua)

A atividade de roteamento condicional (com exclusão mútua) permite a seleção da próxima atividade a ser executada com base nas condições pré-definidas. Observando o exemplo da Tabela 2-2a, verifica-se que quando a atividade A termina, as condições c1,c2,....,cn são avaliadas; uma e somente uma condição deve ser verdadeira, senão um erro é reportado ao responsável pela atividade. A condição verdadeira permite a execução da atividade associada.

Este é o único tipo de atividade que não pode aumentar o número de atividades "ativas" (no final da atividade de entrada, uma e somente uma atividade de trabalho passa a estar habilitada para execução).

➢ *Fork* Total

Uma conexão *Fork* total faz com que todas as atividades de saída sejam habilitadas quando a atividade de entrada termina (Tabela 2-2b).

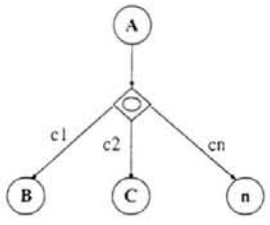
➢ *Fork* Condicional

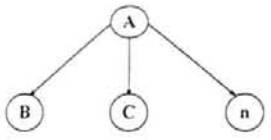
Um *Fork* condicional é realizado quando a atividade de entrada termina e todas as atividades de saída, com condições associadas verdadeiras, são executadas (Tabela 2-2c).

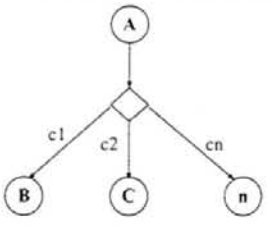
➢ *Fork* Não Determinístico

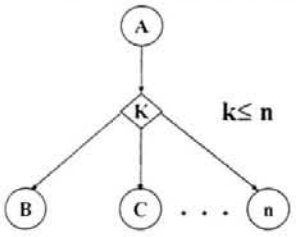
Com *Fork* não determinístico, o sistema seleciona (randomicamente) "k" atividades de saída para serem executadas ao término da execução da atividade de entrada; "k" precisa ser menor ou igual ao número de atividades de trabalho conectadas como saídas (Tabela 2-2d).

Tabela 2-2: Conexão entre atividades do tipo Fork

<table>
<tr>
<td>a)
 

Condicional com Exclusão Mútua</td>
<td>b)
 

Fork Total</td>
</tr>
<tr>
<td>c)
 

Fork Condicional</td>
<td>d)
 

Fork Não Determinístico</td>
</tr>
</table>

- **Atividades do tipo *Join***

➢ *Join* Total

Com atividades do tipo *Join* total a saída da atividade é habilitada quando todas as atividades de trabalho de entrada terminam (Tabela 2-3a).

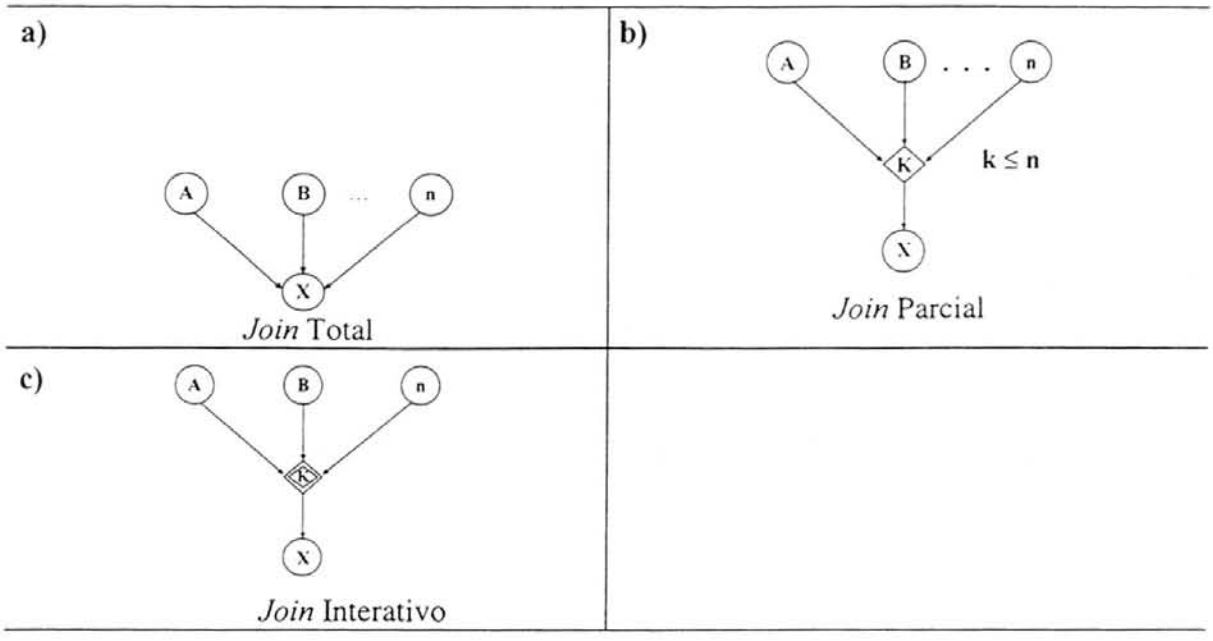
➢ *Join* Parcial

Join Parcial é usado para determinar que a saída da atividade seja habilitada depois que "k" atividades de entrada forem terminadas. "k" deve ser inteiro, menor ou igual ao número total de atividades de entrada (Tabela 2-3b).

➢ *Join* Interativo

Atividades com *Join* interativo funcionam como um contador, que é incrementado toda vez que uma atividade de trabalho termina (independentemente do número de ativação); quando o contador atinge "k" a atividade de saída é executada e o contador é zerado. *Join* interativo com k=1 é freqüentemente usado para descrever ciclos (Tabela 2-3c).

Tabela 2-3: Conexão entre atividades do tipo Join



2.2.3 Evento

Um evento é alguma coisa que acontece, alguma coisa que ocorre. Exemplo: a ocorrência de uma carta sendo remetida ou um sinistro ocorrido por um acidente de carro do Sr. Fulano em junho de 1995. Um evento pode ocorrer como resultado da execução de uma atividade. Da mesma forma, atividades são executadas como resultado da ocorrência de eventos.

Eventos são observáveis em um certo momento, que é normalmente chamado de ocorrência do evento. Uma importante distinção entre atividades e eventos é que atividades podem ser associadas com um intervalo de tempo, enquanto que um evento ocorre em um instante de tempo específico.

2.2.4 Instância (de Processo ou de Atividade)

Uma instância é uma ocorrência de um processo ou de uma atividade em um processo, incluindo seus dados associados. Cada instância pode representar uma forma diferente de execução (por exemplo, em um processo que inclui atividades paralelas, cada instância deste processo pode representar linhas diferentes de execução). Cada instância de processo ou de atividade pode ser controlada independentemente, tem seus estados internos próprios e identidade externamente visível.

2.2.5 Participante do *Workflow* ou Ator

Participante do *workflow* ou ator é um recurso que executa o trabalho representado por uma instância de atividade de um *workflow* (é aquele que atua). Este trabalho

normalmente manifesta-se como um ou mais itens de trabalho (*Work Item)* alocados ao participante do *workflow*, via lista de trabalho (*Work List* - Lista de trabalho é uma lista de itens de trabalho associados a um determinado participante ou um grupo de participantes do *workflow* que podem compartilhar uma lista de trabalho em comum). Exemplo: pessoa colocando uma carta na caixa de correio. Atores podem ser humanos ou automatizados. Ambos os tipos de atores são tratados em um mesmo nível de abstração com o objetivo de modelar a interação propriamente. Esta natureza híbrida, ou seja, aspectos humanos e automatizados tratados de forma homogênea é característica de sistemas de *workflow*.

2.2.6 Processo

Processos de negócios são um conjunto de um ou mais procedimentos ou atividades relacionados que coletivamente realizam um objetivo de negócios ou uma meta (política) normalmente dentro do contexto de uma estrutura organizacional que define papéis funcionais e relacionamentos. Processos de negócios podem consistir de atividades automatizadas, capazes de serem gerenciadas por um *workflow*, e/ou atividades manuais que encontram-se fora do escopo do gerenciamento do *workflow*.

Um processo é um conjunto de atividades que compartilham um objetivo em comum. Os processos são definidos para dar um nome a um conjunto de atividades que estão relacionadas em uma determinada situação.

Processos podem ser divididos em subprocessos. Uma instância de um processo para uma dada situação (específica) constitui-se em um caso.

Em *workflow*, processo é a representação de um processo de negócio através de uma rede de atividades e seus relacionamentos, critérios para indicar o início e término do processo, e informação sobre atividades individuais, tais como participantes associados a aplicações, dados, etc.

2.2.7 Gatilho (*Trigger*)

Um gatilho pode ser visto como uma regra que é avaliada em função da ocorrência de um evento; dada a análise da regra, o gatilho dispara (faz com que inicie a execução) ou não de uma ou mais atividades. Por exemplo, uma atividade é acionada quando algum objeto é transmitido, tal como um formulário, um arquivo, uma mensagem eletrônica, uma chamada telefônica. Este objeto é chamado de transportador (carregador) do evento.

2.3 Os cinco níveis de *workflow*

A tecnologia de *workflow* é definida como um conjunto funcional de agentes implementado em vários níveis de detalhamento. Em [MCK 94] são definidos 5 níveis de *workflow*, descritos a seguir. A partir da análise destes 5 níveis verificou-se a falta de algumas características necessárias à tecnologia de *workflow* o que nos levou a definir um sexto nível, de forma a complementar esta classificação. Alguns aspectos relacionados ao gerenciamento de *workflow* são tratados nestes níveis, porém uma análise mais aprofundada destes aspectos está fora do escopo deste trabalho.

2.3.1 Nível 1: Eventos, Escalonamento e Monitoração

Neste nível, o sistema deve reconhecer eventos, usuários podem escaloná-los, e o

sistema pode monitorá-los permitindo o rastreamento do processo. Eventos incluem ações como inserção, alteração e exclusão em banco de dados e ações do tipo remeter uma fatura, executar um relatório, gerar um arquivo sobre um balanço e alertar sobre eventos que resultam de uma ação como por exemplo, a entrada de uma compra com valor acima do crédito do cliente.

2.3.2 Nível 2: Imagem e Roteamento

O segundo nível providencia características para resultados provenientes de consultas, como pacotes de dados ou imagens digitais, e roteamento destes para recipientes através de mensagens. A maior parte dos processos de negócios gera documentos que tipicamente são impressos, armazenados, enviados por correio - eletrônico (mail), ou fax, tal como faturas, ordens de compra, etc. Armazenando estes documentos na forma de imagens digitais economiza-se espaço físico na organização; bem como a utilização de papéis, podendo preservá-los indefinidamente. Roteando estes documentos automaticamente através de um sistema de envio de mensagens (mail, fax) ganha-se tempo na realização dos trabalhos da organização; reduzindo o ciclo de vida das transações.

2.3.3 Nível 3: Alertas e Ações

O terceiro nível é construído sobre as funcionalidades dos primeiros dois níveis, alertando os usuários do sistema sobre eventos específicos e providenciando um caminho para estes agirem de acordo com o evento. Esta situação é análoga, por exemplo, a caixas de entrada (escaninhos), caixas de saída e listas de a-fazer em uma empresa onde existindo um documento na caixa de entrada, deverá ou não ser tomada uma ação relacionada à lista de afazeres (dependendo da prioridade), ação esta que dará origem a um novo evento.

Tanto os alertas quanto as ações necessitam de uma base de conhecimento para poderem responder aos eventos. Além disso, o sistema precisa providenciar bases de conhecimento adicionais: uma base de regras e uma árvore hierárquica. A base de regras encapsula regras sobre os eventos (que disparam os alertas), guarda informações de como o sistema acumula dados relativo aos alertas para executar ações, e que funcionalidades (tal como tratamento de exceções) os alertas providenciam. A árvore hierárquica indica o relacionamento hierárquico na organização (quem presta contas a quem) de forma a automatizar o roteamento dos alertas enviando-os assim às pessoas corretas. Por exemplo: dada a entrada de uma requisição, um supervisor é alertado com uma caixa de diálogo na sua tela, requisitando sua aprovação imediata. O supervisor não tem conhecimento algum sobre a aplicação que gerou o alerta, porém saberá quem lhe enviou a requisição. Dada sua resposta (aprovado ou não), o sistema fará o roteamento automático da resposta ao remetente.

A base de regras é essencialmente baseada em construções do tipo Se...Então...Senão (*If...Then...Else*). O sistema precisa capturar uma lista de possibilidades relacionada a cada evento, alertas e ações. Por exemplo, uma nova requisição insere um evento, este evento deve ter uma lista completa de entradas no sistema para cobrir um determinado processo de negócios, limites de autorizações, permissões, níveis de prioridade, e tratamento de exceções. Similarmente, a árvore hierárquica precisa incorporar regras para os níveis de roteamento, substituições alternativas, e sobreposição de condições.

2.3.4 Nível 4: Análise do *Workflow*

Este nível providencia um método para a análise do *workflow* e otimização dos processos de negócios com base na informação capturada durante seu monitoramento. Estas funções registram eventos do *workflow*, documentos, alertas, ações e condições em que ocorrem exceções para providenciar um método de análise das atividades do usuário, uso do sistema e tendências dos eventos. Isto permite sistemas de informação relatando a utilização dos recursos e determinando gargalos na execução dos processos, auxiliando também no reprojeto de processos de *workflow*. A análise necessária para reprojetar os processos do *workflow* procurando um melhor desempenho e relacionamento, promove a aplicação de reengenharia. A principal missão da reengenharia é obter uma nova estrutura de empresa com desempenho substancialmente superior à anterior [MAR 95]. De diversos modos, a implantação de sistemas de *workflow* pode colaborar neste sentido; entre eles está o uso de *workflow* para minimizar a transferência e/ou cópia física (papel) de documentos e para permitir que o trabalho executado por cada funcionário seja precisamente medido. Desta forma, a empresa pode adotar políticas de produtividade.

2.3.5 Nível 5: Automação dos Processos Visuais

Este é o nível mais sofisticado, onde a tecnologia de *workflow* permite a utilização de uma ferramenta de automação de processos que representa os processos de negócios e automaticamente gera os componentes do *workflow* anteriormente descritos. Pela busca em um repositório de eventos subjacente, opções de roteamento, alertas e ações, e a ferramenta de automação de processo cria um esquema visual do processo para o conjunto de softwares de gerenciamento de *workflow*. Isto permite definir um caminho para automatizar processos de negócios.

2.3.6 Metanível: Modelagem do *Workflow*

Um sistema de *workflow* contém dois componentes básicos [ELL 96]: o primeiro é o modulo de modelagem de *workflow*, que habilita administradores e analistas a definir procedimentos e atividades, analisá-los e simulá-los e então destiná-los à pessoas. O segundo componente é um módulo de execução do *workflow*, consistindo da interface de execução vista pelos usuários finais e o ambiente de execução que auxilia na coordenação e execução de procedimentos e atividades (nível 5). Dessa maneira, analisando os níveis anteriores, verifica-se a ausência dos componentes de modelagem que devem integrar o sistema de *workflow* constituindo assim, a necessidade do Nível 6 - Metanível de Modelagem de *Workflow*. Neste nível a tecnologia de *workflow* providencia uma ferramenta de modelagem. Esta ferramenta deve ser baseada em uma metodologia que permite definir procedimentos e atividades relacionadas aos processos de negócios, permitindo que estes processos sejam analisados e simulados antes de serem completamente automatizados.

3. Tipos de Sistemas de *Workflow*

Para cada organização, processos de negócios apresentam características próprias, havendo assim a necessidade de um modelo de *workflow* que permita representar de forma realística as informações qualitativas e quantitativas da entidade em estudo. O processamento de transações financeiras, o processamento de documentos legais, entre outros, são exemplos de atividades distintas, mas que podem ou não ser representadas por um mesmo modelo de *workflow*. A identificação dos tipos de sistemas de *workflow* permitirá que a seleção de um modelo utilizado para representar o *workflow* seja facilitada, visto que será possível verificar em qual desses tipos determinado processo se enquadra, impedindo assim a escolha de um modelo inadequado que proporcione uma modelagem complexa com tempos grandes [NIM 96a] . Existem vários tipos de *workflow*, a seguir serão apresentados alguns dos tipos existentes.

3.1 *Workflow* Orientados a Humanos e *Workflow* Orientado a Sistemas

Workflow orientado a humanos envolve humanos na execução e coordenação de tarefas (*ad hoc workflow*) e *workflow* orientado a sistemas, envolve sistemas de computadores que executam operações computacionais intensas e *softwares* especializados em tarefas (*workflow* do tipo produção) [GEO 95]. Enquanto *workflows* orientado a humanos, controlam e coordenam tarefas humanas, *workflow* orientado a sistemas controlam e coordenam tarefas de *softwares* (com uma pequena ou nenhuma intervenção humana). Consequentemente, implementações de *workflow* orientado a sistemas precisam incluir *softwares* para controle de concorrência e técnicas de recuperação para assegurar consistência e segurança. Torna-se importante observar que questões tais como tratamento de exceções, liberação de usuários, priorização e encerramento podem aparecer de diferentes formas em ambos os tipos de sistemas, e necessitam também ser analisadas.

3.2 Ad hoc, Administrativo e Produção

As publicações comerciais diferenciam três tipos de *workflow* [MCC 92,GEO 95,KOB 95]:

- *ad hoc*;
- administrativo;
- produção.

3.2.1 Ad hoc

Workflows do tipo *Ad hoc* suportam definição rápida e execução de modelos de processos menos complexos que podem ser usados para facilitar o fluxo de um único documento em uma única ocasião, ou o fluxo dos documentos de negócios principais em uma base corrente [KOB 95]. Este tipo de *workflow* executa processos de negócios, tais como documentação de produtos ou pedidos vendidos, onde não há um padrão de movimentação de informação entre pessoas.

Tarefas do tipo *ad hoc* para *workflow* tipicamente envolvem coordenação humana, elaboração ou co-decisão [SCH 91]. A ordenação e a coordenação de tarefas em um *workflow* do tipo *ad hoc* não são automatizadas mas são, em vez disso, controladas por humanos. *Ad hoc workflow* tipicamente envolvem pequenos grupos de

profissionais que têm a intenção de apoiar pequenas atividades que requerem uma solução rápida. Figura , representa um *workflow* simplificado tipo *ad hoc* envolvendo o processo de conferência de artigos. As tarefas do processo de revisão consistem em selecionar revisores, distribuir os artigos para os revisores selecionados, ter os revisores executando (fazendo) as revisões, e colaborando na produção de revisão conjunta (agrupada) de um documento, e finalmente enviando as revisões para os autores.

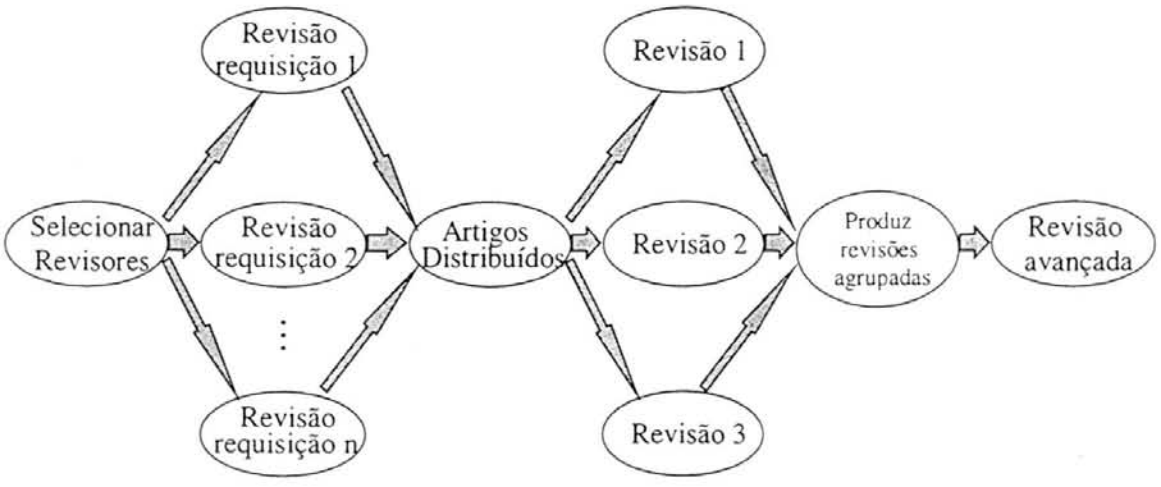


FIGURA 3-1: *Ad hoc workflow* de revisão de artigos [GEO 95]

O processo de conferência de artigos é um *workflow* do tipo *ad hoc* por apresentar as seguintes características:

1. negociação para a seleção de revisores
2. colaboração entre os revisores para produção de uma revisão agrupada

Revisões subsequentes de artigos podem não ser executadas pelos mesmos revisores.

A natureza não estruturada das atividades das organizações apresenta um desafio no desenvolvimento de sistemas de *workflow* e sua subsequente aplicabilidade para os problemas do mundo real, sendo assim necessário utilizar a possibilidade de representação de atividades não-estruturadas como um critério válido para a escolha de um modelo para nossos trabalhos futuros [BLU 96, 96a]. Assim, o estudo de caso escolhido para validação da metodologia apresentada neste trabalho, consiste na representação do processo de revisão de artigos para uma conferência (congresso). Duas implementações distintas foram feitas: uma para a situação que representa um *workflow* do tipo *ad hoc* e outra para um *workflow* do tipo administrativo apresentadas neste capítulo. O detalhamento do estudo de caso é definido no capítulo 8.

3.2.2 Administrativo

Envolve processos repetitivos com regras de coordenação de tarefas simples, tal como roteamento de um relatório de despesa ou requisição de viagem através de um processo de autorização. A ordenação e coordenação de tarefas em *workflows* administrativo podem ser automatizadas. *Workflows* administrativo não englobam um processo de informações complexas e não requerem acesso para sistemas de informação múltiplos usados para suportar produção ou serviços de compradores. Considerando novamente o processo de revisão de artigos; nesta caracterização, assume-se que os

revisores são anteriormente conhecidos (exemplo: os mesmos revisores são usados para revisão dos artigos). Portanto supõe-se que os revisores não colaboram na produção de uma revisão conjunta. Em vez disso, eles produzem revisões individuais que são consideradas pelo editor que toma a decisão final. Sobre esses aspectos, *workflow* de revisão de artigos torna-se um *workflow* do tipo administrativo como descrito na figura 3-2.

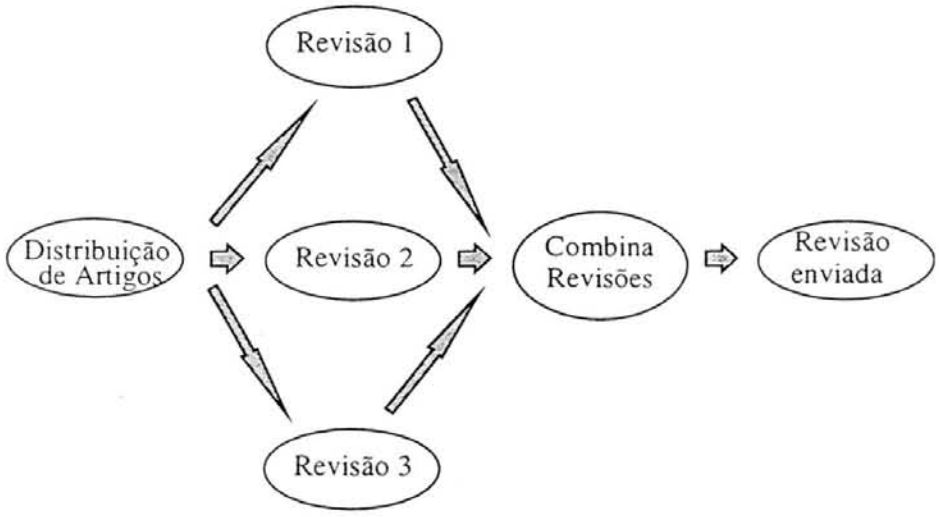


FIGURA 3-2: Administrativo *workflow* de revisão de artigos [GEO 95]

3.2.3 Produção

Envolve processos de negócios repetitivos e previsíveis, tais como empréstimos e seguros. Diferente de *workflows* administrativo, *workflows* de produção englobam processos complexos envolvendo acesso para múltiplos sistemas de informação. A ordenação e coordenação de tarefas nestes tipo de *workflow* podem ser automatizadas. Contudo, para automação de *workflow* de produção, torna-se complicado trabalhar com:

- processos complexos de informações;
- acesso para sistemas de informação múltiplos para execução do trabalho e recuperação de dados para tomada de decisão (*workflow* administrativos confiam em humanos para a maior parte das decisões e execução dos trabalhos).
- interação do sistema de informação como os processos de negócios;
- uso de executores de tarefas automatizados (não humanos).

3.3 *Workflow* Transacional

Outro tipo de *workflow* encontrado na literatura é o *Workflow* Transacional [ALO 95, GEO 95, MOH 94,95]. Este tipo de *workflow* envolve execução e coordenação de múltiplas tarefas que (i) podem envolver humanos, (ii) requer acesso para sistemas do tipo HAD (autônomo, heterogêneo, distribuído), e (iii) suporta o uso seletivo de propriedades transacionais (exemplo: atomicidade, consistência, isolação e durabilidade) para tarefas individuais e entradas de *workflow*. O uso seletivo de propriedades transacionais é necessário para permitir a especialização das funcionalidades de cada *workflow* (exemplo: permitir colaboração de tarefas e suportar estruturas complexas de *workflow*).

3.4 Inter-relacionamento entre os tipos de *Workflow*

A figura 3-3 ilustra o relacionamento entre os tipos de *workflow* e o grau de estruturação e de complexidade das tarefas realizadas nos processos de negócios, segundo [GEO 95].

Workflow com pouca estruturação pode significar um conjunto linear de tarefas a serem realizadas; com alta estruturação pode implicar em uma organização de tarefas somente possível de ser representada na forma de um grafo, onde várias tarefas podem ser executadas concorrentemente e a conclusão de várias é necessária para dar início a outras. A complexidade é determinada pelos tipos de regras ou restrições, de colaboração e coordenação, aplicáveis na execução das tarefas. Complexidade depende, também, dos tipos de sistemas que devem ser integrados para implementar uma tarefa, tais como, aplicações de escritórios, SGBDs e *legacy systems* (sistemas herdados).

FIGURA 3-3: Inter-relacionamentos entre os Tipos de *Workflow*.

Na figura 3-4 são representados alguns segmentos da abrangência das características de um *workflow* onde em um extremo representa-se *workflow* orientado a humanos que envolve colaboração de humanos na execução e coordenação de tarefas e em outro extremo está representado o *workflow* orientado a sistemas que envolve sistemas de computadores que executam operações computacionais intensas e *softwares* especializados em tarefas. Ainda são representadas questões pelo campo de *CSCW* onde *workflows* envolvem predominantemente tarefas humanas e *workflows* do tipo transacional.

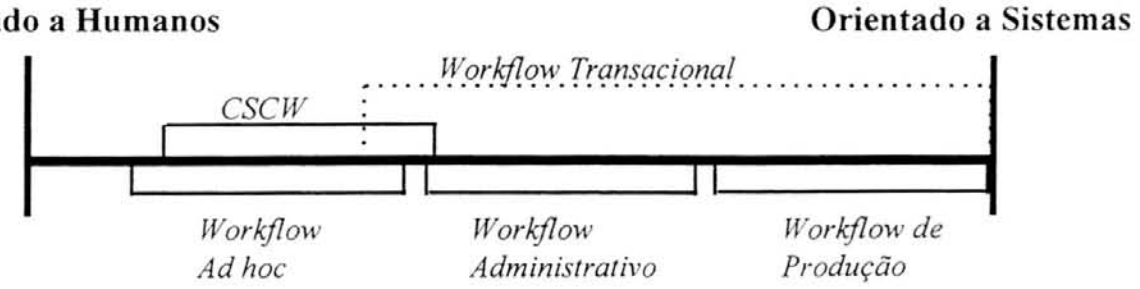


FIGURA 3-4: Caracterizando *Workflow*

4. Modelos de *Workflow*

Os modelos de *workflow* tipicamente envolvem um conjunto de conceitos que são úteis para descrever o processo, suas tarefas, as dependências entre as tarefas, e os papéis necessários para execução de atividades específicas [GEO 95]. Antes de capturarmos um processo precisamos primeiramente compreendê-lo. Isto usualmente envolve entrevistas com especialistas que detêm o conhecimento sobre o processo. Quando a obtenção do conhecimento sobre o processo está completa, o *workflow* é especificado gerando um Modelo de *Workflow*. Existem basicamente duas metodologias de modelagem de *workflow*. São elas: baseada em comunicação e baseada em atividades [GEO 95]. Estas metodologias passarão a ser descritas nos próximos itens.

4.1 Metodologia baseada em comunicação

A metodologia baseada em comunicação tem origem em [WIN 87,87a,88,94]. Os modelos baseados em comunicação enxergam o trabalho como um conjunto de interações humanas bem definidas (laços de trabalho), representando compromissos realizados entre pessoas envolvidas. Esta metodologia é baseada na teoria dos atos da fala e da compreensão dos autores sobre como se desenrola o processo de comunicação dentro das organizações [MED 92].

Uma das metodologias baseadas em comunicação existentes é a da Action *Workflow* Tecnologies [ACT 96, 96a, 96b]. A metodologia da Action *Workflow* assume que o objetivo do processo de negócio é promover a satisfação do cliente. Seu modelo é baseado na pesquisa em comunicação humana que identifica as formas básicas nas quais as pessoas utilizam linguagem humana. A metodologia reduz toda a ação de um *workflow* em 4 fases baseada na comunicação entre um cliente e um executor. As fases são Preparação, Negociação, Execução, Aceitação. Esta metodologia é descrita mais detalhadamente no capítulo 8.

4.2 Metodologia baseada em atividades

O foco desta metodologia está na modelagem do trabalho em vez de no relacionamentos entre humanos. Nesta metodologia, *workflows* consistem de atividades [GEO 95]. A atividade pode ser dividida em subatividades. Esta atividade tem dependências com outras atividades em um mesmo nível, possuindo também um papel que determina um humano ou um programa como executor da atividade. Esta metodologia não impõe nenhuma ordem pré-definida entre as atividades. O projetista é completamente livre para modelar o fluxo da maneira que melhor lhe convier.

Os modelos baseados em atividade enxergam o trabalho como uma seqüência de atividades, onde cada atividade recebe um conjunto de entradas e produz um certo conjunto de saídas. As metodologias baseadas em atividades, caracterizam-se em termos gerais pela existência de:

- um processo de transformação qualquer (atividade);
- um conjunto de entradas para este processo (eventualmente pré-condições para sua ocorrência);
- um conjunto de saídas deste processo (eventualmente pós-condições de sua ocorrência).

4.2.1 Metodologia Orientada a Objeto

Algumas metodologias de modelagem de *workflow* baseadas em atividades, podem utilizar técnicas de modelagem orientada a objetos para definir a especificação do *workflow* (como é o caso da metodologia utilizada em [BAI 93]). Dessa maneira, descrevemos algumas características da modelagem orientada a objetos que podem ser úteis na especificação de um sistema de *workflow*.

Objetos são criados como instâncias de classes; eles têm atributos e comportamentos, e podem ser especializados e estendidos para uma aplicação ou circunstância particular do *workflow*. Por exemplo, as atividades, as ligações entre atividades, e a ordem na qual essas atividades são executadas (uma fila de execução de atividades) são objetos com atributos, estados e identidade dentro do *workflow*. As instâncias do *workflow* (podem ser vistas como instâncias do processo) também são objetos. Uma instância de uma classe do *workflow* contém atributos indicando por exemplo o seu tipo, nome do processo, a descrição, etc. Em outras palavras, a especificação do processo em um modelo de *workflow* orientado a objetos permite representar:

- as instâncias dos processos;
- o comportamento (ou operações) para as instâncias dos processos. Este é um conceito muito útil, permitindo que o *workflow* esteja habilitado a definir operações para processos particulares [KHO 95];
- especialização ou herança – um processo do *workflow* pode ser especializado, e uma classe do *workflow* poderá herdar características de outra classe. Isto significa que o processo representado pelo *workflow* pode herdar o comportamento e a estrutura de um outro processo, tão bem quanto as instâncias do processo;
- como sabemos, um dos conceitos importantes na orientação a objetos é a identidade do objeto. A identidade do objeto é um propriedade de um objeto que distingue o objeto de todos os outros em uma aplicação. Através da identidade dos objetos, sistemas podem suportar compartilhamento de objetos. Isto significa que os objetos podem ser aninhados com outros objetos e estes, compartilhados por múltiplos objetos. Por exemplo, algumas vezes o próprio processo do *workflow* é complexo e contém sub*workflow*s aninhados. A orientação a objeto aqui significa ter um identificador único para cada *workflow* e permitir definir hierarquias para especificação do processo e instâncias do processo.

4.3 Linguagens de Especificação de *Workflow*

Dadas as metodologias de modelagem de *workflow*, convém descrevermos algumas das possíveis formas de especificação que os modelos de *workflow* utilizam. Normalmente, modelos de *workflow* usam para sua especificação Linguagens de *Workflow* que podem ser divididas em linguagens de especificação gráficas e linguagens de especificação textual [CAS 95].

4.3.1 Linguagens de especificação gráficas

Linguagens de especificação gráficas geralmente permitem uma descrição da estrutura do *workflow* pela composição de elementos gráficos em um gráfico de fluxo (*flow chart*); usualmente nodos representam tarefas, enquanto um gráfico orientado

conecta um nodo A com um nodo B indicando que B irá iniciar quando A estiver completo. A parte gráfica é integrada freqüentemente por uma descrição textual especificando as características de tarefas únicas. Um exemplo de um modelo que descreve a estrutura do *workflow* pela composição de elementos gráficos pode ser encontrado em [GEO 95].

4.3.2 Linguagens textuais

As linguagens textuais, diferentes das gráficas, diferenciam-se muito entre si. Algumas delas são extensões das linguagens existentes, enquanto outras são novas, tentando providenciar mais informação semântica para o controlador do *workflow*. Ferramentas gráficas permitem especificações em um alto nível, que podem ser traduzidas em especificações textuais. Algumas especificações textuais podem ser compiladas ou interpretadas em um código de baixo nível específico do sistema; portanto, pode-se dizer que linguagens textuais oferecem especificações em um nível intermediário e as gráficas apresentam especificações em um alto nível. Um exemplo de um modelo que utiliza uma linguagem textual para definição do processo do *workflow* é encontrado em [WFM 96a].

5. Estudo de modelos de *workflow*

Esta seção tratará do estudo e análise de alguns modelos de *workflow* existentes. A partir deste estudo, serão determinados um conjunto de critérios básicos para análise de modelos de *workflow*.

5.1 Modelo de Gatilhos

Este modelo foi proposto por Stef Joosten [JOO94]. O propósito deste modelo é descrever o comportamento dinâmico do sistema em termos de gatilhos. Gatilhos são convencionalmente modelados por meio de Redes de Petri. Na proposta de Joosten é demonstrado também o mapeamento de um modelo de gatilhos para um modelo de Redes de Petri.

O processo de análise do *workflow*, neste modelo, consiste em entrevistas que permitem determinar os papéis a serem representados e quais atividades são executadas por estes papéis. As entrevistas devem estar centradas na determinação das atividades, papéis e gatilhos.

Neste modelo uma atividade é definida como um conjunto de eventos (alguma coisa que acontece) que ocorrem sobre a responsabilidade de um ator (aquele que age). A definição de atividade demonstra que a importância maior é dada ao ator (humano ou automatizado) que é responsável pela atividade, mais do que propriamente por quem a executa. O processo é definido como um conjunto de atividades que compartilham um propósito em comum. Gatilhos disparam atividades em função da ocorrência de eventos. Dessa forma, neste modelo o *workflow* é representado por atividades que se relacionam com outras por um gatilho, e que são disparadas por eventos externos.

5.1.1 Representação

É representada uma representação gráfica na qual, cada atividade é representada por um retângulo, contendo o nome da atividade. Uma linha direcionada para uma atividade significa que a atividade pode ser disparada por eventos que ocorrem como resultado da atividade que inicia a linha. O modelo de gatilhos é dividido em colunas, cada uma das quais contendo uma atividade associada a um papel particular. A FIGURA 5-1 exemplifica um modelo de gatilhos para o exemplo visto de revisão de artigos.

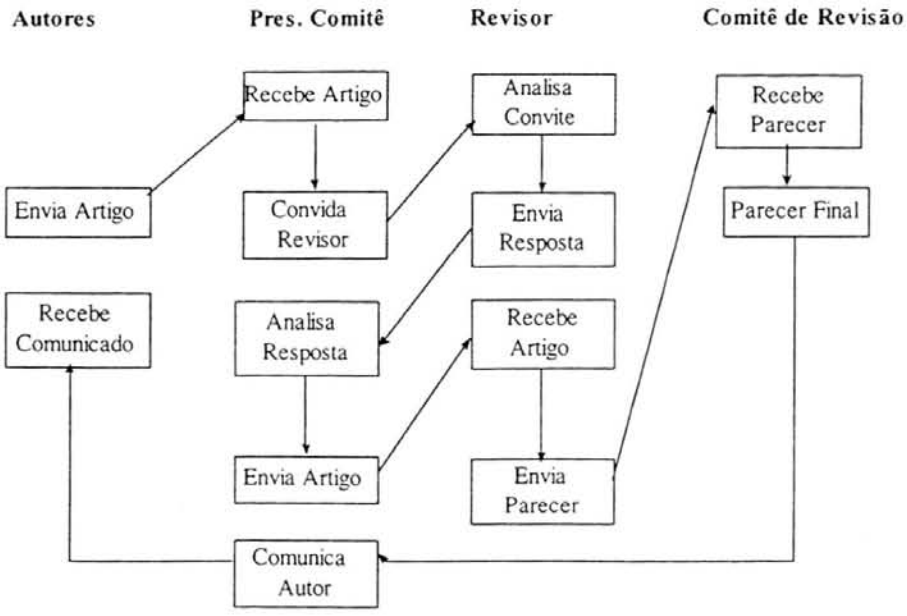


FIGURA 5-1: Modelo de Gatilhos

Importante observar que o modelo permite que atividades sejam regularmente disparadas por um evento interno à atividade (um relógio), demonstrando dessa forma que o modelo de gatilhos não segue necessariamente o fluxo do processo ou da informação e que atividades de controle de tempo, como por exemplo "Data Limite", podem ser representadas no modelo.

Uma linguagem de programação funcional foi utilizada para fazer um protótipo do modelo de gatilhos em termos de grafos. Uma função definida, mapeia um modelo de gatilhos para um modelo de Redes de Petri. Esta função representa a semântica do modelo de gatilhos [JOO 95].

Analisando o modelo de gatilhos, verifica-se que este está habilitado a suportar *workflows* do tipo *ad hoc* e administrativo. Permite a execução de tarefas disparadas por um evento interno a atividade (um relógio), demonstrando, assim, que atividades de controle de tempo poderiam ser representadas no modelo. Além disso, como não está explícita a forma de representação de tarefas automáticas, concluí-se que as tarefas disparadas por um evento interno, controlado por tempo por exemplo, podem ser consideradas automáticas. *Workflows* de produção são difíceis de serem gerenciados neste modelo, visto que para todo evento excepcional, um responsável (humano) precisa ser notificado para seu tratamento. Isto é feito pela ausência de um mecanismo de tratamento de exceções. A representação do sincronismo entre tarefas é permitida, possibilitando, dessa forma, ordenação e coordenação de eventos.

5.2 Modelagem Conceitual de *Workflow* [CAS 95,95a, 95b]

Neste modelo, um esquema de *workflow* consiste em um conjunto de tarefas e um conjunto de conexões entre elas. No esquema descreve-se quais tarefas podem ser executadas, em qual ordem, quem pode estar encarregado dela, e que operações podem ser executadas. Tarefas no esquema podem ser tarefas de trabalho (unidades de trabalho simples), tarefas de roteamento (roteadores de fluxo - tarefas de roteamento são expressadas por mecanismos de fluxo de controle análogos aos das linguagens de programação comuns, com a adição de construções que permitam paralelismo), super-

tarefas (unidades de trabalho complexas, que podem ser decompostas em tarefas de trabalho ou outras super-tarefas), ou multi-tarefas (execuções paralelas de mais cópias da mesma tarefa de trabalho).

Para descrever o esquema do *Workflow* o modelo utiliza uma linguagem de definição gráfica combinada com uma linguagem de definição textual. A linguagem de definição textual descreve tarefas de trabalho a serem executadas durante o *workflow* e os mecanismos que são usados para sua ativação e terminação, ambos em situações normais e excepcionais. A linguagem de definição gráfica permite especificar a estrutura do fluxo de um esquema de *workflow* pela composição de objetos gráficos.

A coordenação de tarefas de trabalho é suportada por um número restrito de caminhos alternativos, providenciando, portanto, construções básicas de roteamento tais como, um *fork* e um *join*. Tarefas do tipo *fork* tem uma tarefa de trabalho como entrada e muitas tarefas de trabalho como saída; quando a tarefa de entrada termina, algumas das tarefas de saída são criadas, concordando com a especificação. Tarefas do tipo *join* tem muitas tarefas de trabalho como entrada e uma tarefa de trabalho como saída; seu propósito é verificar quando uma tarefa de entrada termina, se a tarefa de saída precisa ser criada ou não. O comportamento de tarefas de trabalho é formalmente descrito pela lista de suas pré condições, suas ações, e suas condições excepcionais durante a execução. A característica peculiar da WFDL proposta é habilitar, com condições nas tarefas de trabalho, ações, e condições excepcionais, a manipulação de bancos de dados externos (através do padrão de declarações SQL2).

O modelo descreve uma instância de *workflow* (ou caso) como uma execução particular de um esquema. Por exemplo, um esquema de um *workflow* pode descrever o processo de revisão de artigos submetidos; uma instância daquele esquema é criado quando o um editor recebe um novo artigo. Desta forma, normalmente, muitas instâncias do mesmo esquema podem ser ativadas ao mesmo tempo. A definição do esquema do *workflow* envolve a definição de variáveis e estrutura (fluxo) do *workflow*, e as características das tarefas e super-tarefas. Todo caso de *workflow* é gerenciado pelo responsável que será notificado sobre problemas ou eventos excepcionais ocorridos durante a execução do caso.

Descrição de *workflows* e tarefas (exceto tarefas de roteamento) iniciam com definições de constantes, tipos, variáveis e funções neste modelo. Definições no contexto do esquema do *workflow* são globais (visíveis para toda tarefa no *workflow*); definições no contexto de super-tarefas ou tarefas de trabalho são locais (visíveis para toda tarefa no *workflow*); definições no contexto de super-tarefas ou tarefas de trabalho são locais (visíveis para toda tarefa ou super-tarefa). Em ambos os casos variáveis são não persistentes, ou seja, existe somente durante a execução do *workflow* ou da instância da tarefa de trabalho. Portanto variáveis não podem ser usadas com instâncias do *workflow* para ligar outras instâncias.

Declarações também podem incluir a definição de dados persistentes, que são compartilhados por todos os agentes do *workflow* e possivelmente por agentes de outros *workflows*. Estes dados podem ser normalmente definidos externamente (isto é, sua existência pode ser independente de uma aplicação particular de *workflow* sendo modelado); para simplificar, é utilizado um modelo de dados relacional para denotar dados persistentes.

Nesta abordagem, manipulação de dados e recuperação é somente uma maneira de efetuar trocas de estruturas de dados com outros *workflows*, seguindo desta forma uma

abordagem *blackboard*. Outros bancos de dados externos que são acessados independentemente por agentes durante a execução de suas tarefas de trabalho não precisam ser explicitamente definidos.

Cada *workflow* é definido para um administrador do *workflow* (WFA - *workflow administrator*) que é responsável pela geração e pela compilação do esquema do *workflow*. O processo é gerenciado por um responsável, que é a pessoa que também ordena o início do *workflow*. O responsável pelo processo, é notificado de certas exceções que podem ocorrer durante a execução do *workflow*.

Realizada uma análise do modelo de [CAS 95] observa-se que este também permite a representação de *workflows* do tipo *ad hoc* e administrativo. Neste modelo, o WFMS tem a função de determinar quando uma certa tarefa de trabalho precisa ser iniciada, identificando também através de uma lista pré-definida, o agente executor para esta tarefa. A tarefa de trabalho pode ser completamente automatizada ou designada para um agente. *Workflows* de Produção podem ser gerenciados neste modelo, visto que, todo o evento excepcional para o *workflow* pode ser verificado por um mecanismo de tratamento de exceções definido na própria tarefa de trabalho. Permitindo dessa forma a definição de relacionamentos complexos entre as tarefas e execução e controle de tarefas com pequena intervenção humana. Significativa contribuição é a noção de modularização (super-tarefas), além de construções básicas de paralelismo (também presentes no modelo de gatilhos), tais como *Fork* e *join*; ainda, o comportamento de tarefas de trabalho é formalmente descrito pela lista de suas pré condições, suas ações, e suas condições excepcionais durante a execução. O modelo mostra como descrições formais do *workflow* podem ser usadas como entradas para gerar o esquema de dados do *workflow* e o código de regras ativas para seu gerenciamento. Em adição, o formalismo é providenciado através do paradigma de regras ativas conveniente para expressar computações reativas que normalmente são influenciadas por "eventos externos" gerados fora do WFMS, tal como exceções ou trocas em valores verdadeiros de pré-condições de tarefas.

5.3 Modelo de Interoperabilidade da WfMC

Esta seção tratará do Modelo de Referência de *Workflow* desenvolvido pela *Workflow Management Coalition* (WfMC) [WFM 96a]. A WfMC é um grupo de companhias que tem reconhecido que todos os produtos de gerenciamento de fluxo de trabalho tem algumas características comuns, procurando dessa forma habilitar esses produtos a alcançar um nível de interoperabilidade através do uso de padrões comuns para várias visões. A WfMC foi estabelecida para identificar áreas funcionais e desenvolver especificações apropriadas para implementação de produtos de *workflow*. Sua intenção é de que estas especificações habilitem a interoperabilidade entre produtos heterogêneos de *workflows*.

5.3.1 Introdução

O modelo de referência de *workflow* da WfMC foi desenvolvido a partir da estrutura genérica de aplicações de *workflow* identificando as interfaces com estruturas que habilitam produtos para interoperar com uma variedade de níveis. Todos sistemas de *workflow* contém um número de componentes genéricos que interagem em um conjunto de formas definidas; o modelo de referência da WfMC procura exibir os diferentes níveis de interações com cada um desses componentes genéricos.

5.3.2 O modelo de *workflow*

A FIGURA 5-2 ilustra o modelo de referência da WfMC que representa os componentes e interfaces que compõem sua arquitetura. A WAPI (*Workflow APIs and Interchange Formats*), representa um conjunto de construções pelas quais os serviços relacionados a sistemas de *workflow* podem ser acessados permitindo interações entre *softwares* de *workflow* e outros componentes do sistema.

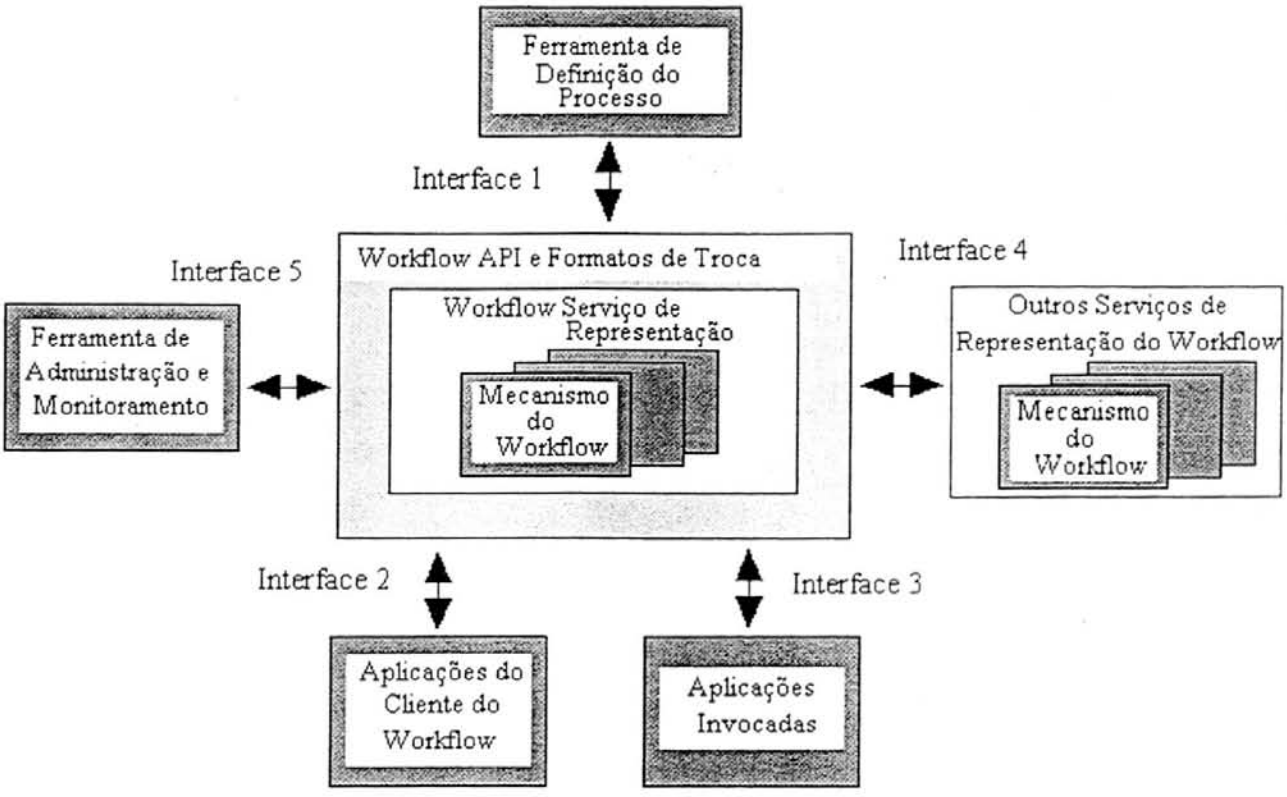


FIGURA 5-2: Modelo de Referência de *Workflow* - WfMC [WFM 96]

Neste modelo, estamos particularmente interessados na interface1 que é neste modelo é utilizada para a representação do processo do *workflow*. Assim, não fazendo parte do escopo deste trabalho as definições e representações relacionadas aos outros componentes e interfaces.

5.3.3 Inteface 1

A Interface1, uma interface comum para a troca de definições de processos de *workflow*, é baseada em uma linguagem padrão para definição do processo, mais especificamente, a WPDL (*Workflow Process Definition Language*). A WPDL é uma linguagem para a descrição do *workflow* como um conjunto de caracteres ASCII usando chaves (por exemplo, *WORKFLOW*, *ACTIVITY*, *DESCRIPTION* etc.) para especificação de objetos, atributos e relacionamentos, e usando variáveis para especificar nomes e valores. A gramática é dada em EBNF (Extended Backus Normal Form). A linguagem WPDL oferece:

- um número mínimo de entidade pré-definidas ;
- um número mínimo de relações pré-definidas entre as entidades;
- um número de atributos pré-definidos (usando chaves);

- atributos adicionais genéricos;
- relações adicionais adversas;
- objetos de dados genéricos adicionais.

Um dos elementos chave da WPDL é sua extensibilidade para manusear informações usadas por uma variedade de ferramentas diferentes. Baseado sobre um limitado número de entidades que descreve um processo de definição do *workflow* a WPDL suporta um número de diferentes abordagens.

As duas possíveis abordagens para uma definição comum do processo são:

i) define APIs em tempo de construção (build-time APIs) para criação de objetos, seus atributos e os relacionamentos dos participantes em um *workflow*;

ii) define uma linguagem comum para descrição do processo do *workflow*.

Este trabalho, particularmente está centralizado na análise do item ii.

A WPDL determina um formato de troca comum que suporta a transferência de definições dos processos do *workflow* entre diferentes produtos de *workflow*; permitindo um método comum para acessar e descrever definições de *workflow*. Assim, um meta modelo de dados de definição do processo foi estabelecido. Este meta modelo identifica as entidades comumente utilizadas com o processo de definição. Uma variedade de atributos descreve as características deste conjunto limitado de entidades. Baseado neste modelo ferramentas específicas de vendedores, podem transferir modelos via um formato de troca comum. O mecanismo de transferência pode ser baseado em uma API ou orientado a lotes (via arquivos ou transferência de memória).

O modelo da WfMC através da WPDL também suporta *workflows* do tipo *ad hoc* e administrativo. A atividade (tarefa) pode ser implementada como atômica ou como um subprocesso. Podendo também ser definida como automática ou manual, sendo disparada pelo sistema ou por um usuário. *Workflows* de Produção são gerenciados neste modelo; todo o evento excepcional para o *workflow* pode ser tratado através de um atributo definido na atividade que permite a invocação de uma aplicação dedicada ao tratamento de erros, permitindo o controle e tratamento das atividades com pequena ou nenhuma intervenção humana. Atividades são conectadas umas com as outras através de informações de transição; estas transições determinam o fluxo de controle entre as atividades, porém são controladas por regras simples, proporcionando uma dúvida com relação a possibilidade de representação de construções mais complexas.

5.4 Critérios necessários à modelagem do *workflow*

Nesta seção, a partir da análise das características dos modelos estudados, será definido um conjunto de critérios que serão utilizados na análise da técnica de modelagem proposta neste trabalho. Os critérios de modelagem foram definidos em função das facilidades propostas por cada modelo, identificando dessa forma um conjunto de conceitos fundamentais necessários à modelagem de *workflow*.

- Um importante critério a ser observada nos modelos é a descrição formal do processo de modelagem. O formalismo garante uma interação correta dos participantes do processo, a consistência dos dados e um processo seguro. Muitas das características apresentadas na análise dos modelos demonstram a falta de formalismo. Deste modo,

para determinadas características, serão consideradas como forma de avaliação o formalismo presente nas mesmas.

- A associação de uma atividade a um determinado grupo de pessoas, demonstrando a possibilidade de agrupar participantes, determina um critério importante a ser analisado na representação de um sistema de *workflow*. Consiste na definição de papéis para os participantes do *workflow*. O conceito de Papel em cada um dos modelos estudados é análogo, permitindo dessa forma uma análise comparativa dos mesmos com relação ao processo de representação. Nos modelos Gatilhos e de [CAS 95], a definição do papel é feita através de um rótulo atribuído ao participante ou grupo de participantes do *workflow*, não apresentando, dessa maneira, descrição das características (habilidades, especialização, nome, capacidade, etc.) necessárias aos participantes para executar as atividades do *workflow*. A mesma situação ocorre para o modelo da WfMC, porém este permite descrever algumas características dos participantes do *workflow* através de um conjunto de atributos (pré-definidos) descritos no papel (por exemplo, a descrição da tarefa a ser executada pelo papel, a lista de participantes que desempenham o papel, etc.), demonstrando assim um certo formalismo e permitindo um maior relacionamento dos papéis com as atividades a serem executadas.

Da análise da definição dos papéis conclui-se que os modelos não oferecem um mecanismo de definição de atributos. A presença deste mecanismo torna-se importante para representação formal do papel. Nota-se também a ausência de algum mecanismo de herança para papéis pertencentes a uma mesma classe, o que determina que características similares a estes papéis precisam ser repetidas para cada um, causando desta forma, um retrabalho de definição desnecessário.

- Um outro critério a ser analisado, consiste na representação de atividades automáticas pelo modelo. Os conceitos de atividade e tarefa confundem-se nos três modelos. [CAS 95] trabalha com o conceito de tarefa de trabalho, o modelo de gatilhos e da WfMC com o conceito de atividades. Entendemos, que para fins de análise, atividade e tarefa tem conceitos análogos. Para a representação de atividades automáticas, cada modelo possui características específicas, algumas explícitas, como o modelo de [CAS 95] (através de um atributo *automatic* que define a tarefa como automática) e da WfMC (através de um atributo *Mode*), e outras implícitas como o modelo de gatilhos onde a tarefa é definida como automática através de um evento interno a ela.

- Considerando o formalismo na representação das atividades, este mostra-se ausente no modelo de gatilhos, enquanto que no modelo de [CAS 95] e da WfMC existe um pequeno formalismo através da descrição de alguns atributos. A justificativa para a ausência de formalismo na representação torna-se clara pois, quanto menos formal mais flexível torna-se a representação do processo. Essa situação seria considerada normal, visto a necessidade dos modelos de *workflow* terem que representar realidades diversas dos processo de negócios. O problema da ausência do formalismo, reflete-se posteriormente na execução do *workflow* pelo WFMS este terá dificuldades em monitorar o *workflow* com relação a características que permitam determinar o término e o andamento de determinadas atividades.

- Observando o conjunto de objetos que compõem o *workflow*, notou-se outros dois fatores relevantes à representação formal do mesmo. Esses fatores são relativos à representação da estrutura do objeto e à representação da estrutura do fluxo. Não vemos

aqui o objeto apenas como um documento ou formulário presente no *workflow*, mas sim como agentes, recursos e atividades que participam e influenciam o mesmo. A estrutura do objeto, consiste na definição de valores que representam o objeto na realidade modelada e que são essenciais à representação do *workflow*. A estrutura do fluxo consiste em definir as dependências entre as atividades. As dependências podem ser através de uma relação de causalidade (predecessor/sucessor) ou envolver a ativação de caminhos alternativos (*and/or join, and/or split*). Os valores existentes na estrutura do objeto e na estrutura do fluxo podem determinar o conjunto de decisões que devem ser tomadas pelo WFMS (por exemplo, decisões relacionadas ao fluxo) para o gerenciamento do *workflow*, como também, permitir o monitoramento do objeto. O modelo de gatilhos, nada oferece para a representação da estrutura do objeto. O modelo de [CAS 95] consegue através da sua linguagem de definição textual, definir uma sintaxe para representação da estrutura de objeto, permitindo a descrição completa deste. A WfMC permite a definição de alguns atributos que influenciam a decisão durante a execução do *workflow*, através da entidade de Dados Relevantes ao Processo do *workflow*. Os modelo da WfMC e de [CAS 95] representam a estrutura do fluxo através da sua linguagem de definição de *workflow*, enquanto que o modelo de gatilhos apresenta o fluxo através de uma representação gráfica.

- Um critério que não pode ser negligenciado na modelagem do *workflow* está relacionado ao tempo. O fator tempo na modelagem do *workflow* pode determinar situações de disparo automático de atividades, lembretes, auditorias, pré-condições temporais de execução, etc. A definição do processo pode centralizar atributos como: - data de início e data limite de início de uma atividade - prazos limites (*deadlines*) e pontos de verificação - resubmissão automática no caso de espera em eventos relacionados a tempo e eventos relacionados a clientes.

Nenhum dos modelos analisados apresentou uma forma de tratamento explícito destas necessidades.

- A possibilidade de tratar exceções também foi avaliada nos modelos. Entende-se por tratamento de exceções a possibilidade de especificar quais ações devem ser executadas caso uma tarefa falhe ou um *workflow* não consiga ser completado [GEO 95]. Esse critério auxilia a avaliação do modelo com relação à capacidade de representar ou não *workflows* altamente estruturados. O modelo da WfMC apresenta a possibilidade de tratamento de exceções através de um atributo definido na atividade que permite a invocação de uma aplicação externa de controle de erros. No modelo de [CAS 95] o tratamento de exceções é feito no módulo de controle de tarefas pela definição de um predicado de condição-reação. O modelo de Gatilhos não demonstrou nenhum mecanismo de controle de exceções.

A partir do estudo realizado neste capítulo, acreditamos que os critérios identificados devem fundamentalmente compor um modelo de *workflow*. Os resultados da análise são apresentados na TABELA 5-1:

	Modelo de Gatilhos	Modelo de [CAS 95]	WfMC
Workflow* do tipo *ad hoc	representa	representa	representa
***Workflow* administrativo**	representa	representa	representa
***Workflow* de produção**	não representa	representa	representa
Formalismo apresentado pelos modelos	informal	semi-formal	semi-formal
Formalismo na definição de papéis	informal (representação através de um rótulo)	informal (representação através de um rótulo)	semi-formal conjunto de atributos (pré-definidos)
representação formal de tarefas automáticas	representa informalmente (eventos internos)	representa (através de atributo)	representa (através de atributo)
formalismo na representação das atividades	não apresenta	semi-formal (através de atributos pré-definidos)	semi-formal (através de atributos pré-definidos)
representação da estrutura do objeto	não representa	representa formalmente	representa semi-formal
representação da estrutura do fluxo	representa informalmente	representa formalmente	representa formalmente
representação de aspectos relacionados a tempo	não definido explicitamente	não definido explicitamente	não definido explicitamente
representação formal do tratamento de exceções	não representa	representa formalmente (predicado de condição-reação - "on" <condition> "do" <reaction>	representa formalmente - porém o tratamento é externo ao modelo (atributo definido na atividade que permite a invocação de uma aplicação externa de controle de erros)

TABELA 5-1: Tabulação dos critérios relativos a análise dos modelos

6. Introdução ao TF-ORM

A especificação, ou modelagem da aplicação, baseia-se na coleta de requisitos a partir do mundo real, procurando a identificação e descrição de todas características dinâmicas das aplicações. Deve contudo ser, totalmente independente da implementação, servindo como base para a realização da mesma. Ao construir a especificação de um sistema de informação, não só a estrutura dos dados manipulados deve ser definida, mas também sua dinâmica - seu comportamento com a passagem do tempo. Na coleta dos requisitos do sistema, a qual fornece os elementos necessários à posterior especificação, devem ser identificados os requisitos temporais da aplicação em questão. O método utilizado na especificação do sistema de informação correspondente à aplicação deve permitir que todos os aspectos temporais sejam representados [EDE 94]. Além disso, como visto na seção 5.4, existe a necessidade, na especificação do *workflow*, de um modelo formal para garantir uma interação correta dos participantes do processo, a consistência dos dados e um processo seguro. Assim, para este trabalho foi escolhido o modelo de dados TF-ORM (*Temporal Functionality in Objects with Roles Model*) [EDE 93, 94, 94a] como referência, devido ao seu formalismo, as suas características de orientação a objetos, lógica temporal e, também, devido ao significativo conjunto de atividades desenvolvidas com este modelo no CPGCC. Este capítulo introduz alguns conceitos básicos relacionados ao TF-ORM necessários a compreensão e desenvolvimento deste trabalho.

O modelo TF-ORM é um modelo de dados orientado a objetos que utiliza o conceito de papéis para representar diferentes comportamentos dos objetos, facilitando o processo de análise da aplicação e a representação da evolução dos objetos através do tempo. O modelo permite modelagem dos aspectos estáticos e dinâmicos da aplicação pois considera todos estados do objeto, associando informações temporais às propriedades que podem mudar de valor ao longo do tempo. Apresenta ainda aspectos temporais e aspectos próprios do paradigma de orientação a objetos, tais como: i) a hierarquia de classes; ii) a recuperação de atributos apresentados por classes; e iii) a existência de diversas instâncias de uma classe identificada.

Os processos relacionados a um *workflow* têm características próprias (por exemplo, a representação de agentes executores e responsáveis pelo processo); assim, para que o TF-ORM represente estas características de forma completa, algumas extensões nesta classe foram realizadas. As extensões são descritas no capítulo 7.

6.1 Definição de Classes e Papéis

Uma classe é definida através de um nome c_n e de um conjunto de papéis R_i, cada um representando um comportamento diferente dos objetos desta classe:

$$\text{classe} = (c_n, R_0, R_1, ..., R_n)$$

Ao definir uma classe, deve ser também definido o seu tipo. São três os tipos de classes: classe de agente (*agent class*), de recurso (*recourse class*) ou de processo (*process class*). Classes de agentes representam pessoas atuando no sistema que está sendo modelado, e cada papel representa um comportamento que esta pessoa pode apresentar na aplicação considerada. As estruturas dos recursos (dados, documentos) são representadas através de classes de recursos, com os papéis que os recursos podem desempenhar durante a sua existência. As classes de processos integram estes agentes e recursos, descrevendo a organização do trabalho desenvolvida na aplicação, e a cooperação entre os agentes. Os papéis nas classes de processo representam diferentes

tarefas a serem executadas no processo. No estudo realizado sobre *workflow*, identificamos que este esta diretamente associado a processos de negócios. Desta forma, a modelagem do *workflow* deve estar centralizada nos processos que compõem o caso. A representação destes processos no TF-ORM é feita através da classe de processos.

Cada papel é constituído de um nome Rn_i, de um conjunto de propriedades P_i deste papel (descrições abstratas dos tipos de dados implementados como variáveis de instâncias), de um conjunto de estados S_i que este objeto pode assumir quando desempenhado este papel, de um conjunto de mensagens M_i que o objeto neste papel pode receber e enviar, e de um conjunto de regras Ru_i (regras de transição entre estados e regras de integridade):

$$Ri = < Rni, Pi, Si, Mi, Rui >$$

Além das mensagens acima mencionadas, as classes de agentes podem também representar um conjunto de decisões D_i, através das quais é modelada a parte não estruturada das aplicações, de uma maneira formal:

$$Ri = < Rni, Pi, Si, Di, Mi, Rui >$$

Todas instâncias de papéis evoluem independentemente, podendo haver interação entre elas através do envio de mensagens.

Dois tipos de instâncias são, portanto, utilizadas no modelo TF-ORM: instâncias de classes (objetos), instâncias de papéis. Uma instância de uma classe (um objeto) pode representar diversas instâncias de papéis. Os dois tipos de instâncias apresentam identificadores únicos – identificadores de objetos e identificadores de instâncias de papéis.

Todo o objeto apresenta um papel R_0, denominado papel básico (*base role*). Neste papel são descritas as características iniciais de toda instância desta classe e as suas propriedades globais. Este papel apresenta somente propriedades (herdadas por todas instâncias dos outros papéis) e regras (que controlam instâncias de outros papéis). Seus estados são pré-definidos (*active* e *supended*), assim como as mensagens que podem ser utilizadas em suas regras (mensagens de criação, suspensão, retomadas e término de uma instância de uma classe e de um papel).

6.2 Definição das Regras

Dois tipos de regras podem ser definidas em TF-ORM: regras de transição de estados e regras de integridade. Todas as regras recebem um nome, nome este que é utilizado somente para facilitar a modelagem – através deste nome pode-se identificar qual o funcionamento da regra.

6.2.1 Regras de Transição de Estados

As regras de transição de estados definem quais são as mudanças de estados que podem ocorrer em instâncias de um papel. A forma mais geral de uma regra de transição de estados é a seguinte:

r_n: $state(s_1), msg(m_i) \Rightarrow msg(m_0), state(s_2); (<condição\ de\ transição>)$

Esta regra tem o seguinte significado: quando a instância recebe a mensagem *mi*, se esta instância estiver no estado s_1 e se a condição de transição for satisfeita, então esta instância envia a mensagem *mo* e muda de estado, assumindo o estado s_2. Nenhum dos elementos acima é obrigatório na definição de uma regra de transição de estados. As regras de transição de estados definem a evolução dinâmica de um objeto. A chegada

de uma mensagem enviada por outra classe não significa que o método correspondente sempre irá ser executado - regras de transição de estados controlam as mensagens determinando a execução ou não do método.

As seguintes combinações podem ocorrer:

- o estado inicial s_1 não é definido – a regra será ativada sempre que chegar a mensagem *mi*, se a condição for verdadeira, independentemente do estado que a instância estiver desempenhando;

- a mensagem de chegada *mi* não é definida – a regra ativada sempre que a instância estiver no estado s_1, devendo a condição ser verdadeira;

- a mensagem de saída *mo* não é definida – ocorre uma transição de estado sem envio de mensagem;

- o estado final s_2 não é definido – a regra, quando ativada, envia alguma mensagem sem que ocorra uma transição de estado;

- a condição não é definida – a ativação da regra fica restrita à chegada da mensagem estando a instância no estado inicial.

Relativas à possibilidade de utilização de múltiplas mensagens de entrada e saída, duas formas alternativas podem ser utilizadas. Assim, a regra a seguir:

rn: state(s_1),msg(mi) ⇨ msg(mo_1), msg(mo_2),..., msg(mo_n), state(s2); (<condição de transição>)

define que, ao ocorrer uma transição, não somente uma mensagem, mas um conjunto de mensagens, é enviado. E a regra:

rn: state(s1),{ msg(mi1), msg(mi2),..., msg(mim)}⇨ msg(mo), state(s2); (<condição de transição>)

define que a transição vai ocorrer somente depois que chegarem todas as mensagens mi_1, mi_2,...,mi_m. A chegada destas mensagens pode ocorrer em qualquer ordem, em tempos diferentes.

Uma regra de transição pode ser acionada por uma tomada de decisão:

rn: state(s_1),decision(d_1) ⇨ msg(m0), state(s_2); (<condição de transição>)

Como exemplo de uma especificação em TF-ORM, é apresentada na figura 6-1 a definição de uma classe de agentes. Esta classe representa uma pessoa que trabalha em uma empresa e possui como propriedades do papel básico o nome da pessoa, o sexo, o cpf e o endereço. Um papel que a pessoa pode desempenhar na empresa é o de empregado. Este é identificado para a classe através do papel *Employee*. Algumas propriedades definidas para este papel são código do funcionário, salário e data de admissão. Os possíveis estados deste papel são iniciando o emprego, empregado, em férias. Um exemplo de mensagens que podem ser enviadas e recebidas são as seguintes: solicitação de novo salário (*new_salary*), valores iniciais do empregado (*initial_values*I).

```
agent class (
PERSON,
<Base_role,
  static properties = {
            (name, string),
            (gender, {"F", "M"}),
            (cpf, integer)
            },
  dinamic properties = {
            (address, string)
            },
  rules = {
            r1: msg(create_object) ⟹ state(active);
            r2: state(active) ⟹ msg(allow_role(employee)),
>,
<Employee,
  static properties = {
             (cod, integer) },
  dinamic properties = {
            (salary, real),
            (admission_date, date)
            };
  decision = {
            new_employee(Name:string, CPF: integer, Addr: string, Gdr: {F, M}, Dbirth: date,
            BenningDate: date, Salary:real, Function:{"manager", "attendance", "accountance",
            "support"}, HoursWeek: hours),
            ...
            };
  states = {
            beginning_employment,
            employed,
            in_vacations
            };
  messages = {
  initial_values(Rid:integer, Code: integer, CPF:integer, Name: string, Addr: string, Gender:{M, F},
            D-birth: date, Admission_Date: Date, Salary: real, Function: { "manager", "attendance",
            "accountance", "support"}, HourWeek: hours)
  from EMPLOYEE_CONTROL.New_Employee_Registration,

  new_salary(Emp: employee, Value: real, Valid_Time: date)
  to EMPLOYEE_CONTROL..General_Employee_Control
  ...
  };
  rules = {
            begin:
            msg(add_role) ⟹ state(beginning_employment),
            initialization:
            state(beginning_employment), msg(initial_values(Rid, Code, CPF, Name, Addr, Gender,
            D-birth, Admission_Date, Salary, Function, HourWeek) ⟹ state(employed),
        ...
            }
  >,
```

FIGURA 6-1: Exemplo de especificação em TF-ORM

6.3 Informações Temporais

O modelo TF-ORM foi criado com o objetivo de armazenar não somente os valores atuais dos dados, mas toda a sua história - valores passados, atuais e previsões de valores futuros; características importantes que permitem a possível avaliação e determinação da evolução do processo no *workflow*. Todos os estados assumidos por uma instância são permanentemente armazenados no banco de dados, com rótulos temporais que indicam quando foram efetuadas as definições. Com este objetivo, informações temporais foram associadas às informações em dois níveis: i) às instâncias, através de propriedades especiais (*object-instance* e *role_instance*), representando a evolução das instâncias como um todo; e ii) às propriedades que possam variar com a passagem de tempo, identificando quando os valores foram definidos e quando se tornaram válidos.

7. Representação do *Workflow* através do TF-ORM

Este capítulo trata das extensões realizadas no TF-ORM. Estas extensões foram desenvolvidas para que o TF-ORM procure representar de forma abrangente os aspectos relacionados a um *workflow*. Também neste capítulo é descrita uma técnica de modelagem de *workflow* utilizando o TF-ORM.

A modelagem de um *workflow* através do modelo TF-ORM é realizada pela definição de 3 tipos de classes (figura 7-1) presentes no modelo: classe de agente (*agent class*), de recurso (*recourse class*) e de processo (*process class*).

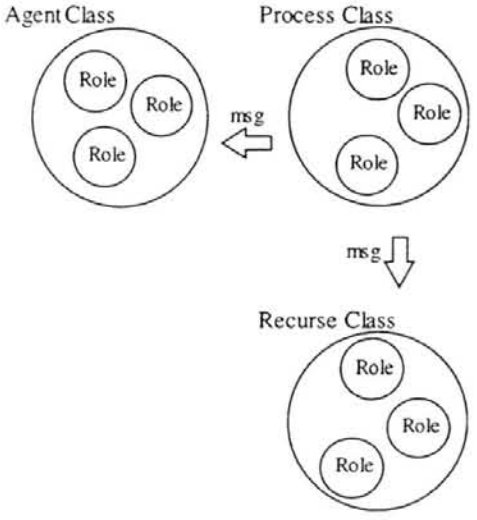


FIGURA 7-1: Classes do Modelo TF-ORM

A maioria dos modelos de *workflow* existentes não representam formalmente a parcela de trabalho que necessita de intervenção humana (trabalho não estruturado), determinante nos processos de tomada de decisão. O TF-ORM, permite a representação da parcela de trabalho não estruturada dos processos através da classe de agentes. A classe de agentes representa pessoas atuando no sistema que está sendo modelado. Os agentes possuem uma funcionalidade própria que é o poder de decisão humana. Uma decisão representa o resultado de um processo de tomada de decisão efetuado por um agente.

As estruturas dos recursos (dados, documentos) são representados por classes de recursos.

As classes de processos integram estes agentes e recursos, descrevendo a organização do trabalho desenvolvida na aplicação e a cooperação entre os agentes. Um processo particular, que é executado de acordo com o modelo de processo, é chamado de instância do processo. Portanto, é possível ter muitas instâncias de processos deste mesmo modelo de processo e cada uma delas pode apresentar estados diferentes em um determinado instante de tempo (representado no TF-ORM através de estados). Cada instância do processo contém um conjunto de atividades que determinam o andamento (evolução) do trabalho. A instância do processo é responsável pelo inicialização e término de atividades de trabalho de acordo com as regras determinadas na definição do processo de negócio. Ela é responsável pela transferência do controle entre as atividades

em um processo de acordo com as condições de transição e o roteamento da informação. Cada processo pode trabalhar com condições que permitem *Joins* e *Forks* que devem ser monitorados pelo sistema gerenciador do processo. Uma instância do processo permite mapear todas as atividades feitas por cada participante. Podendo dessa forma, serem coletadas e relatadas métricas de performance.

As atividades são executadas dentro do contexto funcional da estrutura organizacional da empresa, a qual define os papéis funcionais e os possíveis relacionamentos entre os agentes envolvidos [WFM 96a]. As atividades que compõem os processos são associadas aos papéis dos agentes, através da definição de responsabilidades pela execução das atividades. Um mesmo agente pode desempenhar mais de um papel diferente, simultaneamente ou não.

Os papéis das classes de recursos representam formas diferentes de considerar estes recursos, conforme os processos que estão sendo executados sobre eles. Estes conceitos estão resumidos na tabela 7-1. A visão geral da representação de um *workflow* através do TF-ORM é dado nas figura 7-2, figura 7-3, figura 7-4.

TABELA 7-1: Classes e Papéis em um *workflow*

Classe TF-ORM	**Significado desta classe no *workflow***	**Significado dos papéis desta classe no *workflow***
Processo	Processo a ser executado	Atividades que constituem os processos
Agente	Pessoas que executam processos	Papéis que uma pessoa pode desempenhar no *workflow*
Recurso	Recursos manipulados nos processos	Diferentes formas de visualizar estes recursos

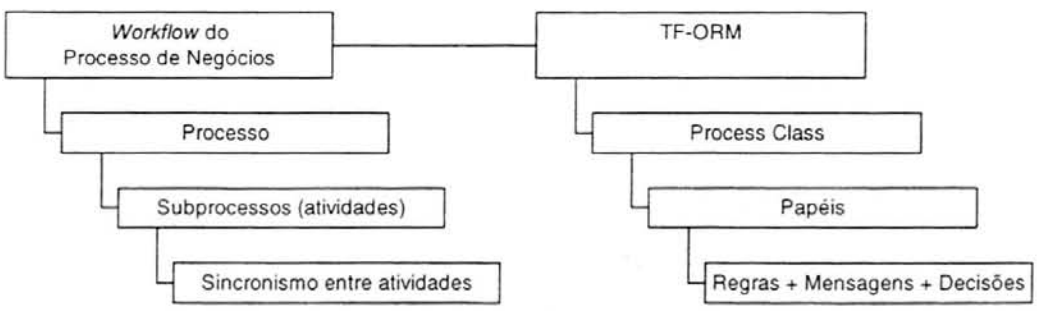


FIGURA 7-2: Mapeamento *Workflow* para TF-ORM (PROCESSO)

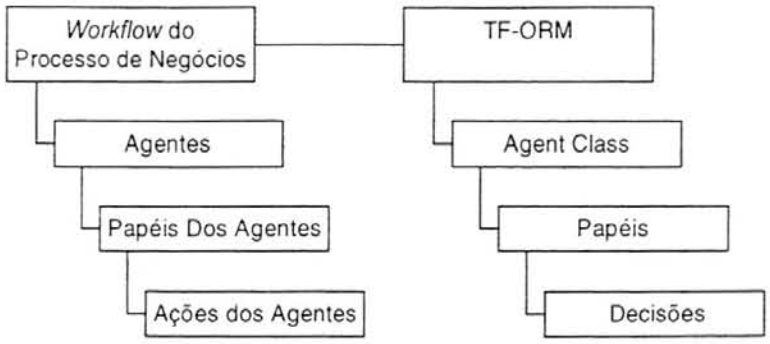


FIGURA 7-3: Mapeamento *Workflow* para TF-ORM (AGENTES)

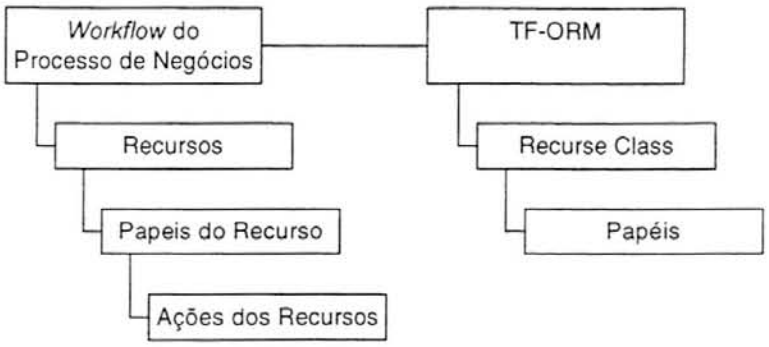


FIGURA 7-4: Mapeamento de *Workflow* para TF-ORM (RECURSO)

A correspondência entre conceitos do *workflow* e sua representação no TF-ORM é apresentada na tabela 7.2.

TABELA 7-2: : Classes de Processos de um *workflow*

Workflow	**TF-ORM**
Processo	Classe de processo
Atividade	Papel de uma classe processo
Agente responsável por uma atividade	Atributos na criação da instância do papel
Propriedades comuns a todas as atividades	Propriedades do papel básico da classe
Propriedades específicas de uma atividade	Propriedades do papel
Operações executadas em uma atividade	Mensagens e decisões do papel
Estados inicial e final de uma operação	Estados do papel
Sincronismo entre as atividades a serem executadas	Regras de transição de estados
Condições gerais de integridade	Regras de integridade

Uma vez definidas as classes de processo, a modelagem é completada identificando os agentes envolvidos nos processos (classes de agentes) e os recursos

manipulados (classes de recursos). As correspondências entre conceitos do *workflow* e sua representação em TF-ORM são apresentadas na tabela 7.3 – para as classes de agentes – e na Tabela 7.4 – para a classe de recursos.

TABELA 7-3: : Classes de Agentes de um *workflow*

Workflow	TF-ORM
Pessoa	Classe de agente
Pessoa executando uma atividade	Papel de classe de agente
Propriedades comuns a todos os comportamentos do agente	Propriedades do papel básico da classe
Propriedades específicas de um determinado comportamento	Propriedades do papel correspondente
Decisões que o agente pode tomar	Decisões do papel correspondente
Ações que o agente executar	Mensagens do papel
Estados inicial e final de uma operação	Estados do papel
Evolução do agente	Regras de transição de estados

TABELA 7-4: Classes de Recursos de um *workflow*

Workflow	TF-ORM
Recurso manipulado em um processo	Classe de recurso
Diferentes aspectos do recurso	Papel da classe de recurso
Propriedades comuns a todos os comportamentos do recurso	Propriedades do papel básico da classe
Propriedades específicas de um determinado comportamento	Propriedades do papel correspondente
Ações executadas sobre o recurso	Mensagens do papel
Estados inicial e final de uma operação	Estados do papel
Evolução do recurso	Regras de transição de estados

7.1 Conceito de Papel em *Workflow*

Em um *workflow*, um papel é um grupo de participantes que apresentam um conjunto específico de atributos, qualificações e/ou habilidades [WFM 96]. Para o TF-ORM, o conceito de papel representa um comportamento que uma pessoa ou grupo de pessoas podem apresentar na aplicação considerada. O papel desempenhado pelo

participante do *workflow* é representado no TF-ORM através da classe de agentes. Para cada papel pode ser definido um conjunto de propriedades (estáticas ou dinâmicas) que representam as características do participante no processo de negócios. Dessa forma o conceito definido pela [WFM 96] é análogo a modelagem do TF-ORM.

Na classe de processos, os papéis representam diferentes atividades a serem executadas no processo. Na classe de recursos os papéis representam os diferentes comportamentos que os recursos podem apresentar durante sua existência.

Também nesta situação torna-se relevante tratar a questão do formalismo. Alguns modelos apresentam limitações na representação formal do agente. O modelo de [JOO 94, 94a] define o papel desempenhado pelo agente documentando-o através de um rótulo; enquanto que os modelos descritos em [CAS 95] e [WFM 96b] apresentam um certo formalismo na definição dos papéis dos agentes, porém as propriedades relacionadas à definição destes são pré-definidas, o que restringe a flexibilidade na representação do papel. No TF-ORM pode-se definir formalmente qualquer conjunto de propriedades (estáticas ou dinâmicas) necessárias à representação do papel desempenhado por um participante do *workflow*.

7.2 Extensões na Classe de Agentes

Uma característica importante em um *workflow* é a representação da parcela de trabalho não estruturada (humana) através de agentes. Estes agentes podem ser responsáveis pelas tarefas ou executores destas. No caso da ausência de um deles, o agente existente assume a função dos dois, ou seja, se o executor estiver ausente, o responsável assume também o papel de executor e vice-versa. Em vista disso, foram realizadas neste trabalho algumas extensões para a classe de agentes. Estas extensões permitem especificar quais atividades o agente é responsável e/ou executor, bem como, todas as decisões tomadas por aquele agente em uma ou mais atividades. Esta estrutura do modelo, permite monitorar o processo do *workflow*, determinando quais agentes executam que atividades e as decisões tomadas por eles nestas. Deve ser observado que nesta representação o agente no TF-ORM não define necessariamente uma pessoa, mas sim um papel que esta está desempenhando no caso. Dessa forma, este papel pode representar um grupo de pessoas atuando no caso.

A figura é um exemplo da representação, na classe de agentes (**Class1**), de um papel (**Role1**) desempenhado por um agente. Os índices têm o seguinte significado: n – representa o processo em que o agente é executor, m – representa o processo em que o agente é responsável, m/n – representam o processo em que são tomadas decisões, i – representa o papel pertencente ao processo.

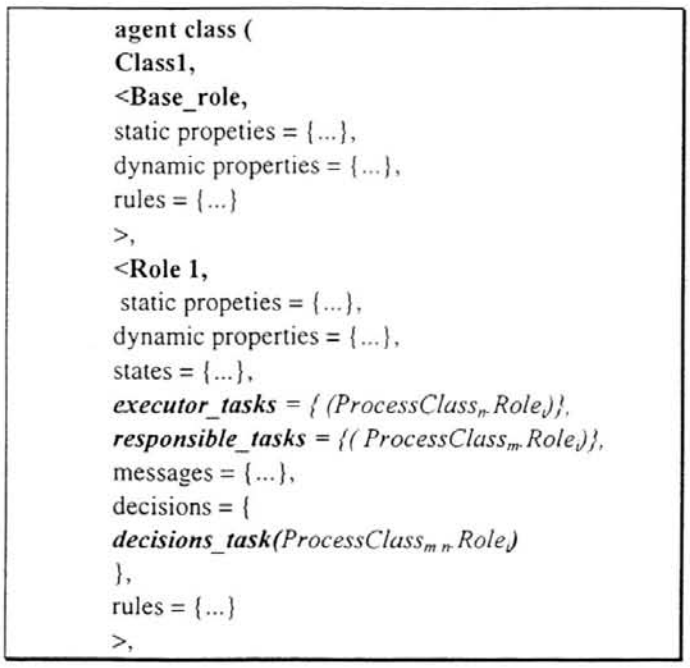


FIGURA 7-5: Representação da classe de agentes com executores e responsáveis

A sintaxe ***executor_tasks*** permite representar todos os processos (*ProcessClass*) e papéis (*Role*) nos quais o agente é executor. Um exemplo desta declaração seria:

executor_tasks = { (PAPERS_REVISION.Invited_referee),
(PAPERS_REVISION.Receive_analyse_paper)}

Esta declaração representa a situação em que o agente é executor do papel "Invited_referee" no processo "PAPERS_REVISION" e executor do papel "Receive_analyse_paper" no processo "PAPERS_REVISION". Este exemplo é retirado da modelagem do estudo de caso encontrado no anexo 2.

A sintaxe ***responsible_tasks***, permite representar todos os processos (*ProcessClass*) e papéis (*Role*) nos quais o agente é responsável. Um exemplo desta declaração seria:

responsible_tasks = {(PAPERS_REVISION.Paper_receive)}

Esta declaração representa a situação em que o agente é responsável pelo papel "Paper_receive", no processo "PAPERS_REVISION". Este exemplo é retirado da modelagem do estudo de caso encontrada no anexo 2.

A sintaxe ***decisions_task*** representa todas as decisões tomadas pelo agente no processo ($ProcessClass_{m/n}$) mais especificamente em um determinado papel ($Role_i$) deste processo. Um exemplo desta sintaxe, retirada do anexo 2, seria:

decisions_task(PAPERS_REVISION.Invited_referee):
{referee_paper_values(Referee_name: string, Cod_referee: string,
Cod_paper:string),
...,}

Esta declaração representa o conjunto de decisões (ex: referee_paper_values) que serão tomadas pelo agente executor no papel "Invited_referee", no processo "PAPERS_REVISION".

É, ainda, necessário fazer algumas observações com relação as decisões. Decisões tomadas por um agente em uma classe ou papel podem ser executadas novamente em outras classes ou papéis por este mesmo agente. Agentes diferentes podem tomar decisões iguais; contudo, a decisão precisa ser declarada para cada um dos agentes que a utiliza. A figura 7-6 exemplifica as situações mencionadas acima. Os papéis 1 e 2 (Role1 e Role2) possuem como agente executor a "secretária" que toma a mesma decisão de enviar o paper (sendmail(paper:text)) nos dois papéis. O papel 3 (Role3) possui como agente executor o "presidente do comitê" que toma a decisão sendmail(paper:text) neste papel. Para que este exemplo seja válido, a mensagem sendmail(paper:text) precisa ser declarada tanto na definição da secretária, quanto na definição do presidente do comitê.

```
process class (
Class1,
<Base_role,
static propeties = {...},
dynamic properties = {...},
rules = {...}
>,
<Role 1,
/*Agente responsável = Presidente do comitê */
/* Agente executor = secretária*/
static propeties = {...},
dynamic properties = {...},
states = {...},
messages = {...},
decisions = {
sendmail(paper:text)
},
rules = {...}
>,
<Role 2,
/* Agente executor = secretária*/
static propeties = {...},
dynamic properties = {...},
states = {...},
messages = {...},
decisions = {
sendmail(paper:text)
},
rules = {...}
>,
<Role 3,
/* Agente executor = Presidente do comitê */
static propeties = {...},
dynamic properties = {...},
states = {...},
messages = {...},
decisions = {
sendmail(paper:text)
},
rules = {...}
>,
```

FIGURA 7-6: Decisões tomadas pelo mesmo agente em papéis diferentes e decisões iguais tomadas por agentes diferentes

7.3 Extensões para a classe de Processo

Três extensões foram realizadas para a classe de processo, com o objetivo de adequar o modelo à modelagem de *workflow*:

i. definição do agente responsável e executor para uma tarefa ao criar a instância de um papel;

ii. listagem das decisões;

iii. definição do número de mensagens.

Estas extensões são descritas nas seções a seguir e foram incorporadas a BNF – anexo 1.

7.3.1 Definição de Responsabilidades

Em um *workflow*, a tarefa pode ser completamente automatizada (isto é, um sistema externo similar a um software ou uma máquina executa a tarefa de trabalho) ou pode ser designada para um agente (humano). Se a tarefa de trabalho apresenta uma destas características ou as duas, esta(s) deve(m) ser especificada(s) adicionando uma qualificação para a atividade. No TF-ORM representamos esta qualificação através de uma propriedade. Esta propriedade deverá ser representada na classe de processos, indicando se a tarefa é automática ou manual. A propriedade, apresenta as seguintes características:

i. se manual, o responsável (ou executor) pode ser uma pessoa ou uma classe;

ii. se automática, deve ser determinado um responsável a ser notificado em caso de exceções. Exceções, neste caso, seriam problemas que o sistema externo não conseguiu solucionar sozinho, necessitando assim de intervenção humana.

A possibilidade de ser adicionado ao papel um agente responsável e/ou executor na sua instanciação será permitida somente a classe de processos. Esta característica permitirá ao modelo uma maior flexibilidade, pois para cada instância do papel poderão ser definidos (dependendo o caso) agentes executores e/ou responsáveis diferentes. É permitida também a situação em que o agente deseja ou não delegar atividades a outros agentes (visto que apenas algumas pessoas da organização têm poder de determinar responsabilidades). A sintaxe é uma extensão da mensagem pré-definida *add_role* demonstrada abaixo:

add_role(Oid, R, Rid [,Resp] [,Exec] [,Deleg] [,Auto])

Esta mensagem cria uma instância do papel *R* da instância da classe identificada por *Oid*, sendo o identificador desta instância de papel devolvido através do argumento *Rid*. O argumento *Resp* identifica o responsável pela tarefa. O argumento *Exec* identifica o executor da tarefa. O argumento *Deleg* determina a quem responsabilidade da tarefa pode ser delegada. E finalmente o argumento *Auto* determina que a tarefa é automática (o argumento *Auto* está definido como palavra reservada na BNF – anexo 1).

Como vimos anteriormente, toda classe no TF-ORM possui um papel básico. Este

papel apresenta as propriedades herdadas por todas instâncias dos outros papéis. Definindo-se no papel básico os agentes executor e/ou responsável, os argumentos *Resp*, *Exec* e *Auto* não são obrigatórios na instanciação de um papel, pois podem ser herdados. Permitindo dessa forma, que os papéis do processo possuam agentes *default* alocados. Ainda, deve-se observar i) que os argumentos *Exec* e *Auto* são mutuamente exclusivos, pois não pode haver um processo que seja automático e que ao mesmo tempo tenha um humano executando-o, e que ii) se o agente executor e/ou responsável necessário a um dos papéis for diferente do definido no papel básico, obrigatoriamente deverá ser especificado o valor do argumento *Resp e Exec* na instanciação do papel. Determinando-se assim agentes executor e/ou responsável específicos para instâncias daquele papel.

Na instanciação de um papel, se apenas um argumento for representado entende-se que este argumento define o agente responsável que será também executor, não havendo assim a possibilidade de delegação. O argumento *Deleg* só poderá ser representado, quando tarefa manual, se forem definidos os argumentos *Resp* e *Exec*. Quando a tarefa for automática, *Deleg* só poderá ser representado se os argumentos *Resp* e *Auto* forem definidos.

7.3.2 Listagem das decisões

Outra extensão realizada consiste em permitir que decisões tomadas pelo agente, sejam representadas na classe de processos, o que antes era somente permitido a classe de agentes. O objetivo desta extensão é o de permitir monitorar e determinar interações dos agentes sobre os processos que estão sendo executados. Melhorando dessa forma o rastreamento e mantendo o formalismo necessário a representação dos participantes do *workflow* junto aos processos. Na classe de agentes como vimos no capítulo 6 pode-se verificar todas as decisões tomadas por aquele agente, bem como em quais tarefas estas decisões estão sendo tomadas.

As decisões são listadas na classe de processo através da sintaxe *decision* e são executadas através da sintaxe

decision "(" <decision name> (" < **message parameters>** **") ")".**

A figura 7-7 exemplifica as extensões descritas nesta seção. No papel Role1, o bloco de decisões contém as decisões tomadas pelo agente na instância de processo Role1. O bloco de regras contém um exemplo de uma mensagem pré-definida *add_role* para a criação de uma instância de role1 determinando o agente responsável (*agent_Class_5.Role_3*), executor (*agent_ Class_4.Role_1*), e o agente para o qual será delegada a execução da atividade (*agent_Class_5.Role_1*) no caso de ausência do agente executor ou problemas na execução quando automática.

```
process class (
Class1,
<Base_role,
static propeties = { }
dynamic properties = { }
rules = { r1: state(active), msg(create_object) ⇒ msg(allow_role(Role1)),
...
}
>,
<Role1,
states = { }
messages = { }
decisions = {
determine_paper_values (Cod_paper: string, Author_name: string, Paper: text, University: string)
...
}
rules = {
msg(add_role(Oid,role1,Rid, agent_Class_5.Role_3, agent_Class_4.Role_1,
        agent_Class_5.Role_1)) ⇒ state(active);
}
>,
```

FIGURA 7-7: Representação do Classe de Processos com decisões

7.3.3 Definição do número de mensagens

O TF-ORM permite determinar a ordem de execução das atividades através das regras de transição e das mensagens que as compõem. Uma extensão relacionada ao sincronismo está na figura 7-12 que representa a situação em que restringe o número de mensagens de chegadas necessárias para que ocorra uma transição. A sintaxe da regra é demonstrada abaixo:

$$st(s_1),\ \mathrm{K}\{msg(m_1),\ msg(m_2),\ ...,\ msg(m_n)\} \Rightarrow msg(m_o),\ st(s_2)$$

Esta regra representa a situação em que diversas mensagens (n) podem ser recebidas, mas quando um determinado número ($k < n$) destas mensagens tiver chegado, a próxima atividade será ativada. K, variando entre 0 e n, determina o número de mensagens necessárias para que ocorra a transição. Para este caso a chegada de mensagens pode ser aleatória.

7.4 Sincronização de tarefas

Processos de negócios estão sujeitos a mudanças contínuas. O modelo de *workflow* deve oferecer meios para se definir (ou redefinir) a seqüência na qual as atividades, relacionadas ao processo, devem ser executadas, bem como condições contendo regras associadas a essa seqüência de execução. Dessa forma, torna-se possível a coordenação da execução das atividades, respeitando não apenas a ordem planejada de execução das mesmas, como também o comprimento das dependências e pré-condições entre elas.

Este item mostra como é realizada a representação dos relacionamentos entre atividades através do TF-ORM permitindo, dessa forma, especificar a sincronização das atividades de um *workflow* através deste modelo.

7.4.1 Definição do sincronismo entre atividades

A definição do sincronismo entre atividades no TF-ORM é realizada através das regras de transição de estado. Como vimos no capítulo 6, as regras de transição de estados definem quais são as mudanças de estados que podem ocorrer em instâncias de um papel. A forma mais geral de uma regra de transição de estados é a seguinte:

r_n: $state(s_1), msg(m_i) \Rightarrow msg(m_0), state(s_2)$; (<*condição de transição*>)

Qualquer transição pode ser ativada em qualquer ordem ou momento desde que atenda as mensagens, estados e condições para realização da transição.

Em um *workflow* a ordem na qual as atividades são executadas determina a evolução do fluxo de trabalho. Observando as regras de transição de estados, verifica-se que esta evolução é representada pelas mensagens no TF-ORM. Dessa forma, a ordem de execução das atividades em um *workflow* é definida no TF-ORM pelas mensagens que podem representar as seguintes evoluções:

- sequencial
- paralela
- convergente (*Join*)
- divergente (*Fork*)
- condicional

7.4.2 Sequencial

Situação em que atividades são disparadas de forma seqüencial, obedecendo uma ordem de execução. Duas atividades são totalmente seqüenciais quando a primeira (a_1) termina sua execução e ativa a execução da seguinte (a_2) – cria uma instância do papel que representa a segunda atividade. Nas regras de transição da primeira atividade, uma regra define a transição para seu estado final (s_f) e a criação da instância da outra atividade do processo pr_i (processo atual ou outro processo), ao ser recebida a mensagem m_1:

$st(s_1), msg(m_1) \Rightarrow msg(add\_role(<pr_i>, < a_2>)), st(s_f)$

Na figura 7-8 utilizamos uma representação gráfica para a sintaxe apresentada.

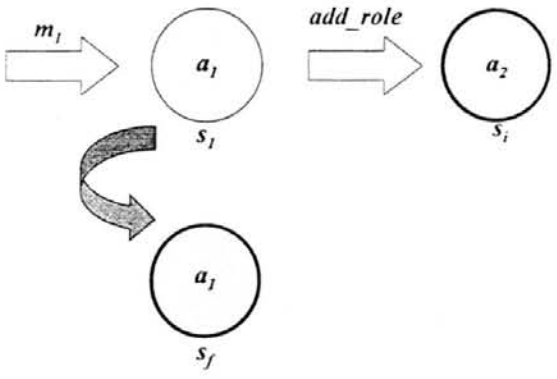


FIGURA 7-8: Representação Gráfica da Sincronização Seqüencial de Atividades

A figura tem o seguinte significado: quando a instância a_1 recebe a mensagem m_1, se esta instância estiver no estado s_1, então esta instância envia a mensagem *add_role* para a atividade a_2 e faz a transição para o seu estado final s_f.

7.5 Paralela

Uma atividade, durante sua execução, pode disparar a execução de outra atividade (do mesmo ou de outro processo). A execução da primeira atividade segue normalmente, ocorrendo sua transição para o estado s_2. A partir deste momento, a execução das duas atividades corre em paralelo (figura 7-9). Esta situação é representada pela seguinte regra da primeira atividade:

$st(s_1), msg(m_1) \Rightarrow msg(add\_role(<pr_i>, < a_2>)), st(s_2)$

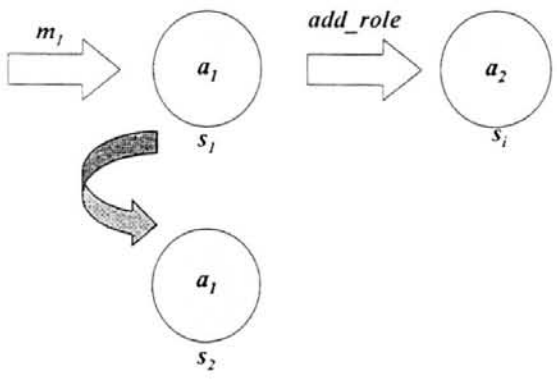


FIGURA 7-9: Ativação de atividade paralela

7.6 Paralelas, com execuções sincronizadas

Duas atividades podem executar paralelamente, de forma sincronizada – uma parte da primeira atividade, quando completada, envia uma mensagem (m_2) e ativa a segunda, a qual está em um estado de espera (s_x); a primeira atividade, uma vez feita a transição, passa a outro estado (s_2), o qual pode corresponder à continuação de sua execução, ou à espera da chegada de outra mensagem que a reative (figura 7-10):

$st(s_1), msg(m_1) \Rightarrow msg(m_2), st(s_2)$

No conjunto de regras da segunda atividade, uma regra deve ser ativada para tratar desta mensagem e retomar sua atividade:

$st(s_x), msg(m_2) \Rightarrow st(s_y)$

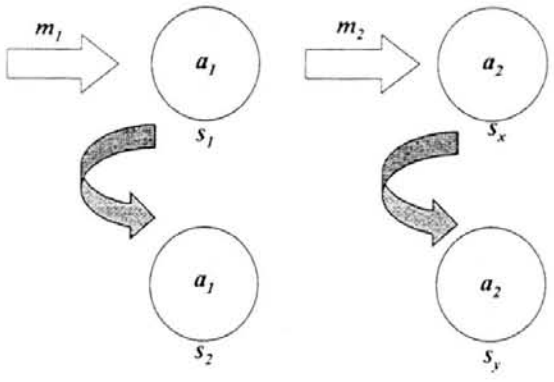


FIGURA 7-10: Atividades paralelas, sincronizadas

7.6.1 Convergente Total (*Join* Total)

Uma atividade pode ser ativada em função de diversas mensagens recebidas (obrigatoriamente todas mensagens devem chegar), possivelmente de atividades diferentes (figura 7-11). Quando não importa a ordem de chegada destas mensagens, esta situação é representada pela seguinte regra:

$$st(s_1), \{msg(m_1), msg(m_2), ..., msg(m_n)\} \Rightarrow msg(m_o), st(s_2)$$

onde a transição de estado ocorre somente quando for recebido o conjunto de mensagens m_1 a m_n. O envio da mensagem m_o ativa a atividade seguinte.

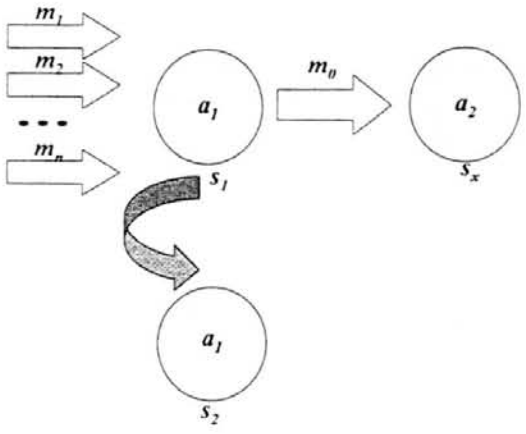


FIGURA 7-11: Representação Gráfica da Sincronização Convergente Total de Atividades

Quando a ordem de chegada das mensagens é definida, ela determina diversas regras de transição.

7.6.2 Convergente Parcial (*Join* Parcial)

Podem acontecer situações em que diversas mensagens (n) podem ser recebidas, mas quando um determinado número ($k < n$) destas mensagens tiver chegado, a próxima atividade será ativada (extensão realizada conforme item 7.3.3). K, variando entre 0 e n, determina o número de mensagens necessárias para que ocorra a transição. Também para este caso a chegada de mensagens pode ser aleatória. A regra que representa este sincronismo é a seguinte:

$$st(s_1), K\{msg(m_1), msg(m_2), ..., msg(m_n)\} \Rightarrow msg(m_o), st(s_2)$$

A figura 7-12 representa a situação para o caso de K=2 (somente duas mensagens são necessárias para que a transição ocorra.

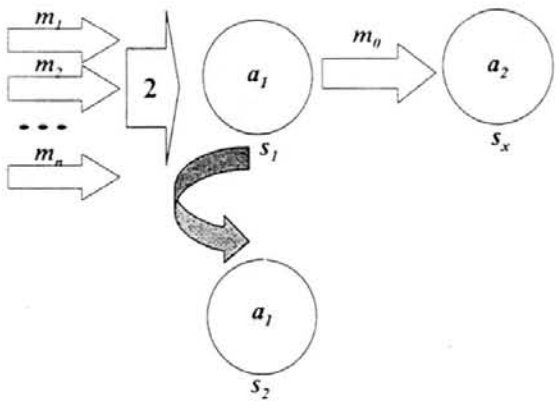


FIGURA 7-12: Representação Gráfica da Sincronização Convergente Parcial de Atividades

7.6.3 Divergente Total (*Fork*)

Sincronismo divergente para atividades corresponde à situação em que diferentes atividades podem ser ativadas simultaneamente por uma mesma atividade (figura 7-13). A regra que representa esta situação é a seguinte:

$st(s_1) \Rightarrow msg(m_1), msg(m_2), \ldots, \text{msg}(m_n), st(s_2)$

onde a atividade que ativa as outras atividades (a_1) está no estado s1.

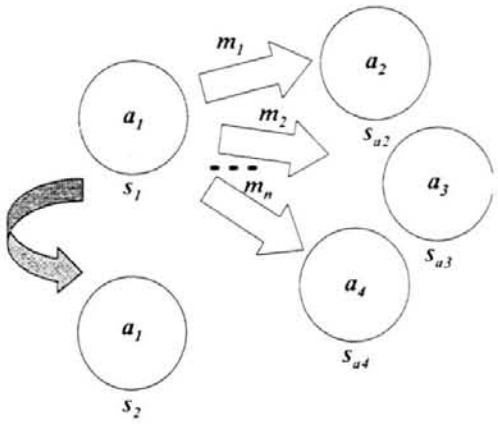


FIGURA 7-13: Representação Gráfica da Sincronização Divergente de Atividades

7.6.4 Condicional

A definição de uma condição associada a uma regra de transição torna a ativação da(s) atividades(s) seguinte(s), feita pelo envio de mensagens ao ser realizada a transição, condicional. Qualquer regra pode ser restringida através de uma condição, residindo neste aspecto um dos mais importantes potenciais deste formalismo. A regra abaixo mostra uma situação em que a transição (e o conseqüente envio da mensagem m2) somente acontece quando a condição *cond* for verdadeira:

$st(s_1), msg(m_1) \Rightarrow msg(m_2), st(s_2); (<cond>)$

Através da utilização condições, mensagens diferentes, possivelmente para atividades diferentes, podem ser enviadas. Por exemplo, se a instância recebe a mensagem m_1, e estiver no estado s_1 e se a condição de transição (<cond1>) for satisfeita, então esta instância envia as mensagem m_2 mudando de estado, assumindo o estado s_2. Se, no entanto, a condição de transição (<cond2>) for satisfeita, então esta instância envia as mensagem m_3 mudando de estado, assumindo o estado s_3. A situação é demonstrada a seguir (figura 7-14, representação gráfica):

$st(s_1)$, $msg(m_1) \Rightarrow msg(m_2)$, $st(s_2)$; (<cond1>)

$st(s_1)$, $msg(m_1) \Rightarrow msg(m_3)$, $st(s_3)$; (<cond2>)

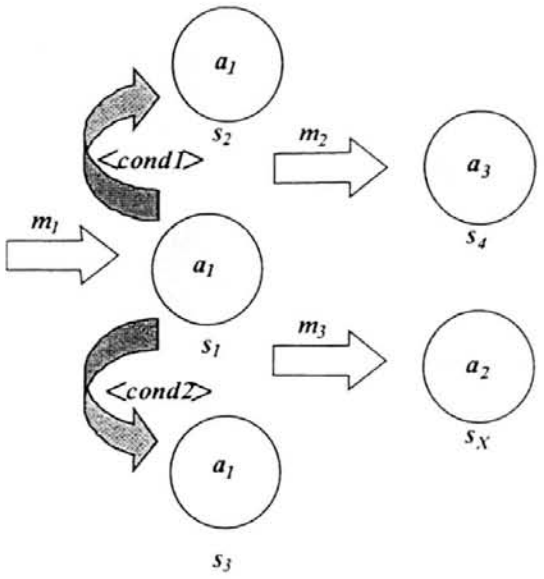


FIGURA 7-14: Sincronismo Condicional de Atividades

Poderia ainda haver a situação em que as duas condições são satisfeitas. Dessa forma, haveria a necessidade de uma regra que previsse esta situação, como demonstrado abaixo:

$st(s_1)$, $msg(m_1) \Rightarrow msg(m_3)$, $st(s_4)$; (<cond1> and <cond2>)

Neste caso, se a instância recebe a mensagem m_1, e estiver no estado s_1 e se as condições de transição (<cond1>) e (<cond2>) forem satisfeitas, então esta instância envia as mensagem m_3 mudando de estado, assumindo o estado s_4.

Qualquer um dos sincronismos citados neste capítulo, podem ser condicionados, pela definição de uma condição de transição.

7.7 Aspectos Temporais

Os modelos de *workflow* precisam disponibilizar expressões relacionadas aos processos, restrições temporais, trocas dinâmicas e tratamento de exceções [ELL 96] o que nem sempre acontece. Técnicas de Modelagem Temporal permitem a representação de muitos dos aspectos citados acima. Através destas técnicas são representadas as características dinâmicas das aplicações e a interação temporal entre diferentes processos. A possibilidade de armazenar, manipular e recuperar dados temporais deve

ser considerada quando da escolha de um método de modelagem de *workflow*.

Normalmente um *workflow* necessita de controles de tempo em um processo. Estes controles permitem disponibilizar lembretes (ex: *Primeiro Lembrete "depois de uma data x", Repetir Lembretes "todo data x",Tempo Limite "termina em data x"*) no *workflow* para automaticamente alertar os participantes deste sobre atividades pendentes, bem como disparar outras atividades relacionadas com tempo. No TF-ORM estes controles são representados pelas regras de transição de estados, mais especificamente na condição de transição. As condições definidas nas regras de transição de estados são expressas através de sentenças de uma linguagem temporal de primeira ordem [EDL 93]. Nestas condições podem ser avaliados valores de propriedades (presentes, passados, futuros) e estados de papéis. Os identificadores de instâncias de papéis podem também ser referenciados, pois estão armazenados nas propriedades pré-definidas *oid* e *rid*.

Podemos representar através de operadores e funções, situações de um processo no *workflow*, como por exemplo, o *deadline* de um artigo. Veja o exemplo abaixo:

```
process class (
        Class1,
        <Base_role,
        static propeties = {...};
        dynamic properties = {(deadline, date)};
        rules = {...}
        >,
        <Role1,
        states = {...};
        messages = {...};
        decisions = {...};
        rules = {
        end_of_date:
        state(active) ⟹ msg(send_paper(paper)), state(active);
        (exists Rid (has_role_instance(Oid, paper, Rid)) and
        (value(Rid, upper_bound(deadline))= now),
        ...
        }
        >
```

A regra *end_of_date* descreve a seguinte situação: se a instância estiver no estado active e se a condição de transição for satisfeita, então esta instância envia a mensagem *(send_paper(paper))* permanecendo no estado *active*. A mensagem *(send_paper(paper))* dispara uma atividade que tratará o término do prazo de envio de artigos. A condição de transição *exists Rid (has_role_instance(Oid, paper, Rid)* verifica se existe uma instância do objeto paper criado; e a condição *(value(Rid, upper_bound(deadline))= now)*, verifica se o limite máximo do deadline para este objeto equivale à data de hoje.

7.8 Metodologia

Esta seção trata da técnica de modelagem que deve ser utilizada para representação do *workflow* através do TF-ORM. Algumas das etapas, conceitos e notações usadas nesta técnica foram baseadas em [RUM 91,94, SCM 96], sofrendo adaptações próprias à técnica. Esta técnica de modelagem conceitual é a base para alcançar a especificação e implementação de *workflow*, procurando facilitar o processo de análise da aplicação e de representação da evolução dos objetos no processo [NIO 98].

7.8.1 Análise

Como toda metodologia de modelagem de sistemas de informação, o processo de análise para um *workflow* preocupa-se com a compreensão e a modelagem da aplicação e do domínio em que ela atua. A entrada da fase de análise compreende um enunciado que descreve o problema a ser solucionado e oferece uma visão geral conceitual do sistema proposto. A saída da análise é um modelo formal, no caso o modelo TF-ORM, que incorpora os quatro aspectos essenciais do sistema: os objetos, os relacionamentos entre eles, o fluxo dinâmico de controle e a transformação funcional dos dados sujeita a restrições. Ainda, nesta fase, três etapas precisam ser observadas:

i. definir o enunciado do problema - Esta situação compreende um enunciado que descreve o problema a ser solucionado e oferece uma visão geral conceitual do sistema proposto. O enunciado do problema não deve ser visto como imutável e deve servir como base para o refinamento dos verdadeiros requisitos.

ii. abstrair características no modelo - O sistema do mundo real descrito na definição do problema precisa ser compreendido, e suas características essenciais abstraídas no modelo.

iii. definição dos Objetivos - Através do enunciado do problema, deve-se estabelecer o que deve ser feito e não como deve ser feito. Deve-se definir as necessidades e não uma proposta de solução.

7.8.2 Modelagem dos Objetos

Para descrever um processo de negócios, é necessário definir as ações ou atividades do processo. Em adição, as pessoas que executam estas atividades precisam ser definidas assim como os recursos que são usados. Portanto, o modelo de representação do *workflow* deve permitir a definição das características dos objetos que compõem o processo de negócios.

Considerando que o TF-ORM é um modelo de dados orientado a objetos, devem ser identificadas, no processo de negócios avaliado, as classes dos objetos e os relacionamentos existentes entre elas. Neste caso, o mais importante é a organização dos níveis mais elevados do sistema de classes interligadas por associações (mensagens); as partições de níveis mais baixo (papéis) tornam-se menos críticas pela possibilidade de herança (papel básico). Dessa forma, as etapas para modelagem dos objetos no *workflow* e seu equivalente no TF-ORM é apresentado na tabela 7-5.

TABELA 7-5:Fases de modelagem do *workflow* e seu equivalente no TF-ORM

Workflow	TF – ORM
1. definir atividades ou tarefas que um grupo de trabalho necessita executar e as regras de negócio que governam a atividade	1. identificar no TF-ORM a classe que representa os processos. • determinar propriedades (estáticas e dinâmicas) da classe
2. dividir as atividades em subatividades (subtarefas). Cada atividade representa uma lista bem definida de passos que devem ser executados para realizar a tarefa. • Decidir o conjunto de habilidades necessárias para executar cada tarefa (papel do agente). Isto especifica o conjunto de funções ou pessoas que podem ser definidas para executar a tarefa.	2. definir as classes de agentes, de recursos e de processos e os respectivos papéis que as compõem (o processo deve possuir o agente responsável e agente executor definido para cada atividade que o compõe). • definir propriedades dos papéis (que não estão definidas no papel básico, pois estas serão herdadas) • definir as decisões tomadas pelos agentes na classe de agentes
3. identificar relacionamento entre classes (definir a seqüência na qual as atividades devem ser executadas). Se algumas dessas tarefas são executadas com base em alguma condição (de forma condicional), identificar as tarefas e definir as condições.	3. - definir as mensagens - definir as regras de transição e as condições (deve-se observar que *workflow* envolve uma seqüência de tarefas ou um processo, e que a comunicação (fluxo) entre tarefas e subtarefas é baseada sobre um conjunto de regras e condições pré-definidas).
4. definir a evolução dos processos/atividades (estados).	4. definir estados dos agentes, recursos e processos.

7.8.3 Análise da Modelagem do TF-ORM através dos critérios fundamentais

A partir da modelagem do estudo de caso, realizado para determinação da técnica de modelagem de *workflow* utilizando o TF-ORM, podemos identificar quais dos critérios fundamentais que devem compor um modelo de *workflow* estão presentes no TF-ORM. Dessa forma, a tabela 7-6 representa os resultados obtidos da análise:

TABELA 7-6: Análise do TF-ORM segundo critérios fundamentais que devem compor um modelo de *workflow*

Critérios Fundamentais	**TF-ORM**
workflow* do tipo *ad hoc	representa
***workflow* administrativo**	representa
***workflow* de produção**	representa
formalismo apresentado pelos modelos	formal
formalismo na definição de papéis	formal
representação formal de tarefas automáticas	representa formalmente (através da classe de processos)
formalismo na representação das atividades	representa formalmente (através da classe de processos)
representação da estrutura do objeto	representa formalmente (através das classes)
representação da estrutura do fluxo	representa formalmente
representação de aspectos relacionados a tempo	permite definição explicita
representação formal do tratamento de exceções	representa formalmente (através de regras de transição de estados)

A partir desta análise, podemos concluir que o TF-ORM apresenta-se como um modelo adequado para representação formal do *workflow*.

8. Modelagem do Estudo de Caso

A representação de um estudo de caso através de ferramentas comerciais de *workflow* tem como objetivo analisar e validar a modelagem realizada através do TF-ORM comparando-a com diferentes metodologias. Procurou-se dessa forma, determinar que características das ferramentas (e metodologias) são suportadas ou não pelo TF-ORM e vice-versa. As características não suportadas pelo TF-ORM procuraram ser incorporadas de maneira a torná-lo um modelo mais completo à modelagem de *workflow*.

As ferramentas comerciais foram selecionadas por caracterizarem, cada uma, as categorias básicas de modelagem de *workflow*:

i) *Process Builder* caracteriza a metodologia de modelagem de *workflow* baseada em comunicação e

ii) *JetForm Workflow Builder* a metodologia de modelagem de *workflow* baseada em atividades.

Dentro dos tipos de sistemas de *workflow* existentes (capítulo 3), procuramos comparar as metodologias (ferramentas e TF-ORM) utilizando um estudo de caso que representa os tipos de *workflow ad hoc* e administrativo. Os tipos de *workflow* foram selecionados por:

i) *ad hoc* - envolve humanos na execução e coordenação de tarefas, permitindo assim que o modelo de representação do *workflow* demonstre sua capacidade em representar a parcela não estruturada de trabalho;

ii) administrativo - escolhido por ter características mistas, ou seja, controla e coordena tarefas automatizadas com pequena intervenção humana.

8.1 Descrição do Estudo de Caso

O estudo de caso escolhido para validação do modelo é o da revisão de artigos para uma conferência [ANT 91]. Duas implementações distintas foram feitas: uma para a situação que representa um *workflow* do tipo ad hoc e outra para um *workflow* do tipo administrativo. O caso geral consiste no seguinte procedimento: O Presidente do Comitê de Programa seleciona revisores (no máximo 3), e distribui artigos aos revisores selecionados. Estes avaliam ou analisam os artigos enviam um parecer ao Comitê de Avaliação. Para cada umas das implementações, *ad hoc* e administrativo, a análise tem procedimentos diferentes, descritos a seguir.

8.1.1 Modelagem Ad hoc

Na modelagem *ad hoc*, o Presidente do Comitê de Programa convida cada revisor a participar do processo de revisão. Os potenciais revisores respondem de forma afirmativa ou negativa. Caso o revisor aceite o convite, o Presidente do Comitê de Programa então envia o artigo ao revisor e espera o retorno. O revisor, por sua vez, revisa o artigo e devolve ao Comitê de Avaliação seu parecer. Caso o revisor não aceite o convite, o Presidente do Comitê de Programa convida outro revisor. Algumas regras precisam ser observadas para a representação do caso: i) um mesmo revisor não deve ser convidado duas vezes; ii) revisores podem enviar artigos, porém não podem revisá-los.

Basicamente as seguintes atividades são realizadas pelos participantes do *workflow*:

- **Presidente do Comitê de Programa**

  1. convidar revisor;
  2. enviar artigo ao revisor;
  3. receber considerações sobre o artigo.

- **Revisor**

  1.enviar resposta do convite ao presidente do comitê de programa

  2.revisar artigo

  3.enviar parecer ao comitê de avaliação

8.1.2 Modelagem Administrativa

Na modelagem administrativa, o Presidente do Comitê de Programa submete cada artigo à avaliação de revisores. Os revisores são pré-determinados, e classificados conforme a especialidade. Cada revisor avalia o artigo, atribuindo um conceito ao mesmo. Após avaliação, o parecer é enviado ao Presidente do Comitê de Programa. O Presidente do Comitê de Programa faz a revisão dos pareceres, enviando o parecer final ao autor. Na situação em que os pareceres apresentarem avaliações muito diferenciadas, o Presidente do Comitê de Programa poderá solicitar uma nova avaliação.

As atividades dos atores são listadas abaixo:

- **Presidente do Comitê de Programa**

  1.enviar artigo a revisores pré-determinados;

  2.receber considerações sobre o artigo;

  3.fazer avaliação final

  4.solicitar nova avaliação.

- **Revisor**

  1.revisar artigo;

  2.enviar parecer ao presidente do comitê de programa.

8.2 Descrição das Ferramentas utilizadas

Esta seção descreve a metodologia e as características das ferramentas JetForm *Workflow Builder* (JetForm Corporation) e *Process Builder* (Action Technologies, Inc.) utilizadas na modelagem do estudo de caso definido neste trabalho.

8.3 Metodologia utilizada pela JetForm *Workflow*

Nesta seção são descritas características e a metodologia utilizadas pelo *JetForm Workflow Builder* na representação de um *workflow*.

O *JetForm Workflow Builder* gera um modelo de *workflow* baseado em atividades (capítulo 4) e apresenta as seguintes características na representação de um *workflow*:

- projeta graficamente o processo de *workflow*;

- utiliza uma ferramenta gráfica que constrói um *workflow* como uma série de tarefas desenhadas juntas, similar a um *Flow Chart*;

- pode-se assumir tarefas para papéis ou posições (cargos) na organização;

- pode-se decidir que ações podem ser executadas para cada tarefa tão bem quanto as instruções específicas que podem ser definidas para o executor daquela tarefa.

8.3.1 Passos

Os seguintes passos representam a metodologia utilizada pela JetForm *Workflow* para criar *workflows*.

1. executar uma análise do negócio – consiste em i) identificar o processo de negócios, ii) e analisar cada um dos processos para definir tarefas, ações, rotas, etc.;

2. identificar os participantes do *workflow* – consiste em i) assegurar que os dados necessários para a definição dos atores (participantes) do *workflow* estejam disponíveis, ii) definir o conjunto de papéis a ser usado pelo processo de negócios representado por cada *workflow* ;

3. escolher os formulários que irão tramitar pelo *workflow* – consiste em decidir que formulários irão iniciar o *workflow*, e serão usados pelos atores para executar os trabalhos de cada tarefa;

4. criar o workflow;

5. organizar o *workflow* para teste e produção – quando o projeto e a descrição do *workflow* estiver completa, deve-se testá-lo em nos ambientes de teste e produção;

6. monitorar e rastrear o *workflow* – monitorar e rastrear o progresso das transações do *workflow*, verificando se elas estão sendo executadas conforme determinado.

8.3.2 Tarefa no JetForm

Para o JetForm as tarefas são os blocos de construção básicos de um *workflow*. Relacionado a uma tarefa no JetForm estão os conceitos apresentados na tabela abaixo:

TABELA 8-1: Conceitos segundo o JetForm *Workflow* Builder

Conceito	Descrição
Estímulo (*stimulus*)	a razão da tarefa estar sendo executada. Por exemplo, a tarefa foi iniciada por uma pessoa ou programa que tinha completado outra tarefa, ou submetido um formulário que tinha iniciado uma transação.
Trabalho (*work*)	o trabalho que está sendo executado por aquela tarefa e o formulário que será usado para fazê-la.
Papel (*role*)	o nome do papel que será usado para determinar o endereço de *e-mail* do ator que executará a tarefa.
Lembretes (*reminders*)	mensagens de *mail* que podem ser enviadas para um ator (mas não para um programa de computador) quando o trabalho não é executado em tempo específico.
Deadlines	quando o *deadline* expira, os dados da transação são removidos das caixas de *mail* dos atores e a transação continua, usando a ação que foi especificada para o *deadline*.
Lista de ações (*action list*)	uma lista de ações associadas, que os atores podem escolher para completar o trabalho.
Rotas (*routes*)	uma lista de rotas associadas que definem como os dados são passados entre tarefas.

Uma tarefa pode ser composta de uma ou mais ações. Uma ação é uma escolha apresentada para um ator para completar uma tarefa e para controlar como as tarefas serão roteadas. Ações são usadas pelos atores para executar o trabalho das tarefas. As ações possuem propriedades que providenciam detalhes adicionais para a tarefa selecionada. As propriedades das ações são: i) o nome da ação e a descrição da ação, ii) se a ação é do tipo *deadline*, iii) como a ação será usada, isto é, normal, delegação, consulta, nominação, ou oculta, iv) e como as regras de roteamento das ações serão avaliadas, isto é, seqüencialmente ou em paralelo.

As ações das tarefas relacionam-se através de rotas. Uma rota é o caminho de uma ação, em (de) uma tarefa, para uma subsequente tarefa. Uma tarefa pode ter mais que uma rota associada, cada uma definindo uma possível tarefa subsequente.

No *JetForm*, pode-se especificar como as rotas para ações, com um ou mais papéis, serão avaliadas. Poderão ser avaliadas em seqüência ou em paralelo. Quando escolhe-se por um roteamento seqüencial, as regras serão avaliadas na ordem que forem listadas em uma Tabela de Roteamento definida nas propriedades da tarefa. Quando escolhe-se roteamento paralelo, cada regra de roteamento será avaliada. Todas as regras de roteamento que forem avaliadas como verdadeiras (true) serão disparadas. Além disso, é permitido habilitar determinadas propriedades de roteamento para as rotas. As propriedades de roteamento são usadas para criar e modificar rotas para ações específicas. Para cada rota, identifica-se a ação para a qual ela pertence e a próxima tarefa no *workflow*. Cada rota é definida por uma regra de roteamento, ou uma expressão Booleana. Se a expressão avaliada retornar verdadeiro (TRUE), a rota será utilizada. Se a expressão avaliada retornar falso (FALSE), a rota não será utilizada. Pode-se criar rotas de duas maneiras:

1. utilizando um construtor de regras (Rule Builder) para entrar com um conjunto de condições conectadas por operadores "e" (AND) ou "ou" (OR). Cada condição compara um campo do formulário ou um valor da tarefa com outro valor;
2. para construir uma regra mais complexa, pode-se digitar uma expressão Booleana diretamente em *BasicScript*. Se for utilizado *BasicScript*, não se poderá utilizar o construtor de regras..

8.3.3 Objeto

Os objetos em um *workflow* representados no JetForm incluem tarefas, ações, linhas de roteamento, e o próprio *workflow*.

Um objeto encapsula dados (propriedades) e rotinas (métodos) em uma única unidade. Um objeto é uma função ou rotina em BasicScript. Propriedades podem ser definidas como de leitura ou de escrita (buscar ou habilitar) ou ambos.

8.4 Metodologia utilizada pela Action *Workflow* Tecnologies

Esta seção trata brevemente de algumas das características e da metodologia, apresentadas pela ferramenta de modelagem de *workflow* da Action Tecnologies, ou seja, o Process Builder Analyst Edition.

O Process Builder apresenta ferramentas para criar mapas de processos de negócios e estabelecer papéis e definições que serão utilizados na criação de aplicações de *workflow*, além do estabelecimento de formulários disponíveis aos participantes do *workflow* na aplicação final.

A criação de um processo de negócios requer, além de definições gerais sobre o processo, uma descrição de seu funcionamento e a determinação do modelo que mais se adequa a aplicação. Para isso, o Process Builder permite a descrição do processo de negócios através de um conjunto de propriedades pré-definidas. A propriedades são:

1.nome do processo de negócios;

2.autor do mapa;

3.tempo limite para a conclusão do processo;

4.tempo total necessário à conclusão do processo de negócios - considerando que

todos os subprocessos devem estar completos ao término do *workflow* primário, o tempo calculado do ciclo é igual ao tempo total para o *workflow* primário, ou seja, a soma do tempo para cada uma das quatro fases;

5. modelo padrão a ser utilizado - modelos são conjuntos de definições pré-estabelecidas aplicáveis a *workflows*. quando um *workflow* é baseado em determinado modelo, automaticamente inicia de acordo com as definições deste;

6. grupo do processo de negócios - quando uma aplicação para o usuário final é baseada em mais de um mapa, todos eles podem receber o mesmo valor neste campo, possibilitando ao programador utilizar as apis (*application programming interface*) para fazer perguntas que façam referência a todos eles;

7. horas de trabalho por dia;

8. efeito da conclusão do *workflow* primário - quando esta opção não está selecionada, a propriedade *bpstatename* assumirá o valor "bps_completed" ao encerramento do *workflow* primário. caso contrário, o valor "bps_in_progress" será assumido.

8.4.1 Metodologia

A *ActionWorkflow* utiliza um método sistemático para projeto e reengenharia de processos de negócios. Coordena as atividades das pessoas dentro de um sistema de *workflow*, concentrando-se em cada participante a fim de alcançar o objetivo global do processo - satisfação do cliente. O responsável por um processo de negócios é considerado o cliente deste processo. Esta metodologia define como a atividade de cada participante contribuirá para alcançar o objetivo global e suporta inovações para possibilitar aumento de qualidade e redução de tempo e custo.

A metodologia da *ActionWorkflow* faz distinção entre papéis num processo de negócios e papéis organizacionais. Nos mapas, são utilizados papéis no processo de negócios, a partir dos quais serão especificados os participantes no *workflow*. Os mapas são projetados para serem adaptáveis a diferentes organizações ou divisões dentro de uma organização, portanto estes papéis não são específicos à organização. Após a aplicação ser gerada, é necessário, então, o mapeamento dos papéis no processo de negócios para papéis específicos na organização, através do *ActionWorkflow Administrator*.

Papéis (*Roles*) são utilizados para atribuir aos participantes do *workflow* as funções que eles desempenham. Portanto, ao iniciar um novo processo de negócios, faz-se necessária a criação de papéis, a fim de possibilitar esta atribuição. A adição ou modificação de papéis pode ser efetuada a qualquer momento, mas pode ser mais conveniente adicioná-los antes de criar o mapa.

Uma vez adicionados ao processo de negócios, os papéis podem ser utilizados para especificar os participantes do *workflow*. Para qualquer um dos três tipos de participantes de *workflow* (cliente, ator ou observador), é possível estabelecer um lista de papéis, contendo todos os papéis aptos a assumirem a função em questão.

Após estabelecidas as definições do processo de negócios, passa-se a criação do mapa. Todo mapa possui ao menos um *workflow* primário, que determinará o objetivo do processo de negócios. A cada *workflow* estão associadas uma série de informações.

8.4.2 Estrutura do Modelo

O *workflow* principal, que tem em suas condições de satisfação o propósito do processo de negócios, é denominado *workflow* primário. Quando estas condições são encontradas, a satisfação é declarada, o *workflow* primário é dito completo e o processo de negócios encerrado. Os outros elementos do processo de negócios são *workflows* filhos (ou netos, ou bisnetos...) ou componentes do *workflow* primário, e suas funções são um subconjunto do propósito geral do processo de negócios. Para o alcance das condições de satisfação pelo *workflow* primário, faz-se necessário que todos os *workflows* filhos estejam completos.

Em todo *workflow*, o cliente faz uma solicitação (*request*) de trabalho ao ator ou aceita uma oferta (*offer*) de serviço feita por este. Na solicitação, o cliente inicia o *workflow*. As condições de satisfação são encontradas quando o ator aceita a solicitação. Em uma oferta, o ator inicia a execução e as condições de satisfação são alcançadas quando o cliente aceita a oferta. Dessa forma, o modelo da Action *Workflow*, descreve interações entre participantes de um processo de negócios como *workflows*, ocorrendo entre clientes e executores (figura 8-1).

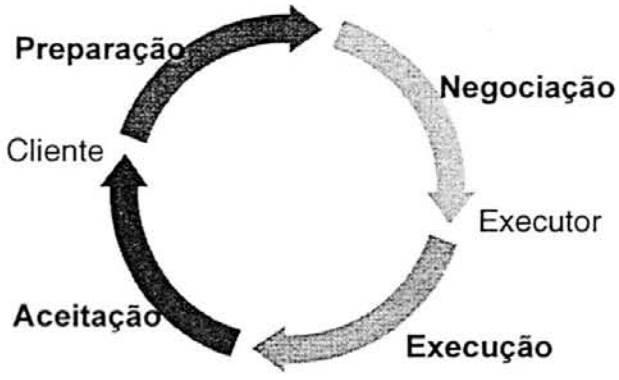


FIGURA 8-1: *Workflow* representado em 4 fases

O propósito destas fases é fazer com que duas ou mais pessoas concordem com relação a execução de uma ação. Esta metodologia define os elementos do *workflow* baseada na execução de cada fase. O significado de cada fase é descrito a seguir:

- Preparação - um cliente requisita uma ação para ser executada, ou um executor se oferece para executar alguma ação.
- Negociação - ambos, cliente e executor, concordam na execução da ação e definem os termos da negociação.
- Execução - a ação é executada de acordo com os termos estabelecidos.
- Aceitação - o cliente relata a satisfação (ou não satisfação) com a ação.

Cada laço do *workflow* pode ser agrupado com outros laços de *workflow* para completar um processo de negócio. O cliente em um laço do *workflow* pode ser executor em outro.

Embora exista um número infinito de interações humanas possíveis, a quantidade de tipos diferentes de interações que ocorrem é relativamente pequena. Na metodologia da *ActionWorkflow*, estes tipos distintos de interações são chamadas ações. O tipo de ação possível em determinado momento depende da fase do *workflow* e se o cliente ou ator está agindo. Nenhuma ação direciona o *workflow* à conclusão, nem tampouco o

finaliza sem este estar concluído.

8.4.3 Definições utilizadas pela Action *Workflow*

Esta seção apresentará conceitos utilizados pela Action *Workflow* na modelagem de um *workflow*, por serem específicos de sua metodologia.

- **Participantes do *Workflow***

São participantes do *workflow* clientes, atores e observadores. Clientes são responsáveis por avaliar o trabalho e determinar se as condições de satisfação foram alcançadas. Pode ser alguém externo à organização (solicitando mercadorias ou serviços), ou um cliente interno. Ao ator cabe realizar a tarefa e notificar o cliente quando o trabalho estiver completo. Também é possível especificar observadores do *workflow*.

- **Condições de Satisfação**

As condições de satisfação representam o objetivo do *workflow* em questão. Estas condições são utilizadas como critério para determinar se o cliente se encontra satisfeito, ou seja, se a meta do *workflow* foi alcançada. O período de tempo no qual um *workflow* é completado chama-se ciclo. A duração do ciclo faz parte das condições de satisfação, sendo explicitamente fixada para cada *workflow*.

- **Mapas**

O mapa é a representação gráfica do processo de negócios. Descreve o fluxo de trabalho e define uma série de eventos que têm um objetivo comum a ser alcançado. Mostra a rede de interconexões de compromissos pessoais e o fluxo de trabalho necessário para alcançar as condições de satisfação do cliente. Um mapa é formado por *workflows*, relacionados através de gatilhos (triggers). No mapa são descritas as diferentes interações que ocorrem entre pessoas num processo de negócios. A partir da análise do mapa, é ocorrem várias oportunidades de melhorar o processo de negócios, antes que ocorra qualquer codificação.

Analisando as conversações que ocorrem entre as pessoas no processo de negócios, o mapa auxilia a eliminar esforços duplicados ou atividades que não suportam nenhum objetivo claramente definido, evidenciar os papéis de cliente e ator, além das condições de satisfação e, ainda, identificar a causa de interrupções no processo.

8.5 Comparativo TF-ORM, JetForm Workflow Builder e Process Builder Analyst Edition

A tabela a seguir representa um comparativo entre o TF-ORM e as ferramentas utiizadas na modelagem. O comparativo é realizado segundo os critérios fundamentais estabelecidos para a modelagem do *workflow*.

TABELA 8-2: Comparativo entre o TF-ORM e JetForm *Workflow* Builder e Process Builder Analyst Edition

	TF-ORM	***Jetform Workflow Builder***	***Process Builder Analyst Edition***
workflow* do tipo *ad hoc	Representa	representa	representa
***workflow* administrativo**	representa	representa	representa
***workflow* de produção**	representa	representa parcialmente	representa parcialmente
formalismo apresentado pelos modelos	formal	formal	formal
formalismo na definição de papéis	formal	informal (representação através de um rótulo)	informal (representação através de um rótulo)
representação formal de tarefas automáticas	representa formalmente (através da classe de processos)	representa semi-formal (através de atributos pré-definidos)	representa semi-formal (através de atributos pré-definidos)
formalismo na representação das atividades	representa formalmente (através da classe de processos)	semi-formal (através de atributos pré-definidos)	semi-formal (através de atributos pré-definidos)
representação da estrutura do objeto	representa formalmente (através das classes)	representa semi-formal	representa semi-formal
representação da estrutura do fluxo	representa formalmente	representa semi-formal	representa semi-formal
representação de aspectos relacionados a tempo	permite definição explicita	permite definição explicita (pré - definidos)	permite definição explicita (pré - definidos)
representação formal do tratamento de exceções	representa formalmente (através de regras de transição de estados) - podem ser representadas algumas exceções previsíveis [EDE 98]	não pode ser avaliado (utiliza BasicScript, uma linguagem para definir regras para o fluxo das atividades e avaliação dos papéis, não disponível na versão de análise (modelagem))	não pode ser avaliado (utiliza scripts, através da Action*Workflow* Basic Language Reference, não disponível na ferramenta de modelagem)

Uma observação precisa ser feita com relação ao critério tratamento de exceções. Nos dois casos as ferramentas de modelagem apresentam uma linguagem (proprietária) que possivelmente permitiria a representação de regras necessárias ao tratamento de exceções, porém apenas o pacote que incluí a ferramenta de gerenciamento permite a utilização desta linguagem, bem como sua validação. A princípio, como trabalho futuro, seria interessante mapear o TF-ORM para as ferramentas através da utilização das suas linguagens.

9. Conclusão

Um dos grandes problemas relacionados a modelagem de *workflow* consiste na utilização de técnicas de modelagem conceitual específicas a cada sistema de *workflow*, não havendo, dessa forma, um modelo aceito consensualmente. Esta situação determina que muitas das características relacionadas a modelagem sejam negligenciadas. Um importante aspecto a ser tratado nos modelos é a questão da modelagem formal do *workflow* e que constitui o tema central deste trabalho. O formalismo garante uma interação correta dos participantes do processo, a consistência dos dados e a representação segura do processo. Assim, neste contexto, esta dissertação apresenta uma técnica de modelagem de *workflow* utilizando o modelo de dados TF-ORM como referência. Esta escolha deveu-se ao seu formalismo, as suas características de orientação a objetos, a capacidade de representação de lógica temporal e ao domínio da modelagem estabelecida no CPGCC.

9.1 Objetivo do trabalho

Este trabalho teve como objetivos:

1. revisar os artigos existentes e outra bibliografia sobre o tema *workflow*, procurando determinar o que é um *workflow*, suas características e qual é a definição mais apropriada para este trabalho. Esta atividade é necessária pois este conceito, útil para a modelagem de sistemas de informação, é abordado por diferentes autores de formas complementares;
2. como *workflow* trata de um domínio aplicado, foi decidido analisar algumas ferramentas comerciais para validar a praticidade dos modelos conceituais desenvolvidos;
3. descrever e classificar os principais conceitos envolvidos em *workflow* de forma a possibilitar, posteriormente, a validação tanto dos conceitos quanto da modelagem através de um estudo de caso e a eventual utilização de um sistema comercial;
4. melhorar e tornar mais rigorosa a especificação do *workflow* em um nível conceitual, formalizando com um modelo único tanto seu comportamento interno (cooperação e interação entre tarefas) quanto o seu relacionamento com o ambiente (designação de tarefas de trabalho para executores);
5. apresentar uma técnica de modelagem conceitual que é a base para alcançar a especificação e implementação de *workflow*.

Neste modelo, construções são apresentadas para representar, eficientemente, a modularização e o paralelismo. Uma linguagem de definição textual de *workflow* é apresentada para permitir a especificação do *workflow*. Ela mostra como descrições formais do *workflow* podem ser usadas como entradas para gerar o esquema de dados do *workflow* e o código de regras para seu gerenciamento. Em adição, o paradigma de regras providencia um formalismo conveniente para expressar computações reativas que normalmente são influenciadas por eventos externos, gerados fora do WFMS (*Workflow Manager System*), tal como exceções ou pré-condições de tarefas.

9.2 Contribuições do Trabalho

Neste trabalho, as seguintes contribuições foram alcançadas:

- Sincronismo - foram identificadas as formas de sincronismo existente entre atividades, mostrando que a representação formal destas conexões é um fator fundamental que deve estar presente em um modelo de *workflow*. O fluxo de controle permite a sincronização do processo de execução de atividades. A interação dessas atividades através do fluxo de controle é representada por conexões. Neste trabalho, mostrou-se como, utilizando-se as regras de transição de estados, é possível representar as interações entre atividades de um *workflow* através do TF-ORM.
- Metanível de Modelagem de *Workflow* - a tecnologia de *workflow* é definida como um conjunto funcional de agentes implementado em vários níveis de detalhamento. Em [MCK 94] são definidos 5 níveis de *workflow*. A partir da análise destes 5 níveis verificou-se a necessidade de complementar algumas características necessárias à tecnologia de *workflow* propostas nestes níveis, o que nos levou a definir um sexto nível. Neste nível a tecnologia de *workflow* oferece uma ferramenta de modelagem. Esta ferramenta deve ser baseada em uma metodologia que permita definir procedimentos e atividades relacionadas aos processos de negócios, permitindo que estes processos sejam analisados e simulados antes de serem completamente automatizados;
- Análise de alguns modelos de *workflow* - este estudo permitiu que definíssemos um conjunto de critérios básicos necessários à análise e modelagem do *workflow*.
- Técnica de modelagem de *workflow* utilizando como modelo de referência o TF-ORM - este modelo foi estendido afim de representar formalmente e de modo mais completo as características de um *workflow*. Esta contribuição constitui o objetivo principal deste trabalho.
- Validação da técnica de modelagem de *workflow* através da modelagem de um estudo de caso - foi feita a modelagem de um estudo de caso (que representa dois tipos de *workflow* - *ad hoc* e administrativo) em duas ferramentas comerciais que representam as duas metodologias de modelagem de *workflow*.
- Análise das modelagens (ferramentas e TF-ORM) através dos critérios básicos de modelagem de *workflow* estabelecidos neste trabalho.

9.3 Trabalhos Futuros

A modelagem de *workflow* consiste em representar, através de algum formalismo, as atividades que compõem os processos, sua seqüência de execução e seus inter-relacionamentos, assim como os agentes responsáveis por sua execução e os recursos envolvidos. O sistema de gerência do *workflow* se baseia neste modelo para coordenar a execução das atividades. Falhas e exceções podem ocorrer durante a execução, o que pode acarretar graves problemas quando as aplicações são críticas. Nos próximos trabalhos será mostrado como o formalismo TF-ORM, quando utilizado para modelagem de *workflow*, permite a representação de algumas exceções e a definição de informações que possibilitem a recuperação do sistema em caso de falha [EDE 98,98a].

Outro trabalho em andamento consiste no desenvolvimento de uma metodologia

de modelagem de ensino a distância na Internet [NIM 98a, OLI 98]. Esta metodologia utiliza como modelo o TF-ORM e técnicas de sincronismo entre atividades baseadas em *workflow*.

Nos trabalhos mencionados acima, um estudo mais detalhado das características temporais do *workflow* permitirá que as mesmas sejam analisadas e representadas através do TF-ORM utilizando de forma mais completa a representação de aspectos temporais permitidos por este modelo.

Bibliografia

[ACT 96] ACTION TECHNOLOGIES CORPORATION. The Action *Workflow* Method. Disponível em Brian_Hawkins@actiontech.com, December 1996.

[ACT 96a] ACTION TECHNOLOGIES Home Page, Jun. 1996. Disponível na Intenet em: http://www.actiontech.com

[ACT 96b] ACTION TECHNOLOGIES CORPORATION. Process Builder User's Guide - MB200-0396, December 1993-1996.

[ALO 95] ALONSO, G. et al. Advanced Transactional Models in Workflow Context. In Proceedings of 12th International Conference on Data Engineering, New Orleans, Lousiana. Institute for Information Systems. ETH – Zentrum CH-8092, Zürich, Switzerland.

[AMA 93] AMAVARDI, C. S. et al. The adequacy of office models. Working Paper submited to ACM Computing Surveys, May 1993.

[ANT 91] ANTONELLIS, V. et al. F-ORM - METHOD: a F-ORM methodology for reusing specifications, IFIP WG. 8.4 Working Conf. on The Object Oriented Approach in Information Systens, Quebec City, Oct. 1991.

[BAI 93] BAIR, James A. Constrating Workflow Models: Getting to the Roots of Three Vendors. New Science Associates - 2755 Campus Drive, Suite 145. San Carlos, CA- 1993.

[BLU 96] BLUMENTHAL, Richard. Unstructured Activities in Office Information and Workflow Systems. US WEST Advanced Technologies - Boulder, CO.

[BLU 96a] BLUMENTHAL, Richard and Nutt, Gary J. Supporting Unstructured Workflow Activities in the Bramble ICN System. Departament of Computer Science. The University of Colorado - Boulder, CO.

[BUR 93] BURNS, Nina. Ebb and Flow. Lan Magazine , 118-123, may 1993.

[CAE 88] CASAIS, E. An objetct Oriented System Implementating KNOs. In: Conference of Information System, Palo Alto California, 23-25 Mar 1988. Proceedings. Ed. Robert B. Allen, 1988. p. 284-90.

[CAS 95] CASATI, F.; Ceri S.; Pernici, B.; Pozzi, G. Conceptual Modeling of Workflows. Proceedings of OO-ER Conference. Gold Coast, Autrália, 1995, p. 341-354.

[CAS 95a] CASATI, F.; Ceri S.; Pernici, B.; Pozzi, G. Deriving Active Rules for Workflow Enactement. . Dipartamento di Elettronica e Informazione. Politecnico di Milano. In: Proc. of 7th International Conference on Database and Expert Systems Applications. Zurique, Suiça. Set. 1996.

[CAS 95b] CASATI, F.; Ceri S.; Pernici, B.; Pozzi, G. Workflow Evolution. Dipartamento di Elettronica e Informazione. Politecnico di Milano. In: Proc. of the 15th ER'96 International Conference. Cottbus, Alemanha, Out. 1996.

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

[EDE 93] EDELWEISS, N; Oliveira, J. Palazzo M. de; Pernici B. An Object-Oriented Temporal Model. In: International Conference on Advanced Information System Engineering (CAISE'93), 5., 1993, Paris. Proceedings... Heildelberg: Springer-Verlag, 1993. p. 397-415. (Lecture Notes in Computer Science, 685).

[EDE 94] EDELWEISS, N. Sistemas de Informação de Escritórios: Um modelo para especificações temporais. Porto Alegre: CPGCC da UFRGS, 1994. Tese de Doutorado.

[EDE 94a] EDELWEISS, N. Modelagem de Aspectos Temporais de Sistemas de Informação. In: IX Escola de Computação. Recife, 1994. 142p.

[EDE 98] EDELWEISS, N.; NICOLAO, M. Representando Exceções e Prevendo Tratamento de Falhas no Nível de Modelagem de Workflow. XIII Simpósio Brasileiro de Banco de Dados - 12-14 de outubro de 1998- Maringá, Paraná - Brasil

[EDE 98a] EDELWEISS, N.; NICOLAO, M. Workflow Modeling: Exception and Failure Handling, N. Edelweiss and M. Nicolao, Representation International Conference of the Chilean Computer Science Society (SCCC'98) - Antofagasta, Chile on November 12-14, 1998 - will be included in the Conference Proceedings.

[EDJ 95] EDER, J.; LIBHART, W. The Workflow Activity Model WMO. Proceedings 3rd International Conference on Cooperative Information Systems, Wien, 1995, p. 87-98.

[EDJ 98] EDER, J.; LIBHART, W. Contributions to Exception Handling in Workflow Management. Departament of Informatics-Systems. University of Klagenfurt, Autria. The EDBT Workshop on Workflow Management Systems, March 27-28, 1998, p. 1-10.

[EDL 93] EDELCO, Oliveira; CASTILHO, JMV. A Temporal Logic for Temporal Conditions Definition. Proc. 13th Int. Conf. of the Chilean, Chile, pg. 163-178, Oct 1993.

[ELL 91] ELLIS, C. A. et. Al. Groupware: some issues and experiences. Communications of the ACM, 34(1). Jan, 1991, p. 38-58.

[ELL 96] ELLIS, C. A.; NUTT, G. J. Workflow: The Process Spectrum. Departament of Computer Science. University of Colorado. Boulder, CO 80309-0430.

[GEO 95] GEORGAKOPOULOS, D. An Overview of Wokflow Management: From Process Modeling to Workflow Automation Infrastructure. Distribuid and Parallel Database, 3, 119-153,1995.

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

[HEI 98] HEINL, P. Exceptions During Workflow Execution. Departament of Computer Science VI. University of Earlangen – Nurembeg. The EDBT Workshop on Workflow Management Systems, March 27-28, 1998, p. 11-20

[HIR 86] HIRSCHHEIM, R. A. The effect of a priori views on the social implications of computing: The case of office automation. ACM Computing Surveys, 18(2): 165 – 196, June1986.

[JEN 94] JENSEN, C. S. (ed.). A Consensus glossary of temporal database concepts. Sigmod Record, v.23, n.1, p.52-63, Mar. 1994.

[JOO 94] JOOSTEN, M. M. S. Trigger Modelling for Workflow Analysis. Design Methodology Group. Center for Telematics and Information Technology, University or Twente, P.O. Box 217, 7500 AE Enschede, the Netherlands.

[JOO 94a] JOOSTEN, M. M. S. et al. Workflow Literature Guide, version 1.1. University of Twenty, Set. 1994. Disponível na Internet em: http://www_is.cs.utwente.nl:8080/~joosten/documents/guide.mccrory.ps

[JOO 95] JOOSTEN, M. M. S. Conceptual Theory for Workflow Management Support Systems. Center for Telematics and Information Technology, University or Twente, P.O. Box 217, 7500 AE Enschede, the Netherlands.

[JOO 95a] JOOSTEN, M. M. S. A Method for Analysing Workflows , Computer Information System Departament. Georgia State University - Atlanta, GA - USA.

[JOO 95b] JOOSTEN, M. M. S. Fundamental Concepts for Workflow Automation in Pratice. University of Twenty, Set. 1994. Disponível na Internet em:
: http://www_is.cs.utwente.nl:8080/~joosten/documents/artikel.ICIS.ps

[KOB 95] KOBELIUS, J. THE RHYTM OF WORK : A Buyer's Guide to Workflow Tools. Network World /Collaboration, November/December 1995.

[KHO 95] KHOSHAFIAN, Setrag et, al. Introduction to Groupware, Workflow, and Workgroup Computing. John Wiley & Sons, Inc - 1995.

[KRE 91] KREIFELTS, T. et al. Experiences with domino office procedure system. Proceedings of the second European in the computer Supported Cooperative Work, Milano, Italy, September 1993. Kluer Academic Plubishers

[MAN 86] MANOLA, F.; Dayal, U. PDM: An Object-Oriented Data Model. Proceedings of the International Workshop on Object Oriented Database Systens, Pacific Grove, California, 23-6 Sept. 1986. p. 18-25.

[MAR 95] MARSHAK, R.; Rethinking T. Workflow: Incremental Steps to BPR, Workgroup Computing Report, 1995 - http://www.psgroup.com/snapshot/1995/ed395w.htm

[MCC 92] MCCREADY, S. "There is more than one Kind of Workflow Software", Computerworld, November 2, 1992.

[MCK 94] MCKIE, S. The Five Levels of Workflow. DBMS, 74-76, april 1994.

[MED 92] MEDINA-MORA, R. et al. The Action Workflow approach to Workflow Management Technology. In: CSCW, 1992 [S.1.]. Proceedings...[S.1.:s.n.],1992.

[MIL 95] MILLER, J. A. et al. Simulation Modeling Within Workflow Technology - Departament of Computer Science/LSDIS Lab - The University of Georgia - Athens, Georgia 30603 - 7404, USA.

[MOH 94] MOHAN, C. Tutorial: Advanced Transaction Models – Survey and Critiquee. In: ACM SIGMOD International Conference on Management of Data, Minneapolis, Mai. 1994.

[MOH 95] MOHAN, C. et al. Advanced Transaction Model in Workflow Contexts. Relatório de Pesquisa. IBM Almaden Research Center, Dez. 1995. Disponível na Intenet em: http://www.almaden.ibm.com/cs

[NIC 96] NICOLAU, C. Organize seus Papéis como Documentos Digitais. PC Magazine Brasil, Junho de 1996, p. 30.

[NIM 96] NICOLAO, M. Um Estudo sobre Conceituação de Workflow, CPGCC/UFRGS, Porto Alegre, Dezembro 1996. (Trabalho Individual I, T.I n. 571)

[NIM 96a] NICOLAO, M.; Oliveira, J. P. Caracterizando Sistemas de Workflow. Revista REAd, ISSN 1413-2311, Programa de Pós-graduação em Administração - UFRGS, n. 3, out. '96, {publicação eletrônica}.

[NIM 98] NICOLAO, M.; EDELWEISS, N. Workflow Modeling using a Temporal Object-Oriented Model. The EDBT Workshop on Workflow Management Systems, March 27-28, 1998, p. 71-79.

[NIM 98a] NICOLAO, M. et al. Modelagem Conceitual de Workflow para Cursos Remotos- Mariano Nicolao et. al. Simpósio Brasileiro de Informática na Educação – Fortaleza-Ceará, 17-19 de novembro de 1998.

[OLI 95] OLIVEIRA J. P. M. et al. Implementation of an Object-Oriented Temporal Model. In: Database And Expert Systems Applications Conference – DEXA 6., 1995, London, U. K. Proceedings..., 1995.

[OLI 98] OLIVEIRA, J. P. M.; Nicolao M. and Edelweiss N. Conceptual Workflow Modelling for Remote Courses, , 15th IFIP World Computer Congress, 31 August - 4 September 1998, Vienna and Budapest.

[PER 90] PERNICI, B. Objects with Roles. SIGOIS Bulletin, v. 11, n. 2-3, p. 205-15, 1990.

[ROB 96] ROBERTS, B. Notes X Web - BYTE - volume 5, n° 7, p. 40 - 53, Julho 96.

[RUM 94] RUMBAUGH, James et al. – Sistema de Banco de Dados – MAKRON Books do Brasil Editora Ltda, 1994.

[RUM 91] RUMBAUGH, James et al. Object Oriented Modeling and Design. Prentice Hall, Englewood Cliffs, New Jersey 07632.

[SAI 94] PIERRE , Saint. Universidad de Chile, Facultad de Ciencias Fisicas y Matematicas. Santiago-Chile.

[SCH 91] SCHAEL, T. et al. "Design Principles for Cooperative Office Support Systems in Distributed Process Management", in Support Functionality in the Office Environment, A. Verrijn-Stuart (ed), North Holland, 1991.

[SHE 95] SHETH, A. P. Workflow Automation: Applications, Technology and Research. Tutorial Notes, SIGMOD Conference, Maio de 1995, California. University of Georgia, Dept. of Computer Science, Graduate Studies Research Center - Athens GA, USA.

[SHE 97] SHETH, A. P. Modeling The Bussiness, a Tutorial . Disponível na Intenet em: http://wwwis.cs.utwente.nl:8080/~joosten/tutorial.html

[SCM 96] SCHIPPER, M.; JOOSTEN, S. A Validation Procedure for Information Modeling Techinques in: Wand, Y., Siau, K. (ed): Proceedings of the CAiSE*96 Workshop on Evaluation of Modeling Methods, 1996.

[SNO 85] SNODGRASS, R. et al. A Taxonomy of Time in Databases. Proceedings of the ACM SIGMOD International Conference on Management of Data. Texas, May 28-31, 1985. p. 236-46.

[STI 98] STIPHOUT, R. et al. TREX: Workflow Transactions by Means of Exceptions. Univesity of Klagenfurt, Austria. The EDBT Workshop on Workflow Management Systems, March 27-28, 1998, p. 21-26.

[TSI 87] TSICHRITZIS, D. et al. KNOS: Knowledge Acsition, Dissmination and Manipulation Objects. ACM Transaction on Office Information Systens, New York, v. 5, n. 1, p.96-112, Jan. 1987.

[WEI 95] WEIKUM, G. Workflow Monitoring: Queries On Logs or Temporal Databases? Paper Submitted to HPTS'95. Departament of Computer Science. University of Saarland. P.O.Box 151150, D-66041 Saarbruecken, Germany. E-mail:weikum@cs.uni-sb.de

[WFM 96] WORKFLOW MANAGEMENT COALITION . Terminology & Glossary, Document Number WFMC-TC-1011, Document Status - Issue 2.0, June 1996

[WFM 96a] WORKFLOW MANAGEMENT COALITION . Interface 1: Process Definition Interchange, Document Number WFMC-TC-1016, Document Status - Version 1.0 Beta, June 1996

[WFM 96b] WORKFLOW MANAGEMENT COALITION. The Workflow Reference Model, Document Number WFMC-TC00-1003, Document Status - Issue 1.1, November 1994.

[WIN 87] WINOGRAD, T. et al. A language/action perspective on the design of coopetative work. Human-Computer Interaction, v.3, n. 1, 1987, p. 3-30.

[WIN 87a] WINOGRAD, T. et al. Understanding Computers and Cognition: A New Fundation for Design. Reading: Addison-Wesley, 324 p., 1987.

[WIN 88] WINOGRAD, T. et al. Computer Systems and the Design of Organizational Interaction. ACM Transactions on Office Information Systems, New York, v.6, n.2, p.126-152. Apr. 1988.

[WIN 94] WINOGRAD, T.; FLORES, F.; M. HARTFIELD, B.; Computer Systens and the Design of Organizational Interaction, ACM Transactions on Office Informations Systens, 6. Abr. 94, 153-172.

ANEXO 1 - BNF da Linguagem de definição do modelo TF-ORM

A notação utilizada é uma BNF ("Backus Naur Form" ou "Backus Normal Form") simplificada, utilizada em gramáticas livres do contexto. Cada unidade sintática da linguagem é definida através de uma regra de produção. O lado esquerdo de cada regra apresenta somente uma metavariável, separada do lado direito por "::=". Do lado direito das regras aparecem metavariáveis e terminais, além de símbolos especiais que denotam regras alternativas, partes opcionais e partes repetidas. As metavariáveis são delimitadas pelos símbolos "<" e ">". Os terminais podem ser seqüências de caracteres ou símbolos. As seqüências de caracteres são representadas por elas mesmas, em negrito; os símbolos são delimitados por um par de duplas aspas. O símbolo especial "|" separa duas alternativas para a definição da mesma metavariável. Conjuntos de elementos que são opcionais são delimitados pelos símbolos "[" e "]". Conjuntos de elementos que podem ser repetidos zero ou mais vezes são delimitados pelos símbolos "{" e "}*". E conjuntos de símbolos que podem ser repretidos uma ou mais vezes são delimitados pelos símbolos "{" e "}$^{+}$". A precedência dos operadores é a seguinte: opcional, repetição, seqüência e alternativas.

As regras de produção correspondentes à linguagem de definição de dados do modelo TF-ORM, estendidas com as do capítulo 7, são as seguinte:

<class declaration> ::=

<process class declaration> | <resouce class declaration>| <agent class declaration>

<process class declaration> ::= process class <process class definition>

<resource class declaration> ::= resource class <resource class definition>

<agent class declaration> ::= agent class <agent class definition>

<process class definition> ::=

"(" <class name> "," <base-role declaration> {"," <process role declaration> }* ")"

| "(" <class name> <specialization declaration> ","

[<disabled roles declaration> ","]

[<extended role declaration> ","]

<base-role declaration> { "," <process role declaration>}* ")"

<resource class definition> ::=

"(" <class name> "," <base-role declaration>

{ "," <resource role declaration> }* ")"

| "(" <class name> <specialization declaration> ","

[<inherited roles declarations> ","]

[<disabled roles declaration> ","]

[<extended role declaration> ","]

<base-role declaration> { "," <resource role declaration> }* ")"

| "(" <class name> <component declaration> ","
[<disabled roles declaration> ","]
< base-role declaration> { "," <resource role declaration> }* ")"

<agent class definition> ::=
"(" <class name> "," <base-role declaration>
{ "," <agent role declaration> }* ")"
| "(" <class name> <specialization declaration> ","
[<inherited roles declaration> ","]
[<disabled roles declaration> ","]
[<extended role declaration> ","]
<base-role declaration> { "," <agent role declaration> }* ")"

<class name> ::= <identifier>

<identifier> ::= <letter> { <identifier character> }*

<identifier character> ::= <letter> | <digit> | "_"

<specialization declaration> ::= is_a <class name>
| is_a "(" <class name> { "," <class name> }+ ")"

<inherited roles declaration> ::=
inherits [<class name> "."] <role name>
| inherits "(" [<class name> "."] <role name>
{ "," [<class name> "."] <role name> }* ")"

<disabled roles declaration> ::=
not_inherits [<class name> "."] <role name>
| not_inherits "(" [<class name> "."] <role name>
{ "," [<class name> "."] <role name> }* ")"

<extended role declaration> ::=
extends [<class name> "."] <role name>
| extends "(" [<class name> "."] <role name>
{ "," [<class name> "."] <role name> }* ")"

<component declaration> ::=
composed_of "{" <class name> { "," <class name> }* "}"

<role name> ::= <identifier>

<base-role declaration> ::= "<" base_role ["," <static properties declaration>]
["," <dynamic properties declaration>] "," <rule declaration> ">"

<process role declaration> ::= "<" <process role name>

["," <static properties declaration>]

["," <dynamic process properties declaration>]

< message declaration> "," <state declaration> "," <rule declaration> ">"

<resource role declaration> ::= "<" <resource role name>

["," <static properties declaration>]

["," <dynamic properties declaration>]

< message declaration> "," <state declaration> "," <rule declaration> ">"

<agent role declaration> ::= "<" <agent role name>

["," <static properties declaration>]

["," <dynamic properties declaration>]

["," <agent_decision declaration>]

["," <exec declaration>] ["," <resp declaration>] ["," <auto declaration>]

<message declaration> "," <state declaration> "," <agent rule declaration> ">"

< agent_decision declaration> ::= "{"{ decisions_task> "(" [<class nome> "."] <role name> ")" ":" "{" < decision definition> ["," <decision definition] }* "}" "}"

<exec declaration> ::= executor_tasks "=" "{" <class role list> "}"

<resp declaration> ::= responsible_tasks "=" "{" <class role list> "}"

<auto declaration> ::= Auto

<class role list> ::= "(" [<class nome> "."] <role name> ")"

{"," "(" <class nome> "."] <role name> ")"}*

<process role name> ::= <agent name> "." <activity name> | <activity name>

<agent name> ::= [<class name> "."] <agent role name>

<activity name> ::= <identifier>

<resource role name> ::= <identifier>

<agent role name> ::= <identifier>

<static properties declaration> ::= static properties "=" "{" [<property list>] "}"

<dynamic process properties declaration> ::= dynamic properties "=" "{"

[executing agent "," <class name> ","]

[message senders "," "{" <sender list> "}" ","]

[message receivers "," "{" <sender list> "}" ","]

[<property list>] "}"

<sender list> ::= <sender> | <sender> { "," <sender> }

<sender> ::= <class name> | <class name> "." <role name> | <role name>

<dynamic properties declaration> ::= dynamic properties "=" "{"

<property list> "}"

<property list> ::= "(" <property name> "," <property domain> ")" ["," <property list>]

<property name> ::= <identifier>

<property domain> ::= <simple domain> | <complex domain>

<simple domain> ::= <class name> ["." <role name>]

| <role name>

| <predefined domain>

| "{" <string list> "}"

<predefined domain> ::= integer | real | boolean | string | text | place | title | image

| <temporal point type> | <interval> | | <limit>

<temporal point type> ::= instant | date | time | year | month | day | hour | minute

| week | semester | century | weekday

<interval> ::= interval "(" <interval type> "," <interval limits> ")"

<interval type> ::= closed | open | open_down | open_up | floating_down | floating_up

<interval limits> ::= instant | date | time | year | month | day | hour | minute

 ::= span "(" <span type> ")"

<span type> ::= year | month | day | hour | minutel week | semester | century

<limit> ::= after "(" <limit type> ")"

| before "(" <limit type> ")"

| within "(" <interval limits> ")"

<limit type> ::= instant | date | time | year | month | day | hour | minute

<string list> ::= <identifier> ["," <string list>]

<complex domain> ::= "{" <simple domain> "}" | "{" <property list> "}"

| "(" property list> ")" | set_of <simple domain> | list_of <simple domain>

<process_decision declaration> ::= decisions "=" "{"

<decision definition> { "," <decision definition> }* "}"

<decision definition> ::= <decision>

<decision> ::= <decision name> ["(" <decision parameters declaration> ")"]

<decision name> ::= <identifier>

<decision parameters declaration> ::=

<parameter declaration>{ "," <parameter declaration> }*

["," valid_time ":" <date value>]

<parameter declaration> ::= <parameter name> ":" <property domain>

| <property name>

<parameter name>::= <identifier>

<message declaration > ::= messages "=" "{"

<message definition> { "," <message definition> }* "}"

<message definition> ::= <message> to <roles>

| <message> to EXTERNAL_WORLD

| <message> to itself

| <message> from <roles>

| <message> from EXTERNAL_WORLD

<message> ::= <message name> ["(" <message parameters declaration> ")"]

<message name> ::= <identifier>

<message parameters declaration> ::=

<parameter declaration> { "," <parameter declaration> }*

["," valid_time ":" <date value>]

<roles> ::= <role> | "{" <role> { "," <role> }* "}"

<role> ::= [<class name> "."] <role name>

<state declaration> ::= states "=" "{" [<state> { "," <state> }*] "}"

<state> ::= <state name> | "(" <state name> "," <state name> { "," <state name> }* ")"

<state name> ::= <identifier>

<rule declaration> ::= rules "="

"{" [<rule name> ":" <rule> { "," <rule name> ":" <rule> }*] "}"

<rule name> ::= <identifier>

<rule> := <state transition rule> | <integrity rule>

<agent rule declaration> ::= rules "="

"{" [[<rule name>] ":" <agent rule> { "," [<rule name>] ":" <agent rule> }*] "}"

<agent rule> := <agent state transition rule> | <integrity rule>

<state transition rule> ::=

<left predicate> "⇒" <right predicate> [<state transition condition>]

<left predicate> ::= <state predicate> "," [<message predicate in>]

| <message predicate in>

<agent state transition rule> ::=

<agent left predicate> "⇒" <right predicate> [<state transition condition>]

<agent left predicate> ::= <state predicate> ["," <message predicate in>]

| <message predicate in>

| <state predicate> ["," <decision predicate>]

| <decision predicate>

<right predicate> ::= <message predicate out> [“,” <right predicate>]

| <state predicate>

<state predicate> ::= <defined state predicate> | <predefined state predicate>

<defined state predicate> ::= state “(” [<id variable> “,”] <state name> “)”

<predefined state predicate> ::= state “(” [<oid variable> “,”] <predefined state> “)”

<predefined state> ::= active | suspended

<message predicate in> ::= msg “(” <message in> “)”

| “{” <message in list> “}”|<integer number> “{” <message in list> “}”

<message in list> ::= <message in> { “,” <message in> }*

<message in> ::= <message name> [“(” <message parameters> “)”]

| <predefined message predicate>

<decision predicate> ::=

decision “(” <decision name> [“(” <message parameters> “)”] “)”

<message parameters> ::= <parameter> { “,” <parameter> }*

<parameter> ::= <parameter name>

<predefined message predicate> ::=

create_object [“(” <class name> “,” < oid variable> “)”]

| suspend_object [“(” <oid variable> “)”]

| resume_object [“(” <oid variable> “)”]

| kill [“(” <oid variable> “)”]

| kill “(” itself “)”

| add_role [“(” [<oid variable> “,”] <role> [“,”<role variable>] “)”] [“,”<exec definition >] [“,”<deleg definition >] [“,”<auto definition >] “)”]

| suspend_role [“(” <role variable> “)”]

| resume_role [“(” <role variable> “)”]

| terminate_role [“(” <role variable> “)”]

| forbid_role [“(” [<role variable> “,”] <role> “)”]

| allow_role [“(” [<oid variable> “,”] <role> “)”]

| forbid_op “(” [<role variable> “,”] <direction> <message name> “)”

| allow_op “(” <role variable> “,” <direction> <message name> “)”

| forget “(” [<oid variable> “,”] <rule name> “)”

| recall “(” [<oid variable> “,”] <rule name> “)”

| start [“(” [<oid variable> “,”] <role> [“,” <role variable>] “)”]

| stop [“(” [<oid variable> “,”] <role> [“,” <role variable>] “)”]

| in_class "(" <class name> ")"

| out_class "(" <class name> ")"

< exec definition > ::= <identifier>

< deleg definition > ::= <identifier>

< auto definition > ::= Auto

<message predicate out> ::= msg "(" <message out> ")"

<message out> ::= <message name> ["(" <message parameters> ")"]

| <predefined message predicate> [to <receiver>]

<receiver> ::= [<role variable> ":"] <role> | itself

<oid variable> ::= <variable>

<role variable> ::= <variable>

<id variable> ::= <oid variable> | <role variable>

<state transition condition> ::= ";" "(" <logical expression> ")"

<integrity rule> ::= constraint "(" <integrity condition declaration> ")"

<integrity condition declaration> ::=

<simple integrity condition> | <instanciated integrity condition>

<simple integrity condition> ::= <logical expression> "⇒" <logical expression>

<instanciated integrity condition> ::= [<temporal operator>] <quantifier> <variable>

"(" <integrity condition declaration> ")"

<logical expression> ::= <logical term>

| <logical expression> <or operator> <logical term>

<logical term> ::= <logical factor>

| <logical term> <and operator> <logical factor>

<logical factor> ::= <logical element> | not <logical element>

<logical element> ::= <predicate> | "(" <logical expression> ")"

| <logical element> <temporal logical operator> <predicate>

<or operator> ::= or | ";"

<and operator> ::= and | ","

<temporal logical operator> ::= since | until | before | after

<predicate> ::= <predefined predicate>

| <predefined temporal predicate>

| <state predicate>

| <predefined state predicate>

| [<temporal operator>] <quantifier> <variable> "(" <predicate> ")"

| <arit expression> <comp operator> <arit expression>

| <temporal operator> <logical expression>

| false

| true

<predefined predicate> ::=

has_class_instance "(" <class name> "," <oid variable> ")"

| has_role_instance "(" <oid variable> "," <role name> "," <role variable> ")"

| active_class "(" <oid variable> ")"

| active_role "(" <role variable> ")"

| active_class_at "(" <oid variable> ","<temporal instant> ")"

| active_role_at "(" <role variable> "," <temporal instant> ")"

| is_valid "(" <id variable> "," <property name> ")"

| is_valid_at "(" <id variable> "," <property name> "," <temporal instant> ")"

| out_role "(" <oid variable> "," role name> ")"

| role "(" <oid variable> "," <role variable> ")"

<predefined temporal predicate> ::=

belongs "(" <function argument> "," <function name argument> ")"

| contains "(" <function name argument> "," <function argument> ")"

| before "(" <function argument> "," <function argument> ")"

| equal "(" <function argument> "," <function argument> ")"

<function argument> ::= [<id variable> ","] <property name>

| <temporal instant> | <variable>

<function name argument> ::=

[<id variable> ","] <property name> | <variable>

<temporal instant> ::= <number> "/" <number> "/" <number> ","

<number> ":" <number>

<temporal operator> ::= sometime past | immediately past | always past

sometime future | immediately future | always future

<quantifier> ::= exists | forall

<comp operator> ::= "<" | ">" | "=" | "≤" | "≥" | "≠"

<arit expression> ::= <term> | "-" <arit expression>

| <arit expression> "+" <term> | <arit expression> "-" <term>

<term> ::= <factor> | <term> "*" <factor> | <term> "/" <factor>

<factor> ::= <element> | <factor> "**" <element>

| <element> union <element>

| <element> intersection <element>

<element> ::=

[<id variable> ","] <property name>

| [<id varible> ","] <predefined property name>

| <function> | <value> | <variable> | "(" <arit expression> ")" | now

<predefined property name> ::= oId | object_instance | end_object

| rId | role_instance | end_role

<function> ::=

year "(" <function argument> ")"

| month "(" <function argument> ")"

| day "(" <function argument> ")"

| hour "(" <function argument> ")"

| minute "(" <function argument> ")"

| weekday "(" <function argument> ")"

| lower_bound "(" <function name argument> ")"

| upper_bound "(" <function name argument> ")"

| duration "(" <function name argument> ")"

| interval "("<function name argument>","<function name argument> ")"

| to_minutes "(" <function argument> ")"

| to_months "(" <function argument> ")"

| to_days "(" <function argument> ")"

| value "(" [<id variable> ","] <property name> ")"

| past_value "("[<id variable>","]<property name><temporal instant> ")"

| valid_time "(" [<id variable> ","] <property name> ")"

| transaction_time "(" [<id variable> ","] <property name> ")"

| class_creation_time "(" <oid variable> ")"

| role_creation_time "(" <role variable> ")"

| class_end_time "(" <oid variable> ")"

| role_end_time "(" <role variable> ")"

| state "(" <id variable> ")"

| state_at "(" <id variable> <temporal instant> ")"

<value> ::= <integer number> | <string> | <temporal value> | null

<integer number> ::= <digit> { <digit> }*

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<string> ::= " ' " { <any character including blanck> }+ " ' "

<temporal value> ::= <temporal instant> | <date value> | <hour value>

<date value> ::= <number> "/" <number> "/" <number>

<hour value> ::= <number> ":" <number>

<number> ::= <digit> <digit>

<variable> ::= <identifier>

<identifier> ::= <letter> { <letter> | <digit> | "_" }*

<letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N

| O | P | Q | R | S | T | U | V | W | X | Y | Z

| a | b| c | d | e | f | g | h | i | j | k | l | m | n

| o | p | q | r | s | t | u | v | w | x | y | z

ANEXO 2 - Modelagem do Estudo de Caso no TF-ORM

Representação Ad Hoc

agent class (

PERSON,

<Base_role,

static propeties = {(name, string), (d_birth, date), (gender, {F,M}) },

dynamic properties = {
(adress_pers, string),
(e_mail, string),
(telephone, set of integer),
(specialisations, set of string),
(University, string) },

rules = {
r1: state(active), msg(create_object) ⇒ msg(allow_role(Presidente_do_Comitê)),
r2: state(active), msg(create_object) ⇒ msg(allow_role(Secretária)),
r3: state(active), msg(create_object) ⇒ msg(allow_role(Substituto)),
r4: state(active), msg(create_object) ⇒ msg(allow_role(Referee)),
r5: state(active), msg(create_object) ⇒ msg(allow_role(Comitê_de_revisão))}
>,

<Presidente_do_Comitê,

static propeties = {...},
dynamic properties = {...},
states = {active, not active, vacations, analysing, analysing_answer_referee, waiting
},
executor_tasks = { (PAPERS_REVISION.Invited_referee),
(PAPERS_REVISION.Receive_analyse_paper)},
responsible_tasks = {(PAPERS_REVISION.Paper_receive)},
messages = {...},
decisions = {

decisions_task(PAPERS_REVISION.Invited_referee): {
referee_paper_values(Referee_name: string, Cod_referee: string,
Cod_paper:string)},
decisions_task(PAPERS_REVISION.Receive_analyse_paper): {
determine_paper_values (Cod_paper: string, Author_name: string, Paper: text,
University: string)
send_paper_to_referee(Cod_referee:string, Cod_paper:string, Paper:paper)
accept(Author_name:string, result:string)
naccept(Author_name:string, result:string)}
}

rules = {

msg(add_role) ⇒ state(active);
not exist Rid(has_role_instance(oid, Presidente_do_Comitê,Rid),
}
>,

<Secretária,

static propeties = {...},
dynamic properties = {...},
states = {(active), (not active), (vacations), (analysing), (waiting) },
responsible_tasks = {...},
executor_tasks = {(PAPERS_REVISION.Paper_receive)},
messages = {...},
decisions = {...},
rules = {...}
>,

<Substituto,

static propeties = {...},
dynamic properties = {...},
states = {active, not active, vacations, analysing, waiting },
messages = {...},
decisions = {...},
rules = {...},
>,

<Referee,

static propeties = {
(Cod_referee,string),
}
dynamic properties = {...},
states = {(active),(analysing_invitation), (revising), (waiting_answer), (vacations) }
executor_tasks = {(PAPERS_REVISION.Paper_valuation)}
messages = {...},
decisions = {
decisions_task (PAPERS_REVISION.Paper_valuation): {
accepted(Cod_referee: string, Cod_paper:set of string)
naccepted(Cod_referee: string, Cod_paper:set of string)
send_valuation(Cod_paper:string, Cod_referee:string, valuation: set of string)}
},
rules = {
msg(add_role) ⇒ state(active),

```
        }
       >,
< Comitê_de_Revisão,
   static propeties = {...},
   dynamic properties = {...},
   states = {active, not active, vacations,  analyse_valuation_referee }
   executor_tasks = {(PAPERS_REVISION.Analyse_valuation)}
   messages = {...}

   decisions ={
        decisions_task (PAPERS_REVISION.Analyse_valuation): {
        valuation_end(Cod_paper:string, final_valuation: set of string)
        /*avaliação concluída*/
        revaluation (Cod_paper:string, Cod_referee:string, valuation: set of string,
           Observations: set of string)}
   },
   rules = {
        msg(add_role) ⇒ state(active),
        }
       >,
```

```
process class (
 PAPERS_REVISION,
   <Base_role,
   static propeties = {...},
   dynamic properties = {(deadline, date) },
   rules = {
        r1: msg(create_object) ⇒ state (active),
        r2: state(active) ⇒ msg(add_role(receive_analyse_paper, presidente_do_comitê,
           secretária)),
        r3: state(active) ⇒ msg(add_role(invited_referee, presidente_do_comitê,
           presidente_do_comitê),
        r4: state(active) ⇒ msg(add_role(analyse_valuation, comitê_de_avaliação,
           comitê_de_avaliação)),
        r5: state(active) ⇒ msg(add_role(paper_valuation, referee, referee))
           /* segundo extensão proposta  - add_role(Oid, papel, Rid, [Resp], [Exec],
           [Auto])  permite criar instâncias dos papéis dos processos, com agentes
           responsável e executor definidos */
   }
  >,

<Receive_analyse_paper,
```

```
    /*Agente responsável = Presidente do comitê */
    /* Agente executor = secretária*/
states = { active, not active, waiting},
messages = {
    paper_Author(Author_name:string, Paper:text, University:string) from
      EXTERNAL_WORLD,
    paper_receipt  to EXTERNAL_WORLD,
    def_paper_values(cod_paper:string, Author_name:string, Paper:text,
      University:string, final_valuation: set of string)
    to RECURSO.Paper,
    final_ valuation(Cod_paper:string, final_valuation:set of string)
    from  PAPER_REVISION.Analyse_valuation,
    report_paper_naccept to EXTERNAL_WORLD,
    report_paper_accept to EXTERNAL_WORLD,
    report_paper(cod_paper: string, final_valuation: set of string) to
      RECURSO.Paper,
    },
decisions = {
    determine_paper_values (Cod_paper: string, Author_name: string, Paper: text,
      University: string, final_valuation: set of string)
    accept(Author_name:string, result:string)
    naccept(Author_name:string, result:string)
     },

rules = {
    Begin:
    msg(paper_Author(Author_name:string, Paper:text, University:string)),
      decision(determine_paper_values (Cod_paper, Author_name, Paper,
      University, final_valuation)  ⇒ msg(add_role(Oid,paper,Rid)),
      msg(paper_receipt), msg(def_paper_values (Cod_paper, Author_name,
      Paper, University, final_valuation)), state(active),
    Report1:
    state(active), msg(final_ valuation(Cod_paper, final_valuation)) ⇒
      state(analysing),
    Report2:
    state(analysing),  decison(accept(Author_name, result)) ⇒
      msg(report_paper_accept), msg(report_paper(Cod_paper, final_valuation,
      state(active))),
    Report3:
    state(analysing),  decison(naccept(Author_name, result)) ⇒
      msg(report_paper_naccept), msg(report_paper(Cod_paper, final_valuation),
      state(active)
    }
```

>,

<Invited_referee,

/* Agente executor = Presidente do comitê */

states = { active, not active, waiting},

messages = {

invitation(Cod_paper: string, Cod_referee: set of string, Abstract: string, Referee_name:string)

to PAPER_REVISION.Paper_valuation,

accepted (Cod_referee: set of string, Cod_paper: string)

from PAPER_REVISION.Paper_valuation,

naccepted (Cod_referee: set of string, Cod_paper: string)

from PAPER_REVISION.Paper_valuation,

paper_values(Cod_paper: string, Author_name: string, Paper: text, University: string)

from RECURSO.Paper,

referee_paper_values(Cod_referee: set of string, Cod_paper:string)

to RECURSO.Paper,

paper(Cod_paper:string, Cod_referee:set of string, paper:paper)

to PAPER_REVISION.Paper_valuation,

}

decisions = {

referee_paper_values(Referee_name:string, Cod_referee:string, Cod_paper:string)

},

rules = {

begin:

msg(add_role) ⇒ state(active)

Invitation_to_referee:

state(active), msg(paper_values(Cod_paper, Author_name, Paper, University)) ⇒ msg(invitation(Cod_paper, Cod_referee, Abstract, Referee_name)), state(active); value(Recurso.paper, university) ≠ value(Person.referee, university) and (not exists Oid(has_class_instance(resource, Oid) and exist Rid(has_role_instance(Oid, paper, Rid), value(Rid, cod_referee) = Cod_referee),

/*não permite que um revisor analise um trabalho de sua universidade e seja convidado duas vezes para o mesmo artigo*/

Answer_invitation_ok:

state(active), accepted (Cod_referee, Cod_paper),

decision(referee_paper_values(Referee_name, Cod_referee, Cod_paper)) ⇒
msg(referee_paper_values(Cod_referee, Cod_paper)), msg(paper(Cod_paper, Cod_referee, paper), state(active),

Answer_invitation_nok:
state(active), NAccepted (Cod_referee, Cod_paper)⇒
msg(invitation(Cod_paper, Cod_referee, Abstract, Referee_name)),
state(active);
value(Recurso.paper, university) ¹ value(Person.referee, university) and
(not exists Oid(has_class_instance(resource, Oid) and
exist Rid(has_role_instance(Oid, paper, Rid) and value(Rid, cod_referee) = Cod_referee)
/*não permite que um revisor seja convidado duas vezes para o mesmo artigo*/
}

<Analyse_valuation,

/* Agente Executor = {(comitê_de_avaliação) } */

states = { active, not active, waiting},
messages = {
valuation(Cod_paper:string, Cod_referee:set of string, valuation: set of string)
from PAPER_REVISION.Paper_valuation,
/*avaliação é inicialmente enviado ao presidente do comitê
review_criterions(Cod_paper:string, Cod_referee:set of string, valuation: set of string, Observations: set of string) ,
to PAPER_REVISION.Paper_valuation
final_ valuation(Cod_paper:string, final_valuation:set of string)
to PAPER_REVISION.Receive_analyse_paper
},
decisions ={
valuation_end(Cod_paper:string, final_valuation: set of string),
/*avaliação concluída*/
revaluation (Cod_paper:string, Cod_referee:set of string, valuation: set of string, Observations: set of string)
},

rules = {
receive_valuation:
state(active),{msg(valuation(Cod_paper, Cod_referee1, Valuation1),
msg(valuation(Cod_paper, Cod_referee2, Valuation2)),
msg(valuation(Cod_paper, Cod_referee3, Valuation3))} ⇒
state(analysing_valuation_referee); (Cod_referee1 ≠ Cod_referee2 ≠

Cod_referee3) and (exists Oid(has_class_instance(resource, Oid) and exist Rid(has_role_instance(Oid, paper, Rid) and value(Rid, cod_paper) = Cod_paper) and Cod_referee1 ∈ value(Rid, cod_referee) and Cod_referee2 ∈ value(Rid, cod_referee) and Cod_referee3 ∈ value(Rid, cod_referee)),

/*a avaliação pela comissão só passará a ser feita quando chegarem os três pareceres com códigos de revisores diferentes*/

valuation:

state(analysing_valuation_referee), decision(valuation_end (Cod_paper, final_valuation)) ⇒ msg(final_ valuation((Cod_paper, final_valuation)),

revaluation:

state(analysing_valuation_referee), decision(revaluation (Cod_paper, Cod_referee, valuation, Observations) ⇒ msg(review_criterions(Cod_paper, Cod_referee, valuation, Observations))

}

>,

<Paper_valuation,

/* Agente Executor = (Referee) */

static propeties = {(Cod_referee,set of string) },

dynamic properties = {...},

states = {active, analysing_invitation, revising, waiting_answer, vacations, not_active },

messages = {

invitation(Cod_paper:string, Cod_referee:set of string, Abstract:string, Referee_name:string)
from PAPER_REVISION.Invited_referee,

accepted (Cod_referee: set of string, Cod_paper:set of string)
to PAPER_REVISION.Invited_referee,

naccepted (Cod_referee: set of string, Cod_paper:set of string)
to PAPER_REVISION.Invited_referee,

paper(Cod_paper:string, Cod_referee:set of string, paper:paper)
from PAPER_REVISION.Invited_referee,

valuation(Cod_paper:string, Cod_referee:set of string, valuation: set of string)
to PAPER_REVISION.Analyse_valuation,

review_criterions(Cod_paper:string, Cod_referee:set of string, valuation: set of string, Observations: set of string)
from PAPER_REVISION.Analyse_valuation

/* o paper deverá ser reavaliado segundo observações feitas pelo comitê de avaliação*/

},

decisions = {

accepted(Cod_referee: set of string, Cod_paper: string) ,

naccepted(Cod_referee: set of string, Cod_paper: string) ,
send_valuation(Cod_paper:string, Cod_referee:set of string, valuation: set of string)
},
rules = {

Begin:
msg(add_role) ⇒ state(active),

invitation_arrived:
state(active), msg(invitation(Cod_paper, Cod_referee, Abstract, Referee_name)) ⇒ state(analysing_invitation),

accepted_revise:
state(analysing_invitation), decision(Accept (Cod_referee, Cod_paper)) ⇒ msg(Accepted (Cod_referee, Cod_paper)), state(waiting_answer),

naccepted_revise:
state(analyse_invitation), decision(NAccept (Cod_referee, Cod_paper)) ⇒ msg(NAccepted (Cod_referee, Cod_paper)), state(active),

analysing_paper:
state(waiting_answer), msg(paper(Cod_paper, Cod_referee, paper)) ⇒ state(revising),

send_valuation:
state(revising), decision(send_valuation(Cod_paper, Cod_referee, valuation)⇒ valuation(Cod_paper, Cod_referee, valuation),state(active),

revaluation:
state(active), msg(review_criterions(Cod_paper, Cod_referee, valuation, Observations) ⇒ state(revising)
}
>,

---

resource class (

RECURSO,

<Base_role,
static propeties = {
(Cod_paper,string),
(Author_name, string),
(Cod_referee,set of string),
(Referee_name, set of string),
(Paper, paper),

(University, string),

(Valuation, set of string),

(Final_valuation, set of string)

},

dynamic properties = {(deadline, date) },

rules = {

msg(create_object) ⇒ msg(allow_role(paper), state(active))

}

>,

<Paper,

states = { revised, nrevised, revising},

messages = {

paper_values(Cod_paper: string, Author_name: string, Paper: text, University: string)

to PAPER_REVISION.Invited_referee

def_paper_values (Cod_paper: string, Author_name: string, Paper: text, University: string, final_valuation: set of string)

from PAPER_REVISION.Receive_analyse_paper

referee_paper_values(Cod_referee: set of string, Cod_paper:string)

from PAPER_REVISION.Invited_referee

report_paper(cod_paper: string, final_valuation: set of string)

from PAPER_REVISION.Receive_analyse_paper

},

rules = {

begin:

msg(add_role) ⇒ state (nrevised),

initialization:

state(nrevised), def_paper_values (Cod_paper, Author_name, Paper, University, final_valuation) ⇒ msg(paper_values(Cod_paper, Author_name, Paper, University)), state(nrevised);(exist Rid (value, cod_paper) = Cod_paper)),

referee_values:

state(nrevised), msg(referee_paper_values(Cod_referee, Cod_paper)) ⇒ state(revising)

revised:

state(revising), msg(report_paper(cod_paper, final_valuation) ⇒ state(revised)

}

>

Representação Administrativo

agent class (

PERSON,

<Base_role,

static propeties = {(name, string), (d_birth, date), (gender, {F,M}), }

dynamic properties = {(adress_pers, string),(e_mail, string), (telephone, set of integer), (specialisations, set of string), (University, string) }

rules = {

r1: state(active), msg(create_object) ⇒ msg(allow_role(Presidente_do_Comitê)),
r2: state(active), msg(create_object) ⇒ msg(allow_role(Secretária)),
r3: state(active), msg(create_object) ⇒ msg(allow_role(Substituto)),
r4: state(active), msg(create_object) ⇒ msg(allow_role(Referee))}

>,

<Presidente_do_Comitê,

static propeties = {...},

dynamic properties = {...},

states = {active, not active, vacations, analysing, analysing_answer_referee, waiting}

executor_tasks = { (PAPERS_REVISION.Invited_referee),
(PAPERS_REVISION.Analyse_paper_values),
(PAPERS_REVISION.Analyse_valuation)}

responsible_tasks = {(PAPERS_REVISION.Paper_receive)}

messages = {...},

decision = {

decisions_task(PAPERS_REVISION. Invited_referee): {
referee_paper_values(Referee_name: string, Cod_referee: string, Cod_paper:string)
},

decisions_task(PAPERS_REVISION.Analyse_paper_values): {
determine_paper_values (Cod_paper: string, Author_name: string, Paper: text, University: string),
accept(Author_name:string, result:string),
naccept(Author_name:string, result:string)
},

decisions_task(PAPERS_REVISION.Analyse_valuation): {
valuation_end(Cod_paper:string, avaliação_final: set of string), /*avaliação concluída*/
revaluation (Cod_paper:string, Cod_referee:string, valuation: set of string, Observations: set of string)
},

},

rules = {...},

>,

<Secretária,

static propeties = {...},

dynamic properties = {...},

states = {active, not active, vacations, analysing, waiting}

responsible_tasks = {...},

executor_tasks = {(PAPERS_REVISION.Paper_receive)}

messages = {... }

decisions = {...}

rules = {...}

}

>,

<Substituto,

static propeties = {...},

dynamic properties = {...},

states = {active, not active, vacations, analysing, waiting }

responsible_tasks = {...},

executor_tasks = {...},

messages = {...},

decisions = {...},

rules = {...},

>,

<Referee,

static propeties = {

(Cod_referee,string),

}

dynamic properties = {...},

states = {active, analysing_invitation, revising, waiting_answer, vacations }

responsible_tasks = {...},

executor_tasks = {(PAPERS_REVISION.Paper_valuation)}

messages = {...},

decisions = {

decisions_task(PAPERS_REVISION. Paper_valuation): {

active(Cod_referee: set of string, Cod_paper: string)

not_active(Cod_referee: set of string, Cod_paper: string)

send_valuation(Cod_paper:string, Cod_referee:set of string, valuation: set of string)

},

},

rules = {...},

>,

<Editor,

static propeties = {...},

dynamic properties = {...},

states = {(active), (not active), (vacations), (analyse_valuation_referee) }

responsible_tasks = {...},

executor_tasks = {...}

messages = {...},

decisions ={...}

rules = {...}

}

---

process class (

PAPERS_REVISION,

<Base_role,

static propeties = {...},

dynamic properties = {

(deadline, date)

},

rules = {

r1: msg(create_object) ⇒ state (active),

r2: state(active) ⇒ msg(add_role(paper_receive, presidente_do_comitê, secretária)),

r3: state(active) ⇒ msg(add_role(invited_referee, presidente_do_comitê, presidente_do_comitê),

r4: state(active) ⇒ msg(add_role(analyse_paper_values, presidente_do_comitê, presidente_do_comitê),

r5: state(active) ⇒ msg(add_role(analyse_valuation, presidente_do_comitê, presidente_do_comitê),

r6: state(active) ⇒ msg(add_role(paper_valuation, referee, referee))

/* segundo extensão proposta - add_role(Oid, papel, Rid, [Resp], [Exec], [Auto]) permite criar instâncias dos papéis dos processos, com agentes responsável e executor definidos */

}

>,

<Paper_receive,

/* Agente responsável = Presidente do comitê*/

/* Agente executor = secretária*/

states = { active, not active, waiting},

messages = {

paper_Author from EXTERNAL_WORLD
paper_receipt to EXTERNAL_WORLD
analyse_paper to PAPERS_REVISION.Analyse_paper_values
}

rules = {
Begin:
msg(paper_Author) ⇒ msg(create(Oid, paper)), msg(add_role(Oid,paper,Rid)),
msg(paper_receipt), msg(analyse_paper), state(active),
}
>,

<Invited_referee,

/*Agente executor = Presidente do comitê*/
states = { active, not active, waiting},
messages = {
paper_values(Cod_paper: string, Author_name: string, Paper: text, University: string)
from RECUSE.Paper
referee_not_active(Referee_name, Cod_referee, Cod_paper)
from PAPER_REVISION.Paper_valuation
referee_paper_values(Cod_referee: set of string, Cod_paper:string)
to RECURSO.Paper
paper(Cod_paper:string, Cod_referee:set of string, paper:paper)
to PAPER_REVISION.Paper_valuation
}

decisions = {
referee_paper_values(Referee_name, Cod_referee, Cod_paper)
},

rules = {
begin:
msg(add_role) ⇒ state(active)

Send_paper:
state(active), msg(paper_values(Cod_paper, Author_name, Paper, University))
decision(referee_paper_values(Referee_name, Cod_referee, Cod_paper)) ⇒
msg(referee_paper_values(Cod_referee, Cod_paper)), msg(paper(Cod_paper,
Cod_referee, paper), state(active);
value(Recurso.paper, university) ≠ value(Person.referee, university) and
(not exists Oid(has_class_instance(resource, Oid) and
exist Rid(has_role_instance(Oid, paper, Rid), value(Rid, cod_referee) =

```
    Cod_referee),
  /*não permite que um revisor seja convidado duas vezes para o mesmo artigo*/

  Referee_not_active:
  state(active), msg(referee_not_active(Referee_name, Cod_referee, Cod_paper)),
    decision(referee_paper_values(Referee_name, Cod_referee, Cod_paper)) ⇒
    msg(referee_paper_values(Cod_referee, Cod_paper)), msg(paper(Cod_paper,
    Cod_referee, paper), state(active);
  value(Recurso.paper, university) ≠ value(Person.referee, university) and
  (not exists Oid(has_class_instance(resource, Oid) and
    exist Rid(has_role_instance(Oid, paper, Rid), value(Rid, cod_referee) =
    Cod_referee),

  }
```

<Analyse_valuation,

```
/* Agente Executor = {( Presidente_do_comitê) } */
states = { active, not active, waiting},
messages = {
    valuation(Cod_paper:string, Cod_referee:set of string, valuation: set of string)
    from PAPER_REVISION.Paper_valuation
    /*valuation é inicialmente enviado ao presidente do comitê

    final_ valuation(Cod_paper:string, final_valuation:set of string) to
      PAPER_REVISION.Analyse_paper_values

    report_paper(cod_paper: string, final_valuation: set of string)
     to  RECURSO.Paper

    review_criterions(Cod_paper:string, Cod_referee:set of string, valuation: set of
      string, Observations: set of string) to PAPER_REVISION.Paper_valuation
    },
decisions ={
    valuation_end(Cod_paper:string, avaliação_final: set of string)
    /*avaliação concluída*/
    revaluation (Cod_paper:string, Cod_referee:string, valuation: set of string,
      Observations: set of string)
    },
rules = {
   receive_valuation:
    state(active),msg(valuation(Cod_paper, Cod_referee1, Valuation),
      msg(valuation(Cod_paper, Cod_referee2, Valuation)),
      msg(valuation(Cod_paper, Cod_referee3, Valuation))} ⇒
```

state(analysing_valuation_referee);
(Cod_referee1 □Cod_referee2 □Cod_referee3) and (exists
Oid(has_class_instance(resource, Oid) and
exist Rid(has_role_instance(Oid, paper, Rid) and value(Rid, cod_paper) =
Cod_paper) and value(Rid, cod_referee) ∈ Cod_referee),

/*a avaliação pela comissão só passará a ser feita quando chegarem os três pareceres com códigos de revisores diferentes*/

valuation:

state(analysing_valuation_referee), decision(valuation_end (Cod_paper, final_valuation)) ⇒ msg(final_ valuation(Cod_paper,final_valuation)), msg(report_paper(cod_paper: string, final_valuation: set of string)), state(active)

revaluation:

state(analysing_valuation_referee), decision(revaluation (Cod_paper, Cod_referee, valuation, Observations) ⇒ msg(review_criterions(Cod_paper, Cod_referee, valuation, Observations)), state(active)

},

<Analyse_paper_values,

/* Agente Executor = (Presidende_do_comitê) */

/*tarefas em que o executor é humano, poderão conter decisões*/

static propeties = {...},

dynamic properties = {...},

states = {active, not active, vacations, analysing_paper, analysing_answer_referee, waiting },

messages = {

analyse_paper from

PAPERS_REVISION.Paper_receive

def_paper_values(cod_paper:string, Author_name:string, Paper:text, University:string)
to RESCUSE.Paper

final_ valuation(Cod_paper:string, final_valuation:set of string)
from PAPER_REVISION.Analyse_valuation

report_paper_naccept to EXTERNAL_WORLD

report_paper_accept to EXTERNAL_WORLD

},

decisions = {

determine_paper_values (Cod_paper: string, Author_name: string, Paper: text, University: string)

accept(Author_name:string, result:string)

naccept(Author_name:string, result:string)

},

rules = {

Begin:

msg(add_role) ⇒ state(active)

Define_ paper_values:

state(active), msg(analyse_paper), decision(determine_paper_values (Cod_paper, Author_name, Paper, University) ⇒ msg(def_paper_values (Cod_paper, Author_name, Paper, University)), state(active); (exists Oid (has_class_instance(Recurso, Oid) and exist Rid(has_role_instance(Oid, paper, Rid)),

state(active), msg(final_ valuation(Cod_paper, final_valuation) ⇒ state(analysing),

state(analysing), decison(accept(Author_name, result)) ⇒ msg(report_paper_accept), state(active),

state(analysing), decison(naccept(Author_name, result)) ⇒ msg(report_paper_naccept), state(active),

}

>,

<Paper_valuation,

/* Agente Executor = (Referee) */

static propeties = {(Cod_referee,set of string) },

dynamic properties = {...},

states = {active, analysing_invitation, revising, waiting_answer, vacations, not_active },

messages = {

invitation(Cod_paper:string, Cod_referee:set of string, Abstract:string, Referee_name:string)
from PAPER_REVISION.Invited_referee

paper(Cod_paper:string, Cod_referee:set of string, paper:paper)
from PAPER_REVISION.Invited_referee

valuation(Cod_paper:string, Cod_referee:set of string, valuation: set of string)
to PAPER_REVISION.Analyse_valuation

review_criterions(Cod_paper:string, Cod_referee:set of string, valuation: set of string, Observations: set of string)
from PAPER_REVISION.Analyse_valuation

referee_not_active(Referee_name, Cod_referee, Cod_paper)
to PAPER_REVISION.Invited_referee
},
decisions = {
active(Cod_referee: set of string, Cod_paper: string)
not_active(Cod_referee: set of string, Cod_paper: string)
send_valuation(Cod_paper:string, Cod_referee:set of string, valuation: set of string)
},
rules = {
Begin:
msg(add_role) ⇒ state(active)

not_active:
state(active), paper(Cod_paper, Cod_referee, paper), decision(not_active (Cod_referee, Cod_paper)) ⇒ referee_not_active(Referee_name, Cod_referee, Cod_paper)
, state(not_active),

analysing_paper:
state(active), paper(Cod_paper, Cod_referee, paper),), decision(active (Cod_referee, Cod_paper)) ⇒ state(revising),

send_valuation:
state(revising), decision(send_valuation(Cod_paper, Cod_referee, valuation)⇒ valuation(Cod_paper, Cod_referee, valuation),state(active)

revaluation:
state(active), msg(review_criterions(Cod_paper, Cod_referee, valuation, Observations) ⇒
state(revising)
}
>,

---

resource class (

RECURSO,

<Base_role,
static propeties = {
(Cod_paper,string),

```
        (Author_name, string),
        (Cod_referee,set of string),
        (Referee_name, set of string),
        (Paper, paper),
        (University, string),
        ( Valuation, set of string),
        ( Avaliação_final, set of string)
        },
    dynamic properties = {(deadline, date) },
    rules = {
        msg(create_object) ⇒ msg(allow_role(paper), state(active))
        }
        >,

<Paper,
    states = { revised, nrevised, revising}
    messages = {
        def_paper_values (Cod_paper: string, Author_name: string, Paper: text,
            University: string)
        from PAPER_REVISION.Analyse_paper_values

        paper_values(Cod_paper: string, Author_name: string, Paper: text, University:
            string)
        to PAPER_REVISION.Invited_referee

        referee_paper_values(Cod_referee: set of string, Cod_paper:string)
        from PAPER_REVISION.Invited_referee

        report_paper(cod_paper: string, final_valuation: set of string)
        from PAPER_REVISION.Receive_analyse_paper

        },
    rules = {
        begin:
        msg(add_role) ⇒ state (nrevised),
        initialization:
        state(nrevised), def_paper_values (Cod_paper, Author_name, Paper, University)
            ⇒ msg(paper_values(Cod_paper, Author_name, Paper, University),
            state(nrevised);
        (not exist Rid (value, cod_paper) = Cod_paper)),
        referee_values:
        state(nrevised), msg(referee_paper_values(Cod_referee, Cod_paper)) ⇒
```

state(revising),

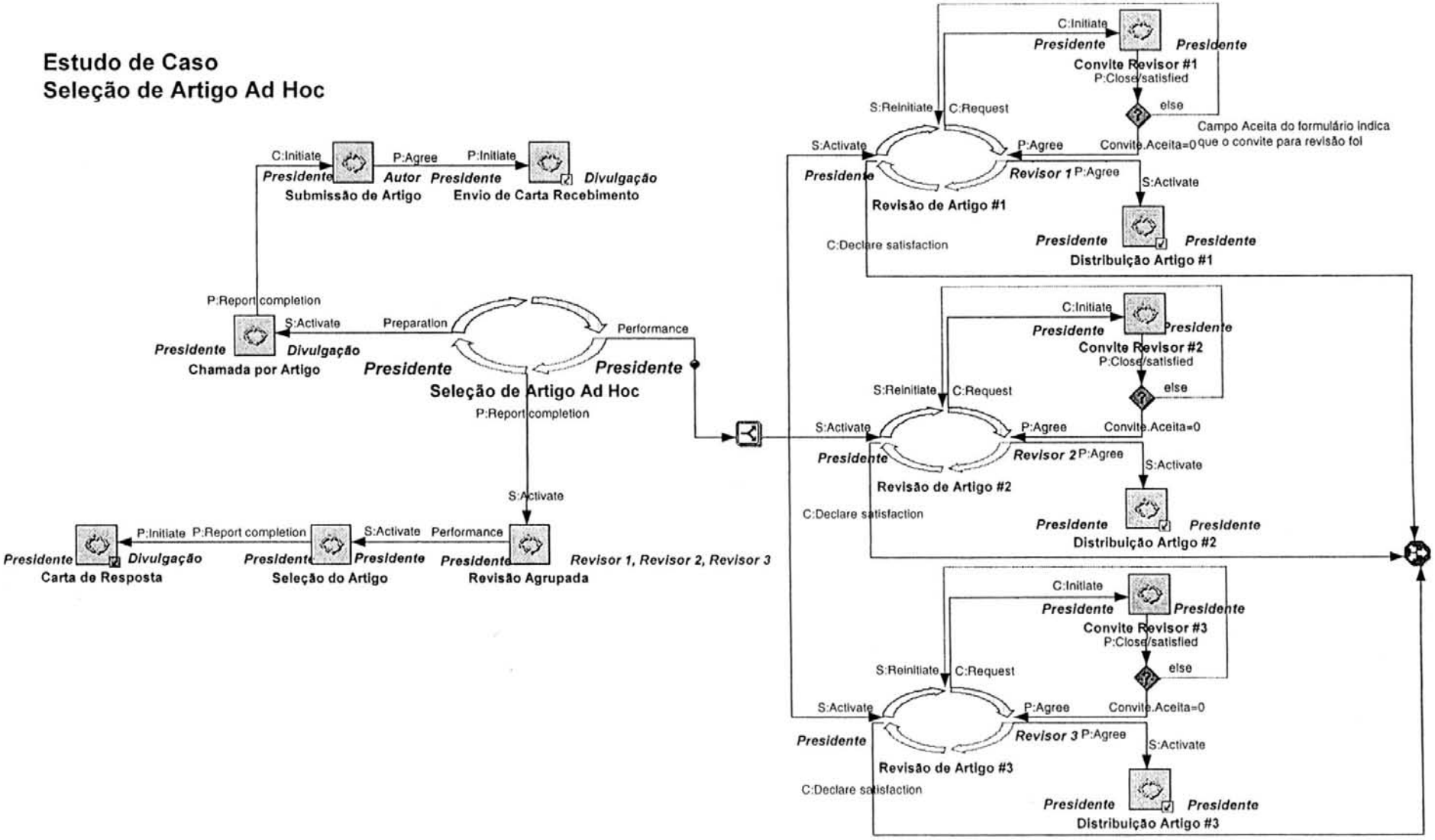
revised:

state(revising), msg(report_paper(cod_paper, final_valuation) ⇒ state(revised),

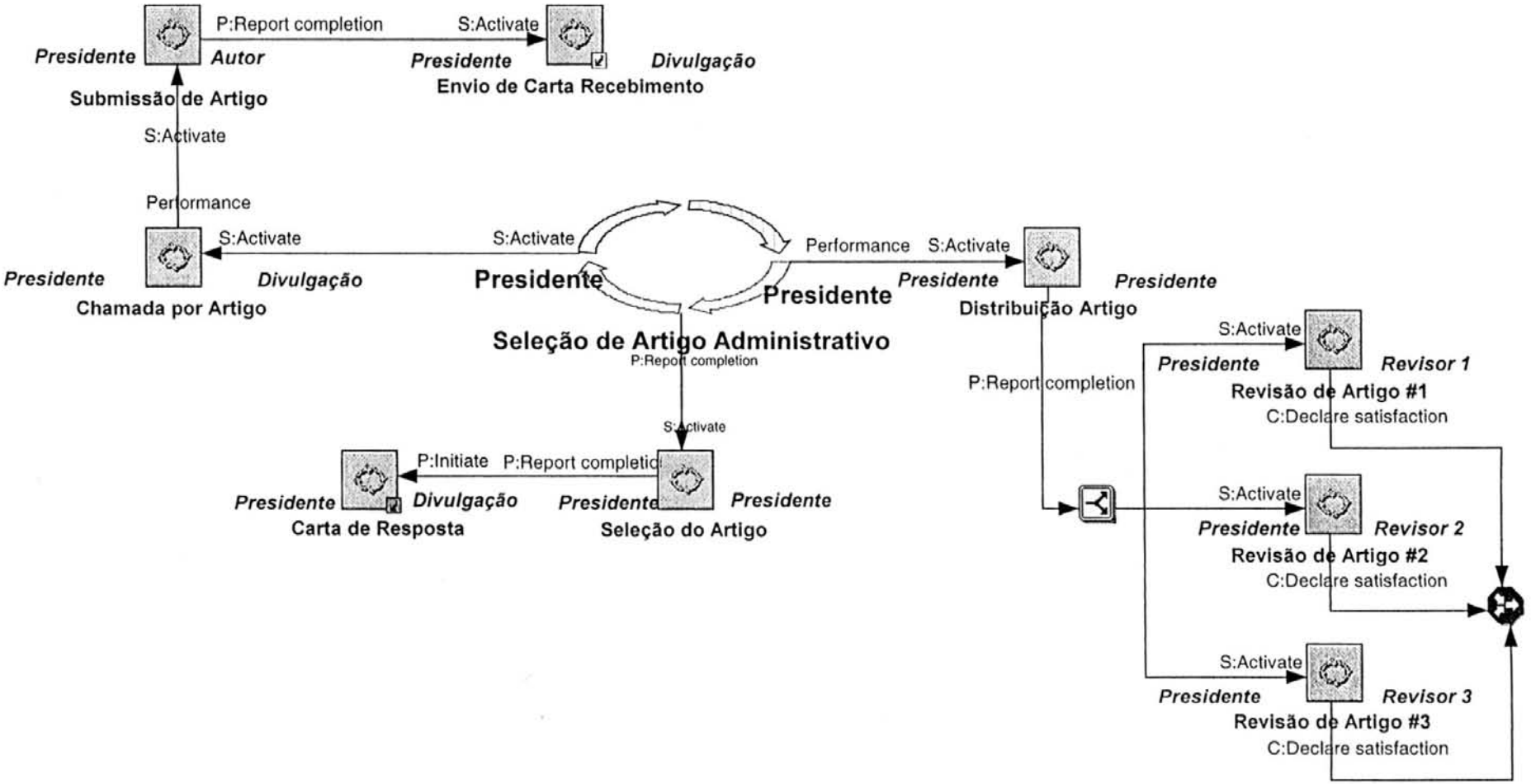
}

>

ANEXO 3 - Modelagem do Estudo de Caso no PROCESS BUILDER (Action *Workflow*)

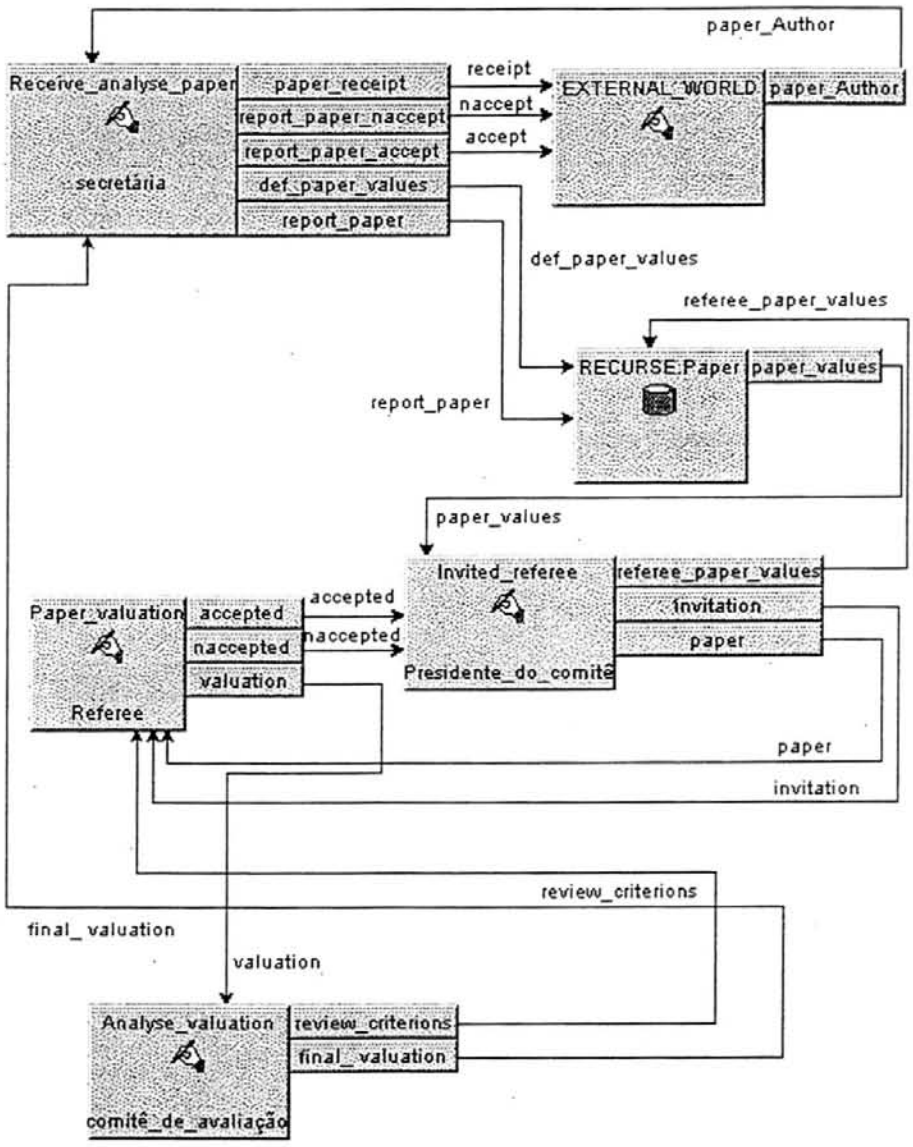


Estudo de Caso
Seleção de Artigo Administrativo

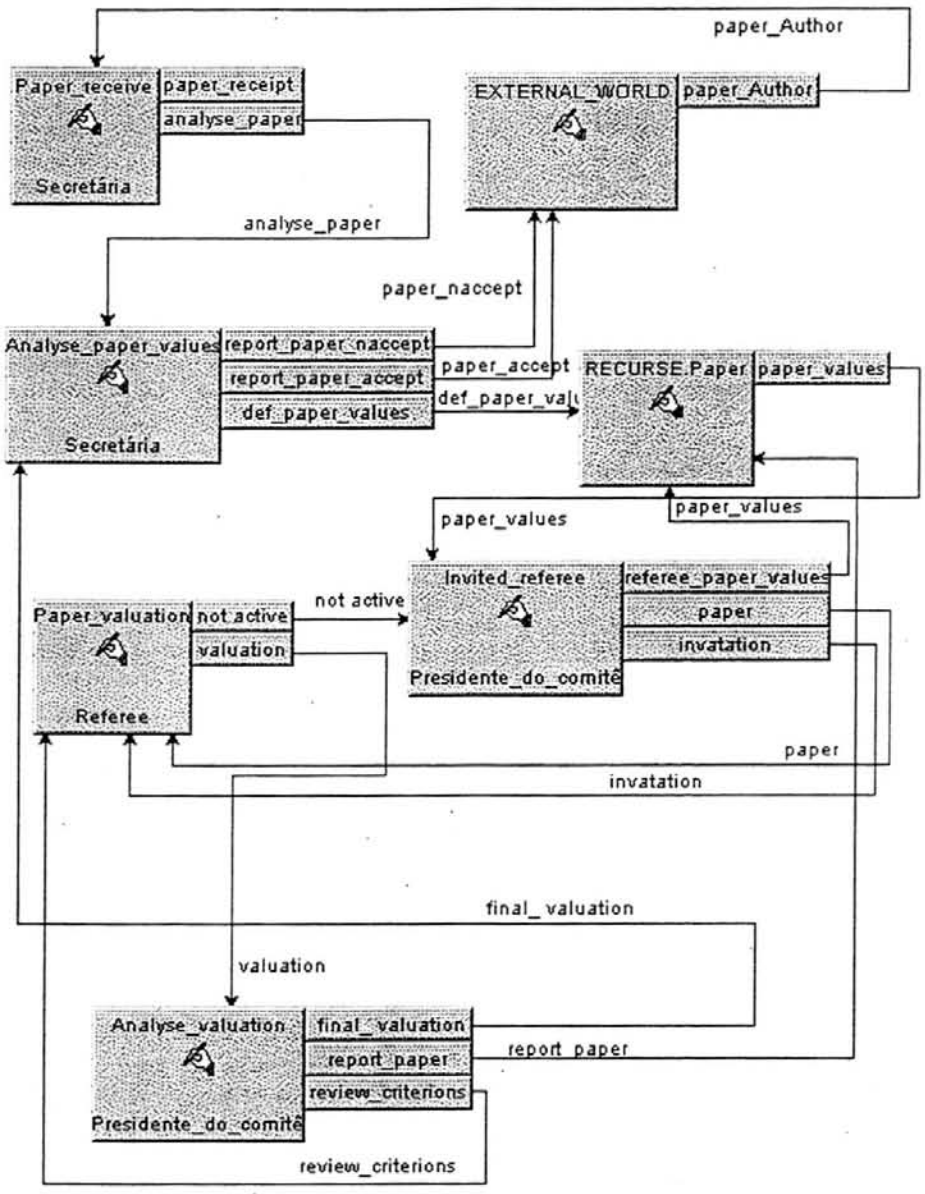


ANEXO 4 - Modelagem do Estudo de Caso no JETFORM WORKFLOW

Representação Ad Hoc



Representação Administrativo



Informática  UFRGS

CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

"Modelagem de Workflow Utilizando um Modelo de Dados Temporal Orientado a Objetos com Papéis"

por

Mariano Nicolao

Dissertação apresentada aos Senhores:

Profa. Dra. Rosa Maria Viccari

Prof. Dr. Clesio Saraiva dos Santos

Prof. Dr. Jacques Wainer (UNICAMP)

Vista e permitida a impressão.
Porto Alegre, __/__/____.

Profa. Dra. Nina Edelweiss,
Orientador.