

**UNIVERSIDADE FEDERAL DE SANTA CATARINA  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA  
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO**

**UM ESTUDO SOBRE PADRÕES DE WORKFLOW E SUAS  
IMPLEMENTAÇÕES EM WORKFLOW MANAGEMENT SYSTEMS**

**Fábio Dall'Oglio da Cunha**

**Florianópolis – SC**

**2005**

**UNIVERSIDADE FEDERAL DE SANTA CATARINA  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA  
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO**

**UM ESTUDO SOBRE PADRÕES DE WORKFLOW E SUAS  
IMPLEMENTAÇÕES EM WORKFLOW MANAGEMENT SYSTEMS**

**Fábio Dall'Oglio da Cunha**

Trabalho de conclusão de curso  
apresentado como parte dos requisitos  
para obtenção do grau de Bacharel em  
Sistemas de Informação

**Florianópolis – SC  
2005**

**UNIVERSIDADE FEDERAL DE SANTA CATARINA  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA  
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO**

**ORIENTADOR:**

---

**VITÓRIO BRUNO MAZZOLA**

**BANCA EXAMINADORA:**

---

**RICARDO PEREIRA E SILVA**

---

**SÉRGIO WEBER**

---

**GISELE F. PELLEGRINI**

## **AGRADECIMENTOS**

A Deus, que permitiu que eu concluísse esta importante etapa de minha vida, e que foi meu refúgio nos momentos de desânimo.

Ao meu orientador, Professor Vitório Bruno Mazzola, por ter aceitado o convite e por ter contribuído com o direcionamento deste trabalho.

Ao colega e amigo Sérgio Weber, por ter aceitado o convite de fazer parte da banca examinadora e pelo tempo dedicado a contribuir com este trabalho.

Ao Professor Ricardo Pereira e Silva e à Gisele Pellegrini, por terem aceitado prontamente o convite de fazerem parte da banca examinadora.

Aos colegas da Nexxera, pelo apoio dado durante a execução deste trabalho, em especial ao Sr. Clovis Ramayana Carrão e à equipe de planejamento e marketing.

À minha querida esposa Cris, pela compreensão, apoio, companheirismo e amor demonstrados nesta fase conturbada que foi execução deste trabalho.

Aos meus pais, João e Elisabete, por terem dado a mim a educação que tenho hoje, e por serem exemplos de caráter e integridade em minha vida.

Aos meus sogros, Arno e Erna, por estarem sempre orando e torcendo por mim.

Aos colegas e professores, que ao longo desses quatro anos e meio fizeram parte de minha vida.

## RESUMO

Este trabalho tem por objetivo estudar Padrões de *Workflow* e analisar como estes padrões são implementados em *Workflow Management Systems*. Para tanto, desenvolveu-se um estudo específico sobre *workflow*, processos de negócio, padrões de projeto, padrões de *workflow* e *workflow management systems*. Realizou-se um aprofundamento em padrões de *workflow*, onde foram descritos e estudados em detalhes todos os padrões propostos para resolverem problemas em processos de negócio em sistemas computacionais. Fez-se ainda um estudo sobre *workflow management systems*, onde foram analisadas as principais ferramentas de gerenciamento de *workflow*, tanto comerciais quanto de código aberto, disponíveis no mercado. A partir deste estudo, identificaram-se ferramentas de *workflow* que implementam todos os padrões propostos. Por fim, elaborou-se uma descrição e análise detalhada de como os padrões de *workflow* são implementados nestas ferramentas.

**Palavras-Chave:** Processos de Negócio, *Workflow*, Padrões de Projeto, Padrões de *Workflow*, *Workflow Management Systems*, WFMS, jBPM, OpenWFE

## LISTA DE SIGLAS E ABREVIATURAS

API	<i>Application Program Interface</i>
BPM	<i>Business Process Management</i>
BPMT	<i>Business Process Modeling Tools</i>
BPR	<i>Business Process Reengineering</i>
CORBA	<i>Common Object Request Broker Architecture</i>
CPI	<i>Continuous Process Improvement</i>
GUI	<i>Graphical User Interface</i>
HTTP	<i>Hypertext Transfer Protocol</i>
jPDL	<i>jBPM Process Definition Language</i>
OMG	<i>Object Management Group</i>
SOAP	<i>Simple Object Access Protocol</i>
TI	Tecnologia da Informação
UML	<i>Unified Modeling Language</i>
WfMC	<i>Workflow Management Coalition</i>
WFMS	<i>Workflow Management Systems</i>
XPDL	<i>XML Process Definition Language</i>

## LISTA DE FIGURAS

Figura 1: Exemplo de Integração Promovida pela Tecnologia <i>Workflow</i> .....	11
Figura 2: Ciclo de Vida dos Processos de Negócio.....	15
Figura 3: Características de Sistemas <i>Workflow</i> .....	17
Figura 4: Implementação do padrão <i>sequence</i> .....	22
Figura 5: Implementação do padrão <i>parallel split</i> .....	23
Figura 6: Implementação do padrão <i>synchronization</i> .....	24
Figura 7: Implementação do padrão <i>exclusive choice</i> .....	24
Figura 8: Implementação do padrão <i>simple merge</i> .....	25
Figura 9: Padrões de projeto para o padrão <i>multi-choice</i> .....	27
Figura 10: Implementação do padrão <i>synchronizing merge</i> .....	28
Figura 11: Implementação típica do padrão <i>multi-merge</i> .....	29
Figura 12: Implementação do padrão <i>discriminator</i> .....	31
Figura 13: Exemplo de implementação de <i>arbitrary cycles</i> .....	33
Figura 14: Padrões de projeto para múltiplas instâncias .....	38
Figura 15: Padrões de projeto para múltiplas instâncias .....	40
Figura 16: Ilustrando a diferença entre <i>XOR-splits</i> implícitos e explícitos.....	41
Figura 17: Estratégias para implementação de <i>deferred choice</i> .....	43
Figura 18: As opções de implementação para execução intercalada de <i>A, B e C</i> .....	45
Figura 19: A execução de <i>A, B e C</i> é intercalada adicionando-se um local de exclusão mútua .....	46
Figura 20: Representação esquemática de um <i>milestone</i> .....	47
Figura 21: O padrão <i>milestone</i> nas suas mais simples formas .....	48
Figura 22: Implementação do padrão <i>cancel activity</i> .....	50
Figura 23: Implementação do padrão <i>cancel case</i> .....	51

## LISTA DE TABELAS

Tabela 1: WFMS de Código Aberto x Padrões de Projeto.....	64
Tabela 2: WFMS Comerciais x Padrões de Projeto (I) .....	65
Tabela 3: WFMS Comerciais x Padrões de Projeto (II).....	66



# SUMÁRIO

<b>1. INTRODUÇÃO .....</b>	<b>10</b>
1.1. JUSTIFICATIVA .....	12
1.2. OBJETIVO GERAL .....	12
1.3. OBJETIVOS ESPECÍFICOS .....	12
1.4. ESTRUTURA DO DOCUMENTO .....	13
<b>2. WORKFLOW E CONCEITOS RELACIONADOS .....</b>	<b>14</b>
2.1. PROCESSO DE NEGÓCIO .....	14
2.2. WORKFLOW .....	16
<b>3. PADRÕES .....</b>	<b>19</b>
3.1. PADRÕES DE PROJETO (DESIGN PATTERNS) .....	19
3.2. PADRÕES DE WORKFLOW (WORKFLOW PATTERNS) .....	20
3.2.1. <i>Padrões de Controle de Fluxo Básicos</i> .....	22
3.2.1.1. Padrão 1 (Sequence) .....	22
3.2.1.2. Padrão 2 (Parallel Split) .....	22
3.2.1.3. Padrão 3 (Synchronization) .....	23
3.2.1.4. Padrão 4 (Exclusive Choice) .....	24
3.2.1.5. Padrão 5 (Simple Merge) .....	25
3.2.2. <i>Padrões de Ramificação e Sincronização Avançados</i> .....	25
3.2.2.1. Padrão 6 (Multi-choice) .....	26
3.2.2.2. Padrão 7 (Synchronizing Merge) .....	27
3.2.2.3. Padrão 8 (Multi-merge) .....	28
3.2.2.4. Padrão 9 (Discriminator) .....	30
3.2.2.5. Padrão 9a (N-out-of-M-join) .....	31
3.2.3. <i>Padrões Estruturais</i> .....	32
3.2.3.1. Padrão 10 (Arbitrary Cycles) .....	32
3.2.3.2. Padrão 11 (Implicit Termination) .....	33
3.2.4. <i>Padrões envolvendo Múltiplas Instâncias</i> .....	34
3.2.4.1. Padrão 12 (Multiple Instances Without Synchronization) .....	34
3.2.4.2. Padrão 13 (Multiple Instances With a Priori Design Time Knowledge) .....	35
3.2.4.3. Padrão 14 (Multiple Instances With a Priori Runtime Knowledge) .....	35
3.2.4.4. Padrão 15 (Multiple Instances Without a Priori Runtime Knowledge) .....	38
3.2.5. <i>Padrões Baseados em Estado</i> .....	40
3.2.5.1. Padrão 16 (Deferred Choice) .....	41
3.2.5.2. Padrão 17 (Interleaved Parallel Routing) .....	43
3.2.5.3. Padrão 18 (Milestone) .....	46
3.2.6. <i>Padrões de Cancelamento</i> .....	48
3.2.6.1. Padrão 19 (Cancel Activity) .....	49
3.2.6.2. Padrão 20 (Cancel Case) .....	50
<b>4. WORKFLOW MANAGEMENT SYSTEMS .....</b>	<b>52</b>
4.1. WFMS DE CÓDIGO ABERTO .....	52
4.1.1. <i>JBoss jBPM</i> .....	52
4.1.2. <i>OpenWFE</i> .....	54
4.1.3. <i>OpenSymphony Workflow – OSWorkflow</i> .....	55
4.1.4. <i>WfMOpen</i> .....	55
4.1.5. <i>OpenFlow</i> .....	56
4.1.6. <i>ObjectWeb Bonita</i> .....	56
4.2. WFMS COMERCIAIS .....	57
4.2.1. <i>TIBCO Staffware Process Suite</i> .....	57
4.2.2. <i>COSA BPM</i> .....	58
4.2.3. <i>InConcert</i> .....	59
4.2.4. <i>MQSeries/Workflow</i> .....	59
4.2.5. <i>Forté Conductor</i> .....	60
4.2.6. <i>Verve</i> .....	61

4.2.7. <i>Visual WorkFlo</i> .....	61
4.2.8. <i>I-Flow</i> .....	62
4.2.9. <i>SAP R/3 Workflow</i> .....	62
4.3. <b>WFMS X PADRÕES DE WORKFLOW</b> .....	64
<b>5. PADRÕES DE WORKFLOW NO JBPM E OPENWFE</b> .....	<b>67</b>
5.1. PADRÃO 01 (SEQUENCE).....	67
5.2. PADRÃO 02 (PARALLEL SPLIT) .....	68
5.3. PADRÃO 03 (SYNCHRONIZATION).....	69
5.4. PADRÃO 04 (EXCLUSIVE CHOICE) .....	70
5.5. PADRÃO 05 (SIMPLE MERGE) .....	72
5.6. PADRÃO 06 (MULTI-CHOICE).....	73
5.7. PADRÃO 07 (SYNCHRONIZING MERGE) .....	74
5.8. PADRÃO 08 (MULTI-MERGE) .....	75
5.9. PADRÃO 09 (DISCRIMINATOR).....	77
5.10. PADRÃO 10 (ARBITRARY CYCLES) .....	78
5.11. PADRÃO 11 (IMPLICIT TERMINATION) .....	79
5.12. PADRÃO 12 (MI WITHOUT SYNCHRONIZATION).....	80
5.13. PADRÃO 13 (MI WITH A PRIORI DESIGN TIME KNOWLEDGE) .....	82
5.14. PADRÃO 14 (MI WITH A PRIORI RUNTIME KNOWLEDGE).....	83
5.15. PADRÃO 15 (MI WITHOUT A PRIORI RUNTIME KNOWLEDGE) .....	85
5.16. PADRÃO 16 (DEFERRED CHOICE) .....	86
5.17. PADRÃO 17 (INTERLEAVED PARALLEL ROUTING) .....	87
5.18. PADRÃO 18 (MILESTONE).....	89
5.19. PADRÃO 19 (CANCEL ACTIVITY).....	90
5.20. PADRÃO 20 (CANCEL CASE) .....	92
<b>6. CONCLUSÃO</b> .....	<b>93</b>
<b>REFERÊNCIAS BIBLIOGRÁFICAS</b> .....	<b>95</b>
<b>ANEXO 1 – ARTIGO</b> .....	<b>98</b>

# 1. Introdução

Estamos numa era onde a competitividade entre empresas nunca esteve tão acirrada. Empresas tradicionais têm modificado radicalmente sua maneira de atuar no mercado, utilizando-se de estratégias de marketing, reduzindo o ciclo de vida de seus principais produtos, aumentando a velocidade de produção em suas fábricas, geralmente através da reconfiguração de seus processos de fabricação, e se esforçando cada vez mais para atender da melhor maneira possível às necessidades de seus clientes [SILVA 01].

Segundo Davenport, um dos motivadores para estas transformações é a Tecnologia da Informação (TI) [DAVENPORT 94]. A razão disso é que a tecnologia da informação tem dado grandes saltos evolutivos nos últimos anos, resultando na criação de maneiras completamente novas de organizar processos de negócio. O desenvolvimento de pacotes genéricos de software para gerenciar processos de negócio, no caso os *Workflow Management Systems* (WFMS), é particularmente importante no que diz respeito a isto [AALST 02a].

Segundo Amaral, *Workflow* pode ser definido como uma coleção de tarefas organizadas para realizar um processo de negócio. Uma tarefa pode ser executada por um sistema de computador, por um agente humano ou pela combinação destes dois [AMARAL 97].

Quanto aos aspectos operacionais, os benefícios advindos do uso de WFMS são:

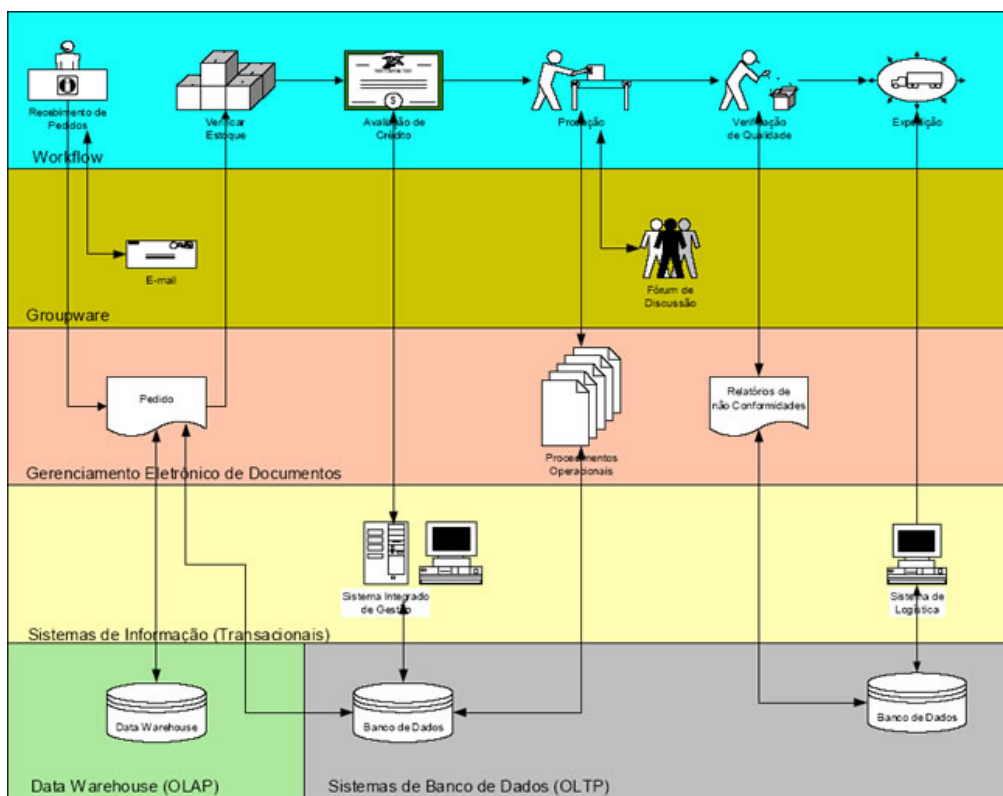
- Execução de processo seguro e confiável.
- Esforço reduzido de implementação em aplicações de processo.
  - Projetos gráficos.
  - Capacidade de condução.
  - Gerenciamento de dados e aplicativos.
  - Modelagem Organizacional.
  - Dependências Inter-processos.
  - Reutilização de partes de processo.
- Transparência e flexibilidade.

Quanto aos aspectos estratégicos, os benefícios são:

- Processo melhorado e qualidade de produto.

- Engenharia simultânea.
  - *Time-to-market* reduzido.
  - Custos reduzidos.
- Monitoramento e controle de processos.
- Flexibilidade para adaptar processos rapidamente, facilmente e com baixos custos.
- Integração de processos de fornecedores.
- Flexibilidade para adaptar relacionamentos com fornecedores competitivamente.

Sistemas de *Workflow* não substituem as já tradicionais ferramentas de automação de escritório, como processadores de texto e planilhas eletrônicas, nem os sistemas corporativos da empresa, como sistemas de contas a pagar, contabilidade, folha de pagamento, planejamento e controle de produção etc. Ao invés disso permitem uma utilização integrada dessas diversas ferramentas para a execução otimizada de todo o processo. A figura 1 apresenta uma representação da tecnologia *Workflow* agindo como integradora para outras tecnologias [SILVA 01].



**Figura 1: Exemplo de Integração Promovida pela Tecnologia *Workflow* [SILVA 01]**

## 1.1. Justificativa

Existem hoje no mercado inúmeros WFMS. A empresa que deseja utilizar um WFMS deve se adaptar à ferramenta, que muitas vezes não cobre peculiaridades dos processos de negócio da mesma, ou mesmo é de difícil adaptação aos sistemas utilizados na empresa. Uma alternativa ao uso de WFMS disponíveis no mercado é o desenvolvimento de um sistema próprio, que atenda todas as necessidades da empresa.

Com isso surge um problema: à medida que a quantidade de processos da empresa que serão sistematizados por meio de *Workflow* cresce, muitas estruturas semelhantes de *Workflow* são reprojatadas e reimplementadas, para cada processo. Assim, não há reutilização de software e o desenvolvimento se torna caro e demorado.

Por esta razão, a adoção do conceito de *padrões de projeto* orientados a *workflows* para atuar como elementos que favoreçam a reutilização de software e, desta forma, permitam agilizar o desenvolvimento de software que implemente os diversos workflows foi o tema escolhido para o desenvolvimento de nosso trabalho.

## 1.2. Objetivo Geral

Este trabalho tem por objetivo identificar e estudar *Workflow Management Systems* (WFMS) disponíveis no mercado, principalmente os de código aberto, que implementem os Padrões de *Workflow*. Pretende-se verificar como estes WFMS implementam os Padrões de *Workflow* propostos e se os mesmos dão suporte adequado ao desenvolvimento de *Workflow Management Systems* próprios.

## 1.3. Objetivos Específicos

Os objetivos específicos deste projeto são:

- Estudo de tópicos relacionados a processos de negócios em empresas.
- Estudo aprofundado da tecnologia *Workflow*, *Workflow Management Systems*, Padrões de Projeto<sup>1</sup> e Padrões de *Workflow*, que será peça fundamental na execução deste projeto.

---

<sup>1</sup> O termo “Padrões de Projeto” deriva de “*Design Patterns*”. Neste trabalho será utilizada a notação em português.

- Pesquisa de trabalhos envolvendo os tópicos citados anteriormente, que possam servir de base à proposta a ser desenvolvida no contexto deste projeto;
- Elaboração de um estudo aprofundado das ferramentas de *workflow* de código aberto que suportam os padrões de *workflow*, a resultar deste trabalho.

#### **1.4. Estrutura do Documento**

Este projeto está organizado em seis capítulos, descritos a seguir:

O Capítulo 2 apresenta os conceitos de processos de negócio e *workflow* que, juntamente com o capítulo 3, serão o alvo desta pesquisa, e às quais servirão de base para a execução do projeto. Neste capítulo estarão descritos: Processos de Negócio, *Workflow* e conceitos relacionados a *Workflow*.

O Capítulo 3 apresenta os conceitos de Padrões, no caso Padrões de Projeto e Padrões de *Workflow*. Além disso, todos os padrões de *Workflow* propostos estão descritos neste capítulo. Os estudos realizados nos capítulos 2 e 3 fazem-se necessários para a compreensão dos conceitos citados, tendo em vista que os mesmos são de essencial importância para o trabalho.

O Capítulo 4 apresenta os WFMS presentes no mercado, tanto os de código aberto<sup>2</sup> quanto os comerciais. Neste capítulo também são apresentadas tabelas que relacionam os WFMS aos padrões suportados pelos mesmos.

O Capítulo 5 apresenta o estudo aprofundado das ferramentas de *workflow* de código aberto que suportam todos os padrões de *workflow*. Neste capítulo são descritas e analisadas as estratégias de implementação de padrões de *workflow* nestas ferramentas

As conclusões e as perspectivas para trabalhos futuros encontram-se descritas no capítulo 6.

---

<sup>2</sup> O termo “Código Aberto” deriva de “*Open Source*”. Neste trabalho será utilizada a notação em português.

## 2. Workflow e Conceitos Relacionados

A seguir, as definições das tecnologias que serão utilizadas no projeto.

### 2.1. Processo de Negócio

Processos de negócio são coleções de uma ou mais atividades ligadas que concretizam um objetivo de negócio ou uma diretriz a ser alcançada, como, por exemplo, o fechamento de um contrato de negócio, e/ou a satisfação de uma necessidade específica de um cliente [GEORGAKOPOULOS 97].

Processos de negócio são descrições das atividades de uma organização, focadas no mercado. Isto é, processos de negócio são compilações de atividades que dão suporte às funções organizacionais críticas, ao realizar um objetivo ou diretriz a ser alcançada [MEDINA-MORA 92].

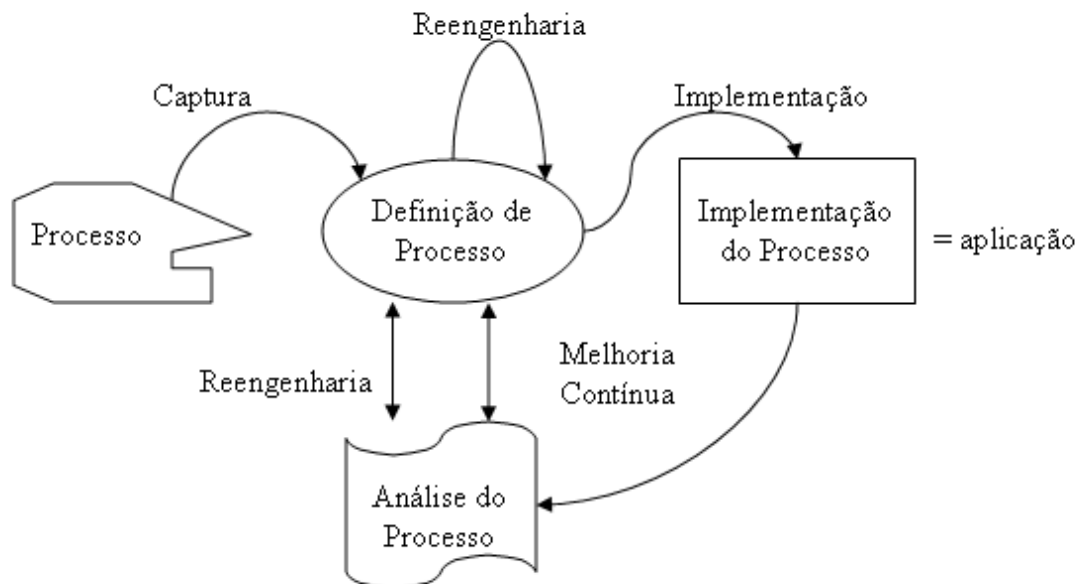
Processos de informação estão relacionados a atividades automatizadas, como as executadas por programas, e a atividades parcialmente automatizadas, como as executadas por humanos interagindo com computadores, que criam, processam, gerenciam e fornecem informação. Tipicamente um processo de informação envolve distribuir e coordenar atividades de trabalho entre humanos e recursos de sistemas de informação.

*Business Process Reengineering* (BPR), ou Reengenharia de Processo de Negócio, é a atividade de capturar processos de negócios iniciando de uma folha branca de papel, um modelo em branco no computador, documento, ou repositório. Uma vez que uma organização captura seus negócios em termos de processos de negócios, ela pode medir cada processo para melhorá-lo ou adaptá-lo aos requisitos que mudam. *Continuous Process Improvement* (CPI) envolve medições, reconsiderações, e re-projeto do processo de negócio.

O ciclo de vida de um processo de negócio envolve tudo que diz respeito à captura do processo em uma representação computadorizada para automatizar o mesmo, como, por exemplo, implementando um processo através de *Workflow*. Isto tipicamente inclui atividades explícitas de medição, análise, e melhoria do processo. A necessidade de gerenciar efetivamente o ciclo de vida do processo de negócio, como, por exemplo, aplicando *Business Process Management* (BPM), tem conduzido ao desenvolvimento

de novos conceitos ferramentas interoperáveis que suportam aspectos complementares de gerenciamento de processos de negócio.

A figura 2 mostra o ciclo de vida dos processos de negócio, apresentado por GEORGAKOPOULOS & TSALGATIDOU.



**Figura 2: Ciclo de Vida dos Processos de Negócio [GEORGAKOPOULOS 97]**

De acordo com esse conceito, todo processo possui um ciclo de vida que passa necessariamente por quatro estágios, a saber: Captura, Reengenharia, Implementação e Melhoria contínua. Se esses estágios forem bem conhecidos, e adequadamente conduzidos, o gerenciamento dos processos de negócio de uma organização pode se tornar efetivo em relação a seu potencial de ganho [SILVA 01].

Hoje em dia, produtos disponíveis no mercado que suportam o gerenciamento de processos de negócio podem ser caracterizados como *Workflow Management Systems* (WFMS) e *Business Process Modeling Tools* (BPMT). Ambos suportam definição ou especificação de processo de negócio. Entretanto, enquanto o escopo da definição de processo em BPMTs é fornecer entendimento e análise ao processo, que pode conduzir à melhoria do mesmo, o escopo de definição de processos em WFMS é dar suporte à automação do processo [GEORGAKOPOULOS 97]. Este trabalho terá seu foco em WFMS.



## 2.2. Workflow

Segundo a *Workflow Management Coalition*, *Workflow* é a facilitação ou automação de um processo de negócio, como um todo ou em parte. *Workflow* é preocupado com a automação de procedimentos onde documentos, informações ou tarefas são passados entre participantes do processo, de acordo com um conjunto de regras que se queiram alcançar, ou contribuir para, um objetivo de negócio como um todo [WfMC 95].

Embora um *Workflow* possa ser organizado manualmente, na prática a maioria dos *Workflow* é organizada dentro de um contexto de um sistema de TI, visando prover suporte informatizado para a automação procedural [WfMC 95].

*Workflow* é frequentemente associado com BPR, que é preocupado com a avaliação, análise, modelagem, definição e subsequente implementação operacional dos processos de negócio centrais de uma organização. Embora nem toda BPR resulte em implementações de *Workflow*, a tecnologia *Workflow* é, muitas vezes, uma solução apropriada, visto que ela proporciona separação entre lógica da regra de negócio e seu suporte operacional de TI, permitindo mudanças subsequentes a serem incorporadas nas regras procedurais que definem o processo de negócio.

*Workflow Management System* (WFMS) é um sistema para definição, criação e gerência da execução de fluxos de trabalho através do uso de software, capaz de interpretar a definição de processos, interagir com seus participantes e, quando necessário, invocar ferramentas e aplicações [WfMC95].

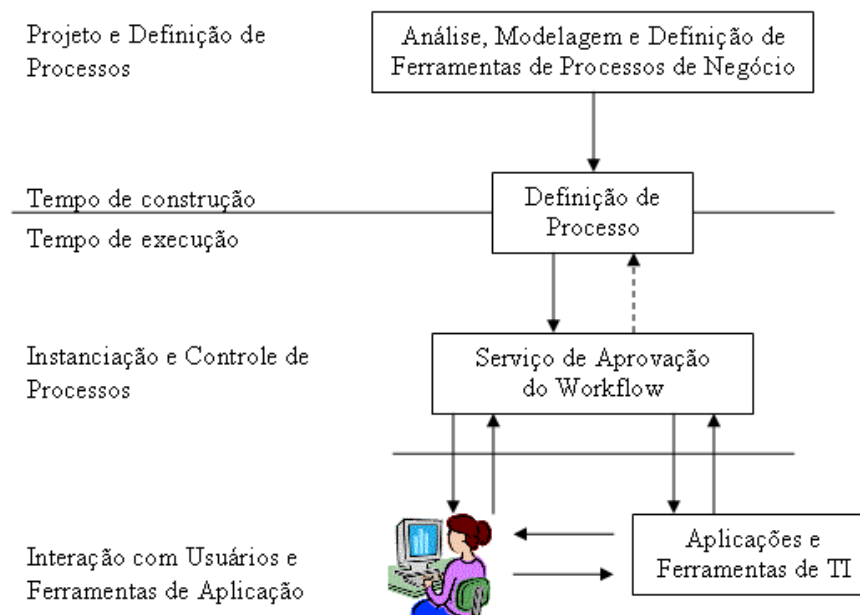
Todos os WFMS exibem certas características em comum, que fornecem uma base para integração de desenvolvimento e capacidade de interoperabilidade entre diferentes produtos.

No nível mais alto, todos os WFMS podem ser caracterizados como fornecendo suporte em três áreas funcionais:

- As funções de tempo de construção, preocupadas com a definição, e possivelmente modelagem do processo de *Workflow* e as atividades que o constituem.
- As funções de controle de tempo de execução, preocupadas com a gerência dos processos de *Workflow* em um ambiente operacional e com a sucessão das várias atividades a serem controladas como parte de cada processo.

- As interações em tempo de execução, com humanos e aplicações de TI, para processar os vários passos de atividade.

A figura 3 ilustra as características básicas de WFMS e os relacionamentos entre as principais funções citadas anteriormente [WfMC 95].



**Figura 3: Características de Sistemas Workflow [WfMC 95]**

Com o WFMS automatizado: [FISCHER 02]

- Trabalho não fica sem rumo nem parado – expedidores são raramente necessários para recuperar de erros ou falta de gerência do trabalho.
- Os gerentes podem focalizar nos aspectos de equipe e negócios, tais como performance individual, procedimentos ideais, e casos especiais, mais do que a rotina de atribuição de tarefas. Não é necessário um exército para distribuir e conduzir o trabalho.
- Os procedimentos são formalmente documentados e seguidos com exatidão, assegurando que o trabalho da maneira como planejado pela gerência, conhecendo todos os requisitos reguladores e de negócio.
- A melhor pessoa (ou máquina) é designada para realizar cada caso, e os casos mais importantes são determinados primeiro. Usuários não perdem tempo escolhendo qual item para trabalhar, talvez adiando casos importantes, mas difíceis.

- Processamento paralelo, onde duas ou mais tarefas são realizadas concorrentemente, é muito mais fácil do que em *Workflow* tradicional, manual.

Com a melhor pessoa fazendo o trabalho mais importante, seguindo os procedimentos corretos, não apenas o negócio é conduzido mais efetivamente, mas também custos são reduzidos e o serviço prestado aos clientes é geralmente melhor [FISCHER 02].

## 3. Padrões

### 3.1. Padrões de Projeto (*Design Patterns*)

Um Padrão de Projeto sistematicamente nomeia, motiva e explica um projeto geral, que endereça um problema de projeto em sistemas orientados a objetos. Ele descreve o problema, a solução, quando aplicar a solução, e suas conseqüências. Ele também fornece dicas de implementação e exemplos. A solução é uma combinação geral de objetos e classes que solucionam o problema. A solução é adaptada e implementada para resolver o problema em um contexto particular [GAMMA 95].

Algumas definições de Padrões de Projeto: [COOPER 98]

- “Padrões de Projeto constituem um conjunto de regras descrevendo como executar certas tarefas no domínio do desenvolvimento de software” [Pree, 1994].
- “Um padrão endereça um problema de projeto recorrente que se origina em situações específicas de projeto e apresenta uma solução para ele” [Buschmann, *et. al.* 1996].
- “Padrões identificam e especificam abstrações que estão acima do nível de classes isoladas e instâncias, ou de componentes” [Gamma, et al., 1993].

Em geral, um padrão possui quatro elementos essenciais, que o identifica: [GAMMA 95]

1. O **nome do padrão** é o meio usado para descrever um problema de projeto, suas soluções, e conseqüências em uma palavra ou duas. Nomear um padrão imediatamente aumenta nosso vocabulário de projeto. Ele nos deixa projetar em um nível maior de abstração. Ter um vocabulário para padrões nos permite falar a respeito deles com nossos colegas, em nossa documentação, e até com nós mesmos. Ele torna fácil pensar sobre projetos e a transmiti-los a outros.
2. O **problema** descreve quando aplicar o padrão. Ele explica o problema e seu contexto. Ele deve descrever problemas de projeto específicos e estruturas de classes ou objetos que são característicos de um projeto

inflexível. Às vezes o problema irá incluir uma lista de condições que precisam ser conhecidas, para que tenha sentido, antes de se aplicar o padrão.

3. A **solução** descreve os elementos que compõem o projeto, seus relacionamentos, responsabilidades, e colaborações. A solução não descreve um projeto ou implementação concreta, porque um padrão é como um modelo que pode ser aplicado em muitas situações diferentes. Em vez disso, o padrão fornece uma descrição abstrata de um problema de projeto e como um arranjo geral de elementos o soluciona.
4. As **conseqüências** são os resultados e medições da aplicação do padrão. Embora as conseqüências sejam muitas vezes não ditas quando descrevemos decisões de projeto, elas são criteriosas para avaliar alternativas de projeto e para entender os custos e os benefícios da aplicação do padrão. As conseqüências para software muitas vezes interessam medições de espaço e tempo. Elas podem endereçar características de linguagem e implementação da mesma forma. Uma vez que reutilização é um fator em projeto orientado a objetos, as conseqüências da utilização de um padrão incluem seu impacto na flexibilidade, extensibilidade, ou portabilidade do sistema. Listar estas conseqüências explicitamente ajuda a entendê-las e avaliá-las.

### **3.2. Padrões de Workflow (Workflow Patterns)**

O trabalho de pesquisa de Wil van der Aalst, Arthur ter Hofstede, Bartek Kiepuszewski, e Alistair Barros resultou na identificação de 21 padrões que descrevem o comportamento de processos de negócio. A base lógica para o desenvolvimento dos padrões foi descrever as possíveis capacidades que um *Workflow* deve ter durante a execução de processos de negócios [WHITE 04].

Estes padrões, chamados Padrões de *Workflow* [AALST 02a], podem ser usados para examinar WFMS aos quais se pretende utilizar ou podem servir como um conjunto de idéias de como implementar requisitos de *Workflow* em um sistema próprio que já esteja rodando na empresa ou que se pretenda implementar.

Os padrões variam de muito simples a muito complexos e cobrem os comportamentos que podem ser capturados dentro da maioria dos modelos de processo de negócio [WHITE 04].

Eles estão divididos nas seguintes categorias:

- Padrões de Controle Básico: *Sequence, Parallel Split, Synchronization, Exclusive Choice* e *Simple Merge*.
- Padrões de Ramificação e Sincronização Avançados: *Multiple Choice, Synchronizing Merge, Multiple Merge, Discriminator* e *N-out-of-M-join*.
- Padrões Estruturais: *Arbitrary Cycles* e *Implicit Termination*.
- Padrões Envolvendo Múltiplas Instâncias: *MI without synchronization, MI with a priori known design time knowledge, MI with a priori known runtime knowledge* e *MI with no a priori runtime knowledge*.
- Padrões Baseados em Estados: *Deferred Coice, Interleaved Parallel Routing* e *Milestone*.
- Padrões de Cancelamento: *Cancel Activity* e *Cancel Case*.

Um Padrão de *Workflow* básico possui os seguintes elementos, além do nome:

1. Uma **descrição**, que explica o funcionamento do padrão.
2. Uma lista de **sinônimos**, que contém a lista de outros nomes pelos quais o padrão também é conhecido.
3. **Alguns exemplos**, que exemplifica situações, através da descrição de situações hipotéticas ou metáforas, para que se possa visualizar o que se quer atingir com o padrão.

Em alguns casos de padrões mais complexos, podem aparecer os seguintes elementos:

4. O **problema**, que descreve porque a construção é difícil de ser implementada em muitos WFMS disponíveis hoje em dia.
5. Possíveis estratégias de **implementação**, que também é referenciado como **solução**, e descreve como, assumindo um conjunto de primitivas, o procedimento requerido pode ser realizado.

A seguir são apresentados os padrões propostos, conforme [AALST 02a].

### 3.2.1. Padrões de Controle de Fluxo Básicos

Padrões de controle de fluxo básico são aqueles que cobrem aspectos elementares do controle de processos. Estes padrões estão muito próximos dos conceitos elementares de controle de fluxo fornecidos pela WfMC em [WfMC 99].

#### 3.2.1.1. Padrão 1 (Sequence)

**Descrição:** Uma atividade em um processo de *workflow* é habilitada depois da execução de outra atividade no mesmo processo.

**Sinônimos:** *Sequential routing, serial routing.*

**Implementação:**

- O padrão *sequence* é usado para modelar passos consecutivos em um processo de *workflow* e é diretamente suportado por cada um dos WFMS disponíveis. A implementação típica envolve a ligação entre duas atividades com uma flecha de controle de fluxo incondicional. A figura 4 ilustra a implementação deste padrão.



Figura 4: Implementação do padrão sequence

#### 3.2.1.2. Padrão 2 (Parallel Split)

**Descrição:** Um ponto no processo de *workflow* onde uma *thread* de controle única divide-se em múltiplas *threads* de controle, que podem ser executadas em paralelo, deste modo permitindo que as atividades sejam executadas simultaneamente ou em qualquer ordem.

**Sinônimos:** *AND-split, parallel routing, fork.*

**Implementação:**

- Todas as *engines* de *workflow* conhecidas possuem construções para a implementação deste padrão. Podem-se identificar duas abordagens básicas:

*AND-splits* explícitos e *AND-splits* implícitos. *Engines* de *workflow* que suportam o construtor *AND-split* definem um nodo de rota com mais de uma transição de saída, que será habilitada tão logo que o nodo de rota é habilitado. *Engines* de *workflow* que suportam *AND-splits* implícitos não fornecem construções de rota especiais – cada atividade pode ter mais que uma transição de saída, e cada transição possui condições associadas. Para realizar execução paralela, o projetista de *workflow* deve se certificar de que múltiplas condições associadas com transições de saída do nodo são avaliadas como *True*. A figura 5 ilustra a implementação deste padrão.

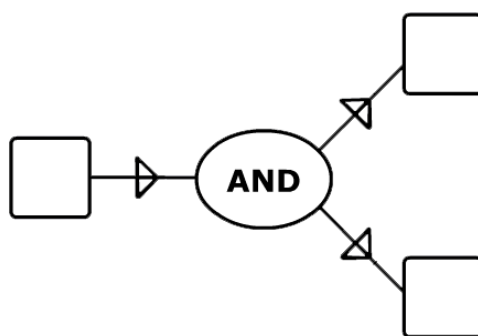


Figura 5: Implementação do padrão parallel split

### 3.2.1.3. Padrão 3 (Synchronization)

**Descrição:** Um ponto no processo de *workflow* onde sub-processos/atividades paralelos múltiplos convergem em uma *thread* simples de controle, deste modo sincronizando *threads* múltiplas. É uma suposição deste padrão que cada ramificação de entrada de um sincronizador é executada apenas uma vez.

**Sinônimos:** *AND-join*, *rendezvous*, *synchronizer*.

#### Implementação:

- Todas as *engines* de *workflow* disponíveis possuem construções de suporte para a implementação deste padrão. Similarmente ao padrão 2, podem-se identificar duas abordagens básicas: *AND-joins* explícitos e *joins* implícitos em uma atividade com mais de uma transição de entrada. A figura 6 ilustra a implementação deste padrão.



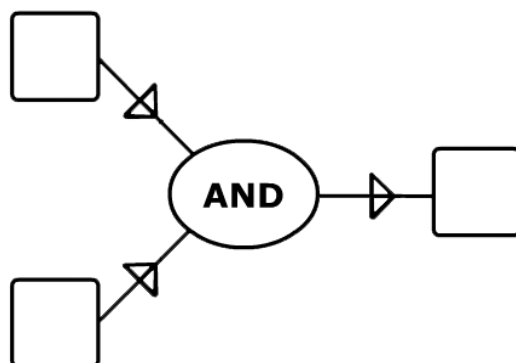


Figura 6: Implementação do padrão synchronization

#### 3.2.1.4. Padrão 4 (Exclusive Choice)

**Descrição:** Um ponto no processo de *workflow* onde, baseado em uma decisão ou dado de controle de *workflow*, uma ramificação é escolhida, em meio a algumas.

**Sinônimos:** *XOR-split*, *conditional routing*, *switch*, *decision*.

##### Implementação:

- Similarmente ao padrão 2, existe um número de estratégias básicas. Algumas *engines* de *workflow* fornecem uma construção explícita para a implementação do padrão *exclusive choice*. Em algumas *engines* de *workflow* o projetista de *workflow* tem que emular a exclusividade de escolha, especificando condições de transição exclusivas. Em outro produto de *workflow*, *Eastman*, uma lista de regras pós-processada pode ser especificada para uma atividade. Depois da conclusão da atividade, a transição é associada com a primeira regra nesta lista para avaliar se a condição *true* foi pega. A figura 7 ilustra a implementação deste padrão.

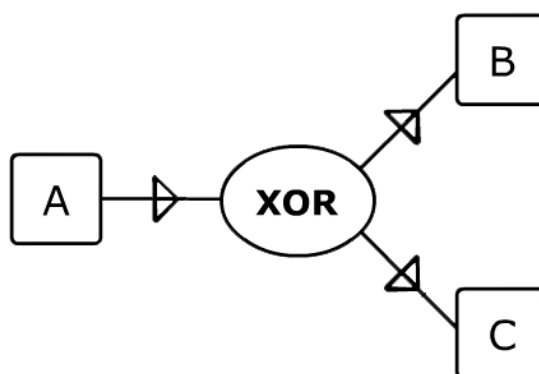


Figura 7: Implementação do padrão exclusive choice

### 3.2.1.5. Padrão 5 (Simple Merge)

**Descrição:** Um ponto no processo de *workflow* onde duas ou mais ramificações alternativas chegam juntas, sem sincronização. É uma suposição deste padrão que nenhuma das ramificações alternativas é executada em paralelo.

**Sinônimos:** *XOR-join*, *asynchronous join*, *merge*.

**Implementação:**

- Dado que nós estamos assumindo que execução paralela de *threads* alternativas não ocorre, esta é uma situação direta e todas as *engines* de *workflow* suportam uma construção que pode ser usada para implementar o *simple merge*. É interessante notar aqui que algumas linguagens impõem certo nível de estrutura para garantir automaticamente que não mais de uma *thread* alternativa está rodando em qualquer ponto em determinado momento. A figura 8 ilustra a implementação deste padrão.

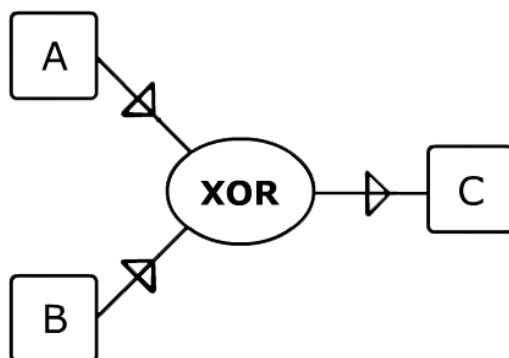


Figura 8: Implementação do padrão simple merge

### 3.2.2. Padrões de Ramificação e Sincronização Avançados

Nessa seção, o foco será nos padrões de ramificação e sincronização avançados. Ao contrário dos padrões encontrados na seção anterior, estes padrões não possuem suporte direto na maioria das *engines* de *workflow*. Contudo, eles são completamente comuns nos cenários de negócio.

### 3.2.2.1. Padrão 6 (Multi-choice)

**Descrição:** Um ponto no processo de *workflow* onde, baseado em uma decisão ou dado de controle de *workflow*, um número de ramificações são escolhidos.

**Sinônimos:** *Conditional routing, selection, OR-split.*

**Problema:** Em muitos WFMS, podem-se especificar condições nas transações. Nestes sistemas, o padrão *multi-choice* pode ser implementado diretamente. Entretanto, Existem WFMS que não oferecem a possibilidade de especificar condições nas transações e que oferecem somente blocos de construção *AND-split* e *XOR-split* puros.

**Implementação:**

- Como dito, para linguagens de *workflow* que especificam condições de transição para cada transição (por exemplo, *Verve, MQSeries/Workflow, Forté Conductor*) a implementação do padrão *multi-choice* é direta. O projetista de *workflow* simplesmente especifica as condições desejadas para cada transição. Deve-se notar que o padrão *multi-choice* generaliza os padrões *parallel split* (Padrão 2) e *exclusive choice* (Padrão 4).
- Para linguagens que apenas fornecem construtores para implementar os padrões *parallel split* e *exclusive choice*, a implementação do padrão *multi-choice* deve ser realizada através de uma combinação dos dois. Cada possível ramificação é processada por um *XOR-split* que decide, baseado em dados de controle, entre ativar a ramificação ou ignorá-la. Todos *XOR-splits* são ativados por um *AND-split*.
- Uma solução similar para a anterior é obtida pela reversão da ordem dos padrões *parallel split* e *exclusive choice*. Para cada conjunto de ramificações que podem ser ativadas em paralelo, um *AND-split* é adicionado. Todos os *AND-splits* são precedidos por um *XOR-split*, que ativa o *AND-split* apropriado. Note que, tipicamente, nem todas as combinações de ramificações são possíveis. Por esta razão, esta solução pode conduzir a uma especificação de *workflow* mais compacta.

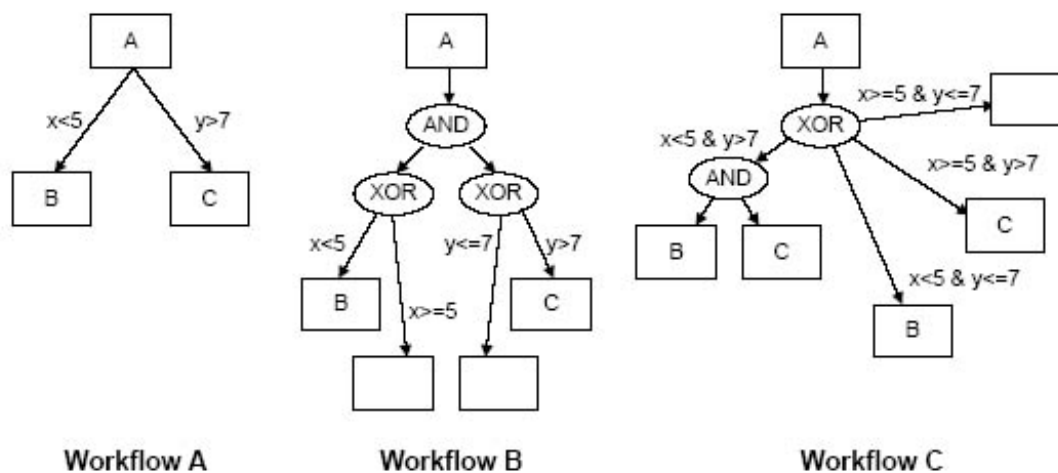


Figura 9: Padrões de projeto para o padrão multi-choice

### 3.2.2.2. Padrão 7 (Synchronizing Merge)

**Descrição:** Um ponto no processo de *workflow* onde caminhos múltiplos convergem em uma *thread* única. Se somente um caminho é tomado, é necessário que haja sincronização das *threads* ativas. Se apenas um caminho é tomado, as ramificações alternativas devem re-convergir sem sincronização. É uma suposição deste padrão que uma ramificação que foi realmente ativada não pode ser ativada de novo enquanto a fusão ainda está esperando pelas outras ramificações para se completar.

**Sinônimos:** *Synchronizing join*.

**Problema:** A principal dificuldade com este padrão é decidir quando sincronizar e quando fundir. Falando de maneira geral, este tipo de fusão precisa ter alguma capacidade para ser apto a determinar se pode (continuar a) esperar ativação por alguma de suas ramificações.

**Implementação:**

- As duas *engines* de *workflow* conhecidas dos autores dos padrões de *workflow* que fornecem uma construção direta para a realização deste padrão são *MQSeries/Workflow* e *InConcert*. Como observado anteriormente, se um *synchronizing merge* suceder um *OR-split* e mais de uma transição de saída daquele *OR-split* pode ser disparada, ela não o será até o tempo de execução em que podemos dizer qual sincronização pode ser feita ou não. *MQSeries/Workflow* trabalha em torno do problema de passar um *token False* para cada transição que avalia para falso e um *token True* para cada transição que avalia para verdadeiro. A fusão irá esperar até que ela receba *tokens* para

cada transição de entrada. *InConcert* não usa um conceito de *token False*. Ao invés disso, ele passa um *token* através de todas as transições em um gráfico. Este *token* pode ou não habilitar a execução de uma atividade dependendo da condição de entrada. Desta maneira, toda atividade tendo mais do que uma transição de entrada pode esperar que irá receber um *token* de cada uma delas, desta maneira *deadlock* não pode acontecer. O leitor cuidadoso pode notar que estas estratégias de avaliação necessitam que o processo de *workflow* não contenha ciclos.

- Em *Eastman*, itens de trabalho não paralelos dirigidos a um *Join* deixam de lado o processamento de junções, conseqüentemente um *XOR-split* seguido de um *AND-join* não precisa levar a um *deadlock*.
- Em outras *engines* de *workflow* a implementação de *synchronizing merge* tipicamente não é direta. A única solução é evitar o uso explícito do *OR-split* que pode disparar mais de uma transição de saída e implementá-la como uma combinação de *AND-splits* e *XOR-splits* (veja o padrão 6 – *Multi Choice*). Desta maneira podemos facilmente sincronizar ramificações correspondentes usando *AND-joins* e construções de fusão padrão.

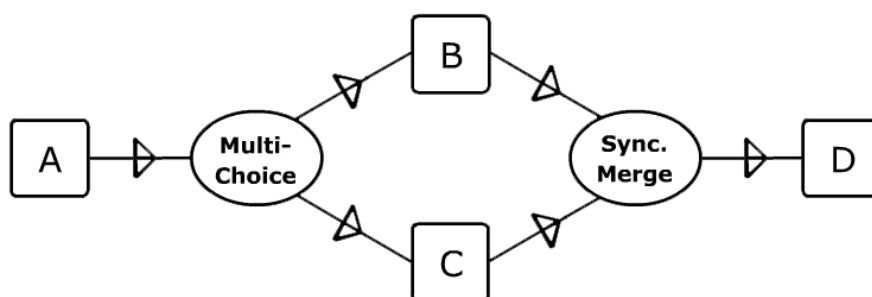


Figura 10: Implementação do padrão synchronizing merge

### 3.2.2.3. Padrão 8 (Multi-merge)

**Descrição:** Um ponto em um processo de *workflow* onde duas ou mais ramificações re-convergem sem sincronização. Se mais de uma ramificação é ativada, possivelmente concorrentemente, a atividade que segue a fusão é iniciada para toda ativação de toda ramificação de entrada.

**Problema:** O uso de uma construção padrão para fusão, como fornecida por alguns produtos de *workflow* para implementar este padrão muitas vezes conduz a resultados

indesejados. Alguns produtos de *workflow* (por exemplo, *Staffware*, *I-Flow*) não irão gerar uma segunda instância de uma atividade se outra instância continua executando. Finalmente, em alguns produtos de *workflow* (por exemplo, *Visual WorkFlo*, *SAP R/3 Workflow*) nem sempre é possível usar uma construção de fusão em associação com um *parallel split*.

### Implementação:

- As construções de *Eastman*, *Verve Workflow* e *Forté Conductor* podem ser usadas diretamente para implementar este padrão.
- Se o *multi-merge* não é parte de um laço, o padrão de projeto comum para linguagens que não são aptas a criar mais de uma instância ativa de uma atividade é replicar esta atividade no modelo de *workflow*. Se o *multi-merge* é parte de um laço, então tipicamente o número de instâncias de uma atividade que segue o *multi-merge* não é conhecido durante a etapa de projeto. Para uma solução típica a este problema, devem-se ver os padrões 14 (*Multiple Instances with a Priori Runtime Knowledge*) e 15 (*Multiple Instances Without a Priori Runtime Knowledge*).
- Uma solução interessante é oferecida pelo sistema *FLOWer*. *FLOWer* dá suporte a sub-planos dinâmicos. Um sub-plano dinâmico é uma sub-processo com um número variável de instâncias. Além disso, o número de instâncias pode ser controlado dinamicamente através de uma variável. Desta maneira é possível modelar de maneira indireta o *multi-merge*.

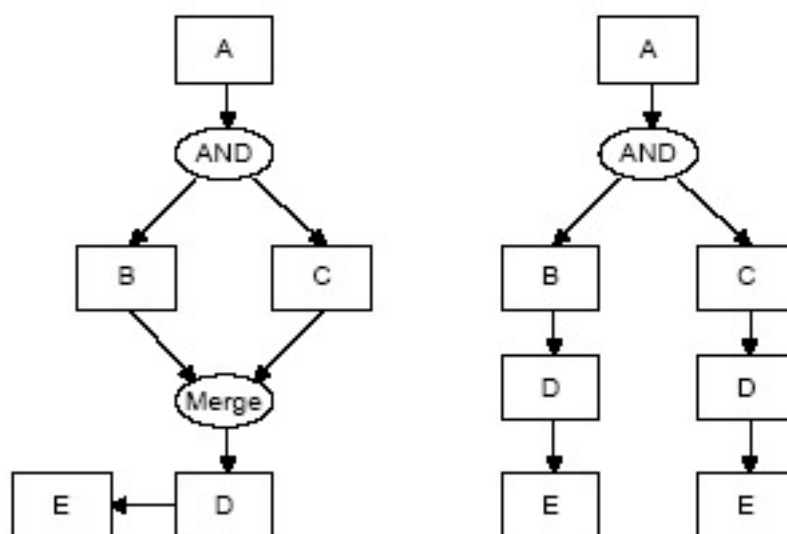


Figura 11: Implementação típica do padrão multi-merge

### 3.2.2.4. Padrão 9 (Discriminator)

**Descrição:** Um ponto em um processo de *workflow* que espera que uma das ramificações de entrada se complete antes de ativar a atividade subsequente. Daquele momento em diante ele espera por todas as ramificações que faltam se completar e as “ignora”. Uma vez que todas as ramificações de entrada foram disparadas, ele reinicializa a si mesmo, podendo assim ser disparado novamente (o que é importante, pois de outra maneira ele não poderia ser realmente usado no contexto de um laço).

**Problema:** A maioria das *engines* de *workflow* não possui uma construção que pode ser usada para uma implementação direta do padrão *discriminator*. Como mencionado no padrão 8 (*multi-merge*), a construção padrão para fusão em algumas *engines* de *workflow* (por exemplo, *Staffware*, *I-Flow*) não irá gerar a segunda instância de uma atividade se a primeira instância continua ativa. Isto não fornece uma solução para o *discriminator*, porém, desde que a primeira instância da atividade termine antes que uma tentativa é feita para iniciar de novo, uma segunda instância será criada.

**Implementação:**

- Existe uma construção especial que implementa a semântica do *discriminator* em *Verve*. Esta construção possui muitas ramificações de entrada e uma ramificação de saída. Quando uma das ramificações de entrada é finalizada, a atividade subsequente é disparada e o *discriminator* modifica sua condição de “preparado” para “esperando”. Daí em diante ele espera por todas as ramificações de entrada restantes completarem. Quando isto acontecer, ele modifica sua condição novamente para “pronto”. Esta construção fornece uma opção de implementação direta para o padrão *discriminator*, entretanto, ela não funciona propriamente quando usada no contexto de um laço (uma vez esperando pelas ramificações de entrada para completar, ele ignora disparos adicionais da ramificação que o disparou).
- No *SAP R/3 Workflow*, para *forks* (uma combinação de um *AND-split* e um *AND-join*) é possível especificar o número ramificações que precisam ser completadas para que o *fork* seja considerado completado. Fixando este número para um, realiza-se o *discriminator* exceto que 1) as ramificações que não foram

completadas recebe, a condição “logicamente apagada” e 2) o *fork* restringe a forma com que paralelismo/sincronização podem ser feitas.

- A semântica do *discriminator* pode ser implementada em produtos que suportam gatilhos personalizados. Por exemplo, no *Forté Conductor* um gatilho personalizado pode ser definido para uma atividade que possui mais de uma transição de entrada. Gatilhos personalizados definem a condição, tipicamente usando alguma linguagem de script interna, que quando satisfeita pode levar à execução de certa atividade. Assim um script pode ser usado para conseguir uma semântica parecida daquela do *discriminator*. O aspecto negativo desta abordagem é que a semântica de uma junção que usa gatilhos personalizados é impossível de ser determinada sem um exame cuidadoso dos scripts de gatilho subjacentes. Como tal, o uso de gatilhos personalizados pode resultar em modelos que são menos adequados e difíceis de entender.
- Alguns WFMS (por exemplo, *FLOWer*) permitem execução dependente de dados. No *FLOWer* é possível ter um marco que espera por uma variável para ser definida. No momento que a variável é definida, o processamento da *thread* paralela contendo o marco irá continuar. A funcionalidade de reinicialização é realizada através do uso de sub-planos sequenciais/dinâmicos antes do que iteração.
- Tipicamente, em outras *engines* de *workflow* o *discriminator* é impossível de ser implementado diretamente na linguagem de modelagem de *workflow* fornecida.

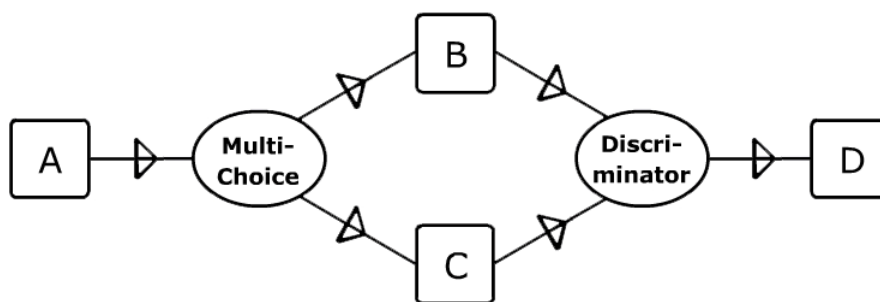


Figura 12: Implementação do padrão discriminator

### 3.2.2.5. Padrão 9a (N-out-of-M-join)

**Descrição:** Este padrão é uma generalização do padrão *discriminator*. *N-out-of-M-join* é um ponto em um processo de *workflow* onde *M* caminhos paralelos convergem em um.



A atividade subsequente deve ser ativada quando  $N$  caminhos forem completados. A conclusão de todos os outros caminhos deve ser ignorada. Similarmente ao *discriminator*, uma vez que todas as ramificações forem disparadas, o join reinicializa a si mesmo, para poder disparar novamente.

### 3.2.3. Padrões Estruturais

WFMS diferentes impõem restrições diferentes em seus modelos de *workflow*. Estas restrições nem sempre são naturais no ponto de vista da modelagem e tendem a restringir a liberdade de especificação do analista de negócios. Como resultado, analistas de negócio ou têm que se sujeitar às restrições da linguagem de *workflow* desde o início, ou modelam seus problemas livremente e transformam as especificações resultantes, posteriormente. Uma questão real é aquela que trata da adequação. Em muitos casos, os *workflows* resultantes podem ser desnecessariamente complexo, o que impacta no usuário final, que pode querer monitorar o progresso de seus *workflows*. Nessa seção são apresentados dois padrões, que ilustram restrições típicas impostas em especificações de *workflow* e suas conseqüências.

#### 3.2.3.1. Padrão 10 (Arbitrary Cycles)

**Descrição:** Um ponto no processo de *workflow* onde um ou mais atividades podem ser realizadas repetidamente.

**Sinônimos:** *Loop, iteration, cycle.*

**Problema:** Algumas das *engines* de *workflow* não permitem ciclos arbitrários – elas possuem suporte para ciclos estruturados apenas, ou através da construção *decomposition* (*MQSeries/Workflow, InConcert, FLOWer*) ou através de um construtor de laço especial (*Visual Workflo, SAP R/3 Workflow*).

**Implementação:**

- Ciclos arbitrários podem tipicamente ser convertidos em ciclos estruturados a menos que eles contenham um dos mais avançados padrões como os de múltiplas instâncias (veja o padrão 14, *Multiple Instances with a Priori Runtime Knowledge*). A conversão é feita através de variáveis auxiliares e/ou repetição de

nodo. A figura 6 fornece um exemplo de um *workflow* arbitrário convertido para um *workflow* estruturado.

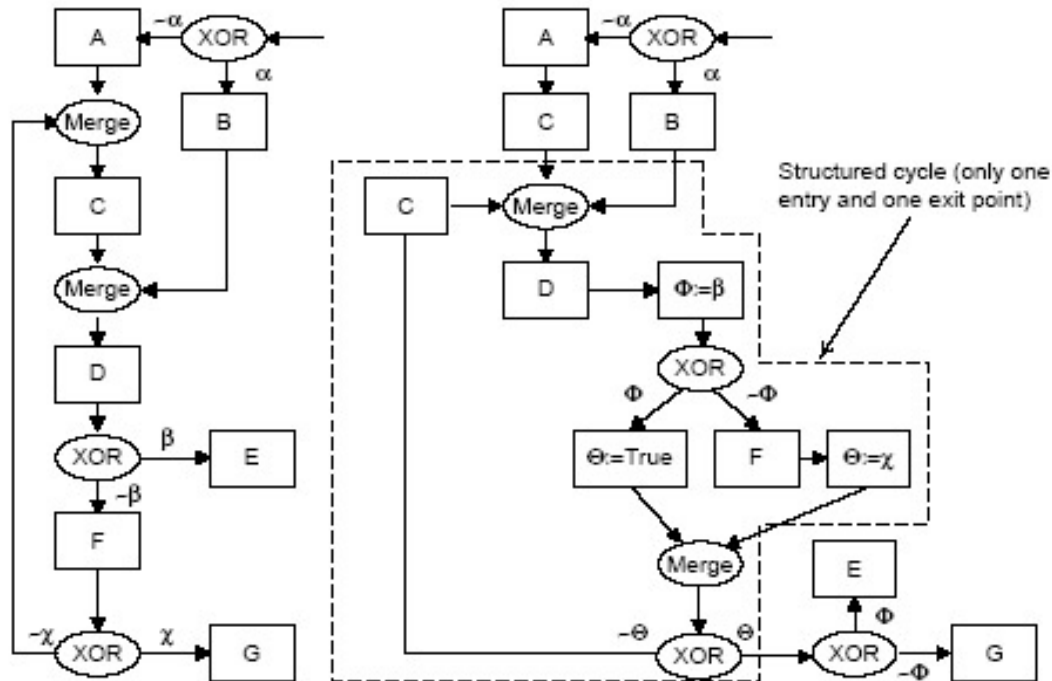


Figura 13: Exemplo de implementação de arbitrary cycles

### 3.2.3.2. Padrão 11 (Implicit Termination)

**Descrição:** Um sub-processo dado deve ser terminado quando não há mais nada a ser feito. Em outras palavras, não existem atividades ativas no *workflow* e nenhuma outra atividade pode ser feita ativa (e ao mesmo tempo em que o *workflow* não está em *deadlock*).

**Problema:** A maior parte das *engines* de *workflow* termina o processo quando um nodo *Final* explícito é alcançado. Quaisquer atividades atuais que estão rodando nesta hora serão abortadas.

**Implementação:**

- Algumas *engines* de *workflow* (*Eastmanm Staffwarem MQSeries/Workflow*, *InCOncert*) suportam este padrão diretamente, de modo que elas terminariam um (sub-)processo quando não existe mais nada a ser feito.
- Para produtos de *workflow* que não suportam este padrão diretamente, a solução típica a este problema é transformar o modelo em modelo equivalente que

possui apenas um nodo de finalização. A complexidade para aquela tarefa depende muito do modelo atual. Às vezes é fácil e direto, tipicamente pelo uso de uma combinação de construções de junção e repetição de atividade. Entretanto, existem situações onde é difícil e até impossível fazer isto. Um modelo que envolve múltiplas instâncias e terminação implícita é tipicamente muito difícil de converter em um modelo com terminação explícita.

### 3.2.4. Padrões envolvendo Múltiplas Instâncias

Os padrões desta seção envolvem características que se referem a múltiplas instâncias. Do ponto de vista teórico, o conceito é relativamente simples, e corresponde a muitas *threads* de execução, fazendo referência a uma definição compartilhada. Do ponto de vista prático, significa que uma atividade em um diagrama de *workflow* pode possuir mais do que uma instância ativa, ao mesmo tempo. O problema fundamental com a implementação destes padrões é que devido às restrições de projeto e necessidade de antecipação para esses requisitos, a maior parte das *engines* de *workflow* não permite que mais de uma instância de uma mesma atividade esteja ativa ao mesmo tempo.

#### 3.2.4.1. Padrão 12 (Multiple Instances Without Synchronization)

**Descrição:** Dentro do contexto de um caso único (por exemplo, instância de *workflow*) múltiplas instâncias de uma atividade podem ser criadas, como por exemplo, se existisse uma facilidade para gerar novas *threads* de controle. Cada uma dessas *threads* de controle é independente de outras *threads*. Além disso, não há necessidade de sincronizar estas *threads*.

**Sinônimos:** *Multi threading without synchronization, Spawn off facility.*

**Implementação:**

- A implementação mais direta para este padrão é através do uso de laço e da construção *parallel split* por tanto tempo quanto a *engine* de *workflow* suporte o uso de divisões paralelas sem junções correspondentes e permite o disparo de atividades que estão realmente ativas. Isto é possível em linguagens como *Forté* e *Verve*. Esta solução é ilustrada pelo *Workflow A* na figura 14.

- Algumas linguagens de *workflow* suportam uma construção extra que possibilita ao projetista criar um sub-processo ou um sub-fluxo que será gerado do processo principal e será executado concorrentemente. Por exemplo, *Visual Workflow* suporta a construção *Release* enquanto *I-Flow* suporta o *Chained Process Node*. *COSA* possui uma facilidade similar, onde um *workflow* pode conter múltiplos fluxos concorrentes que são criados através de uma API e informações compartilhadas.
- Na maior parte dos WFMS existe a possibilidade de criar novas instâncias de um processo de *workflow* através de alguma API. Isto permite a criação de novas instâncias, chamando-se o método apropriado das atividades dentro do fluxo principal. Note que este mecanismo funciona. Entretanto, o sistema não mantém relações entre o fluxo principal e as instâncias que foram geradas.

#### **3.2.4.2. Padrão 13 (Multiple Instances With a Priori Design Time Knowledge)**

**Descrição:** Para uma instância de processo, uma atividade é habilitada múltiplas vezes. O número de instâncias de uma dada atividade para uma dada instância de processo é conhecido no tempo de projeto. Uma vez que todas as instâncias são completadas, alguma outra atividade precisa ser iniciada.

**Implementação:**

- Se o número de instâncias é conhecido a priori durante o tempo de projeto, então uma opção de implementação muito simples é replicar a atividade no modelo de *workflow*, precedendo-o com uma construção usada para a implementação do padrão *parallel split*. Uma vez que todas as atividades são completadas, é simples sincronizá-las usando uma construção padrão de sincronização.

#### **3.2.4.3. Padrão 14 (Multiple Instances With a Priori Runtime Knowledge)**

**Descrição:** Para um caso uma atividade é habilitada múltiplas vezes. O número de instâncias de uma dada atividade para um dado caso varia e pode depender de características do caso ou disponibilidade de recursos, mas é conhecido em algum estágio durante tempo de execução, antes das instâncias daquela atividade ter de ser

criadas. Uma vez que todas as instâncias são completadas alguma outra atividade necessita ser iniciada.

**Problema:** Ao passo que número de instâncias de uma dada atividade não é conhecido durante o projeto, nós não podemos simplesmente replicar esta atividade em um modelo de *workflow* durante o estágio de projeto. Atualmente, apenas alguns poucos WFMS permitem múltiplas instâncias de uma única atividade em um dado tempo, ou oferecem uma construção especial para a ativação múltipla de uma atividade para uma dada instância de processo, de tal maneira que estas instâncias são sincronizadas.

**Implementação:**

- Se a *engine* de *workflow* suporta instâncias múltiplas diretamente (por exemplo, *Forté* e *Verve*), nós podemos tentar e usar a solução ilustrada no *Workflow A* na figura 14. Entretanto, a atividade *E* neste modelo possivelmente será iniciada antes que todas as instâncias da atividade *B* forem completadas. Para realizar sincronização apropriada, há a necessidade de recorrer a técnicas muito além do poder de modelagem dessas linguagens. Por exemplo, pode ser possível implementar a atividade *B* de tal maneira que, uma vez completada, ela envia um evento a alguma fila de eventos externa. A atividade *E* pode ser precedida por outra atividade que consome os eventos da fila e dispara *E* apenas se o número de eventos na fila é igual ao número de instâncias da atividade *B* (como pré-determinado pela atividade *A*). Esta solução é muito complexa, pode ter alguns problemas de concorrência, e para o usuário final é totalmente obscuro o que é a real semântica do processo.
- Problemas similares acontecem quando se usa a construção *Release* do *Visual Workflow*, o *Chained Process Node* do *I-Flow*, os sub-fluxos múltiplos do *COSA*, ou alguma API para invocar o sub-processo como parte de uma atividade em um processo. Em cada um desses sistemas, é muito difícil sincronizar sub-processos concorrentes.
- Algumas *engines* de *workflow* oferecem uma construção que pode ser usada para instanciar um dado número de instâncias de uma atividade.
- Se existe um número máximo de possíveis instâncias, então uma combinação de *AND-splits* e *XOR-splits* pode ser usada para obter o caminho desejado. Um *XOR-split* é usado para selecionar o número de instâncias e dispara um dos diversos *AND-splits*. Para cada número de instâncias possíveis, existe um *AND-split* com a cardinalidade correspondente. A desvantagem dessa solução é que o

modelo de workflow resultante pode se tornar grande e complexo, e o número máximo de possíveis instâncias precisam ser conhecidas mais adiante (veja o *Workflow C*, na figura 14).

- Em muitos casos, o comportamento de rota desejado pode ser suportado muito facilmente, fazendo-o mais seqüencial. Simplesmente use iterações (por exemplo, o padrão 10 – *Arbitrary Cycles*) para ativar instâncias da atividade seqüencialmente. Suponha que a atividade *A* é seguida por  $n$  instanciações de *B*, seguidas por *E*. Em primeiro lugar execute *A*, então execute a primeira instanciação de *B*. Cada instanciação de *B* é seguida por um *XOR-split* para determinar se outra instanciação de *B* é necessária ou que *E* é o próximo passo a ser executado. Esta solução é razoavelmente direta. Entretanto, as  $n$  instanciações de *B* não são executadas em paralelo, mas em uma ordem fixa (veja o *Workflow B*, na figura 14). Em muitas situações isto não é aceitável.
- É apresentado um padrão que é tipicamente o mais difícil de implementar. Nele, o número de instâncias em um processo é determinado em uma solução de execução de maneira dinâmica, como por exemplo, o uso inapropriado do conceito *Bundle*.

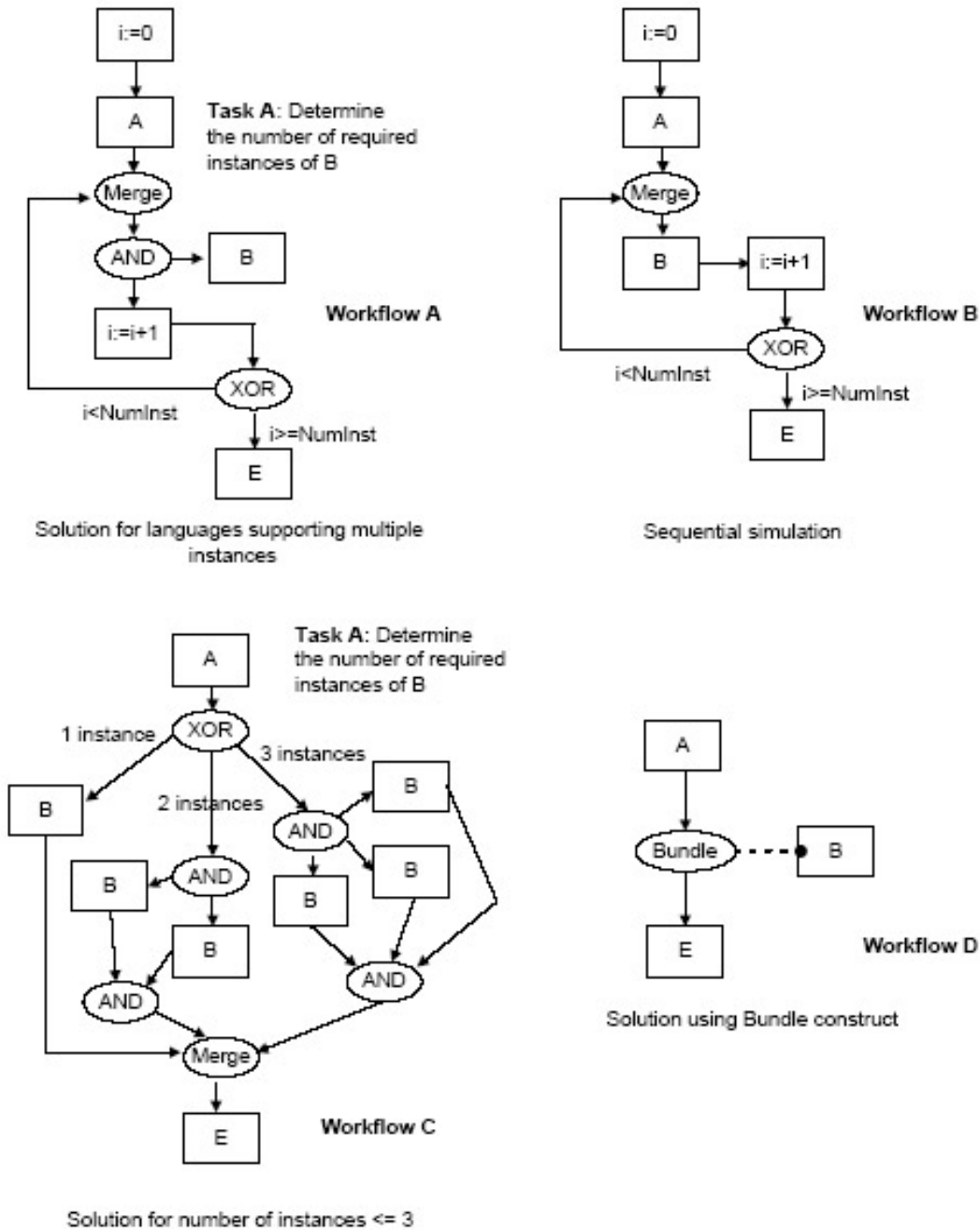


Figura 14: Padrões de projeto para múltiplas instâncias

#### 3.2.4.4. Padrão 15 (Multiple Instances Without a Priori Runtime Knowledge)

**Descrição:** Para um caso uma atividade é habilitada múltiplas vezes. O número de instâncias de uma dada atividade para um dado caso não é conhecido durante a etapa de projeto, nem é conhecido durante qualquer estágio durante o tempo de execução, antes que as instâncias daquela atividade sejam criadas. Uma vez que todas as instâncias são

completadas alguma outra atividade precisa ser iniciada. A diferença com relação ao padrão 14 é que enquanto algumas das instâncias estão sendo executadas ou foram realmente completadas, novas instâncias podem ser criadas.

**Problema:** Algumas *engines* de *workflow* fornecem suporte para a geração de múltiplas instâncias apenas se o número de instâncias é conhecido em algum estágio do processo. Isto pode ser comparado a um laço *for* em linguagens procedurais. Entretanto, estas construções não ajudam em processos que requerem a funcionalidade de laço *while*.

**Implementação:**

- *FLOWer* é um dos poucos sistemas que suportam diretamente este padrão. Em *FLOWer* é possível ter sub-planos dinâmicos. O número de instâncias de cada sub-plano pode ser alterado a qualquer momento (a não ser que especificado de outra maneira).
- Este padrão é uma generalização do padrão 14 (*Multiple Instances With a Priori Runtime Knowledge*). Algumas estratégias de implementação também são aplicáveis aqui. Especificamente, a parte de criação deste padrão pode ser facilmente implementada se a *engine* suporta diretamente múltiplas instâncias. Similarmente, nós podemos também fornecer uma implementação usando construtores especiais para gerar novos processos ou usando APIs para fazer isto. Entretanto, assim como com o padrão 14, sincronização das instâncias é muito difícil de ser feita, pelo fato de que neste padrão é mais difícil ainda, pois não existe uma contagem de atividades geradas prontamente disponíveis. Considere por exemplo o *Workflow A* na figura 15. Uma vez que *NumInst* pode variar enquanto instâncias de *B* são executadas, a implementação desta construção é mais complicada. Dinamicamente, o número de instâncias (a serem) ativadas necessita ser comparado com o número de instâncias completadas. Isto pode ser implementado como uma pré-condição e uma fila de eventos conectada a *E* que conta o número de instâncias completadas de *B*, por exemplo, cada instância de *B* gera um evento para *E* quanto se completa. A atividade *E* tem uma pré-condição comparando o número de instâncias lançadas e o número de instâncias completadas.
- Se a linguagem suporta múltiplas instâncias e um conceito de decomposição com terminação implícita (por esta razão uma decomposição só é considerada finalizada quando todas as suas atividades são finalizadas), então múltiplas instâncias podem ser sincronizadas colocando-se o sub-fluxo do *workflow*



contendo o laço que gera as múltiplas instâncias dentro do bloco de decomposição (veja o *Workflow B* na figura 15). Aqui, a atividade *B* será invocada muitas vezes, e a atividade *C* é usada para determinar se mais instâncias de *B* são necessárias. Uma vez que todas as instâncias de *B* forem completadas, o sub-processo irá se completar e a atividade *E* pode ser processada. Terminação implícita do sub-processo é usada como mecanismo de sincronização para as múltiplas instâncias da atividade *B*. Nós achamos que este caminho pode ser uma solução bastante natural para o problema, entretanto, nenhuma das linguagens incluídas no review dos autores suporta ambos, múltiplas instâncias e um conceito de decomposição com terminação implícita.

- Similarmente ao padrão 14, o comportamento de rota desejado pode ser suportado muito facilmente fazendo-o seqüencial.

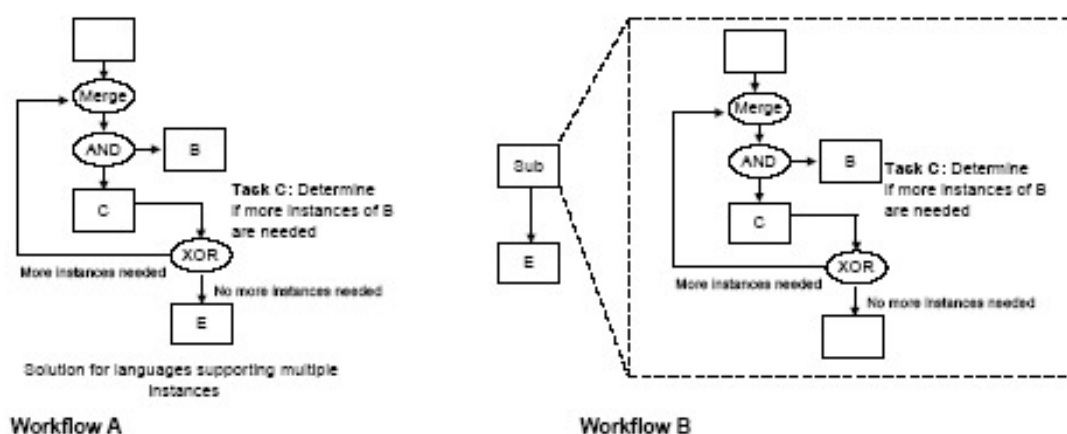


Figura 15: Padrões de projeto para múltiplas instâncias

### 3.2.5. Padrões Baseados em Estado

Em *workflows* reais, a maioria das instâncias de workflow está em um estado de espera de processamento antes de serem processadas. Muitos cientistas da computação, entretanto, parecem ter um pensamento, tipicamente derivado da programação, onde a noção de estado é interpretada de uma forma limitada e é essencialmente reduzida ao conceito de dado. Nesta seção serão tratadas as reais diferenças entre processo e

computação de trabalho, e os cenários de negócio onde uma notação explícita de estado é requerida.

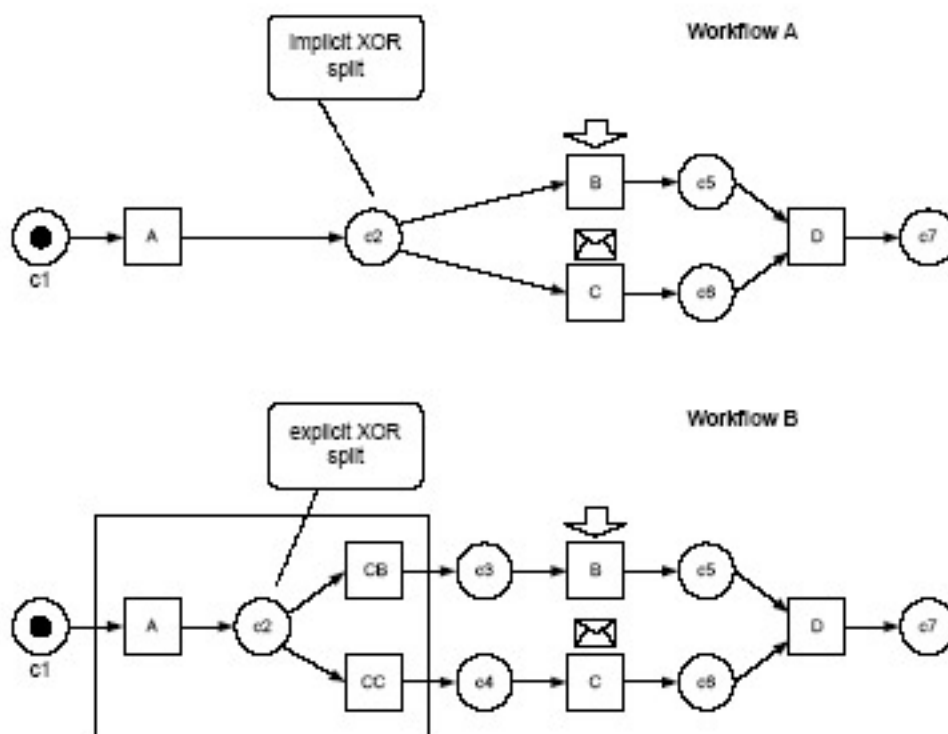


Figura 16: Ilustrando a diferença entre *XOR-splits* implícitos (*Workflow A*) e explícitos (*Workflow B*)

### 3.2.5.1. Padrão 16 (Deferred Choice)

**Descrição:** Um ponto no processo de *workflow* onde uma de várias ramificações é escolhida. Em contraste com o *XOR-split*, a escolha não é feita explicitamente (por exemplo, baseada em dados ou em uma decisão), mas várias alternativas são oferecidas para o ambiente. Entretanto, em contraste com o *AND-split*, apenas uma das alternativas é executada. Isso significa que uma vez que o ambiente ativa uma das ramificações, as outras ramificações alternativas são removidas. É importante notar que a escolha é retardada até que o processamento em uma das ramificações alternativas é iniciada, ou seja, o momento de escolha é tão tardio quanto possível.

**Sinônimos:** External choice, implicit choice, deferred XOR-split.

**Problema:** Muitos WFMS suportam o *XOR-split* descrito no padrão 4, mas não suportam *deferred choice*. Uma vez que ambos os tipos de escolha são desejáveis, a ausência de *deferred choice* é um problema real.

### Implementação:

- *COSA* é um dos poucos sistemas que suportam diretamente o padrão *deferred choice*. Uma vez que *COSA* é baseado em redes de Petri, é possível modelar escolhas implícitas, como indicado na figura 16 (A). Alguns sistemas oferecem suporte parcial para este padrão, fornecendo construções especiais para uma *deferred choice* entre uma ação de usuário e um *time out* (por exemplo, *Staffware*) ou duas ações de usuário (por exemplo, *FLOWer*).
- Assumir que a linguagem de *workflow* sendo usada suporta o cancelamento de atividades através de uma transição especial (por exemplo, *Staffware*, veja padrão 19 – *Cancel Activity*) ou através de uma API (maioria das outras *engines*). Cancelamento de uma atividade significa que a atividade está sendo removida da lista de trabalho designada, contanto que ela não tenha sido iniciada ainda. A escolha tardia pode ser realizada habilitando-se todas as alternativas via um *AND-split*. Uma vez que o processamento de uma das alternativas é iniciado, todas as outras alternativas são canceladas. Considere a escolha tardia entre *B* e *C* na figura 16 (*Workflow A*). Depois de *A*, ambos *B* e *C* são habilitados. Uma vez que *B* é selecionado/executado, a atividade *C* é cancelada. Uma vez que *C* é selecionado/executado, a atividade *B* é cancelada. O *Workflow A* da figura 17 mostra o modelo de *workflow* correspondente. Note que a solução nem sempre funciona, porque *B* e *C* podem ser selecionados /executados concorrentemente.
- Outra solução para o problema é substituir a *referred choice* por um *XOR-split* explícito, ou seja, uma atividade adicional é incluída. Todos os gatilhos que ativam as ramificações alternativas são redirecionados para a atividade adicionada. Supondo que a atividade consegue distinguir entre gatilhos, ele pode ativar a ramificação apropriada. Considere o exemplo mostrado na figura 16. Introduzindo-se uma nova atividade *E* depois de *A* e redirecionando gatilhos de *B* e *C* para *E*, o *XOR-split* implícito pode ser substituído por um *XOR-split* explícito baseado na origem do primeiro gatilho. O *Workflow B* da figura 17 mostra o modelo de *workflow* correspondente. Note que a solução move parte do roteamento para a aplicação ou nível da tarefa. Além disso, esta solução supõe que a escolha é feita baseada no tipo do gatilho.

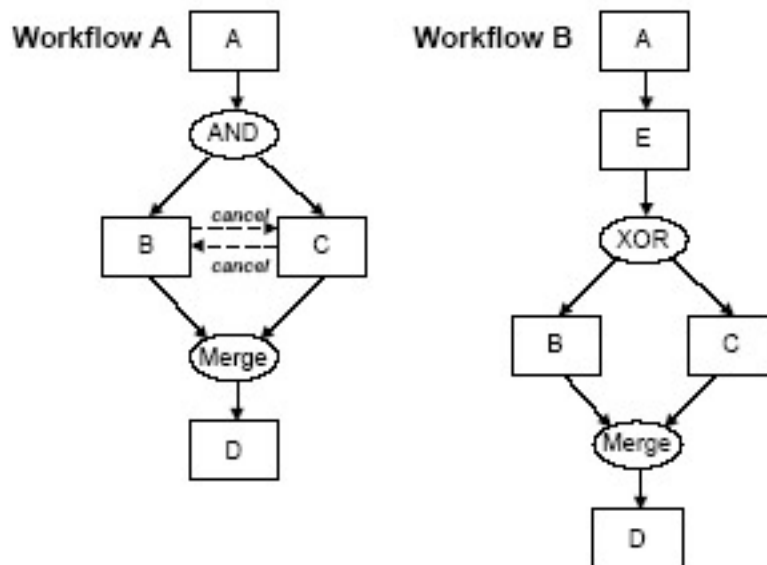


Figura 17: Estratégias para implementação de deferred choice

### 3.2.5.2. Padrão 17 (Interleaved Parallel Routing)

**Descrição:** Um conjunto de atividades é executado em uma ordem arbitrária: Cada atividade no conjunto executada, a ordem é decidida em tempo de execução, e duas atividades não são executadas ao mesmo momento (isto é, duas atividades não são ativas para a mesma instância de *workflow* ao mesmo tempo).

**Sinônimos:** *Unordered sequence*.

**Problema:** Uma vez que a maior parte dos WFMS suporta concorrência quando usam construções como *AND-split* e *AND-join*, não é possível especificar roteamento paralelo intercalado.

**Implementação:**

- Uma solução muito simples, mas insatisfatória, é fixar a ordem de execução, isto é, ao invés de usar roteamento paralelo, é usado roteamento seqüencial. Desde que as atividades possam ser executadas em uma ordem arbitrária, uma solução usando ordem pré-definida pode ser aceitável. Entretanto, fixando-se a ordem, a flexibilidade é reduzida e os recursos não podem ser utilizados em seu máximo potencial.
- Outra solução é usar uma combinação de construções de implementação para os padrões *sequence* e *exclusive choice*, isto é, várias seqüências alternativas são definidas e antes da execução uma seqüência é selecionada usando um *XOR*-

*split*. Uma desvantagem é que a ordem é fiada antes que a execução inicie, e não fica claro como a escolha é feita. Além disso, o modelo de *workflow* pode se tornar muito complexo e grande, enumerando-se todas as possíveis seqüências. O *Workflow B* na figura 18 ilustra esta solução em um caso com atividades em árvore.

- Utilizando-se as estratégias de implementação para o padrão *deferred choice* (ao invés de um *XOR-split* explícito) a ordem não necessita ser fixada antes que a execução se inicie, isto é, o *XOR-split* implícito permite a seleção da ordem *on-the-fly*. Infelizmente, o modelo resultante tipicamente possui uma estrutura confusa, *spaghetti-like*. Esta solução é ilustrada pelo *Workflow C* da figura 18.
- Para modelos de *workflow* baseados em redes de Petri, a intercalação de atividades pode ser forçada adicionando-se um local que é ao mesmo tempo de entrada e saída de todas as atividades concorrentes potenciais. O *AND-split* adiciona um *token* a este local e o *AND-join* remove o *token*. É fácil ver que tal local realiza a “exclusão mútua” necessária. Veja a figura 19 para um exemplo onde esta construção é aplicada. Note que, ao contrário das outras soluções, a estrutura do modelo não é comprometida.

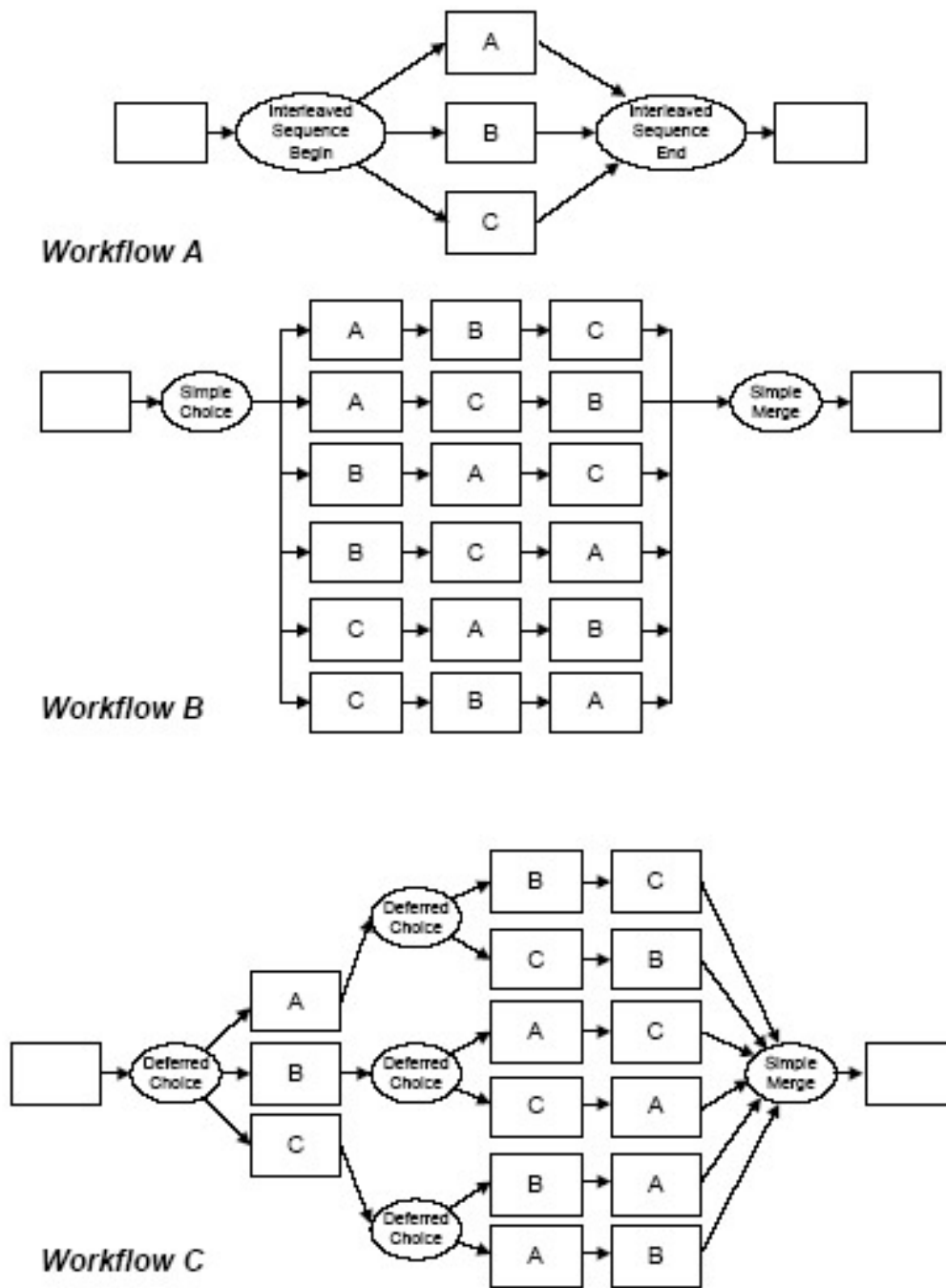


Figura 18: As opções de implementação para execução intercalada de A, B e C

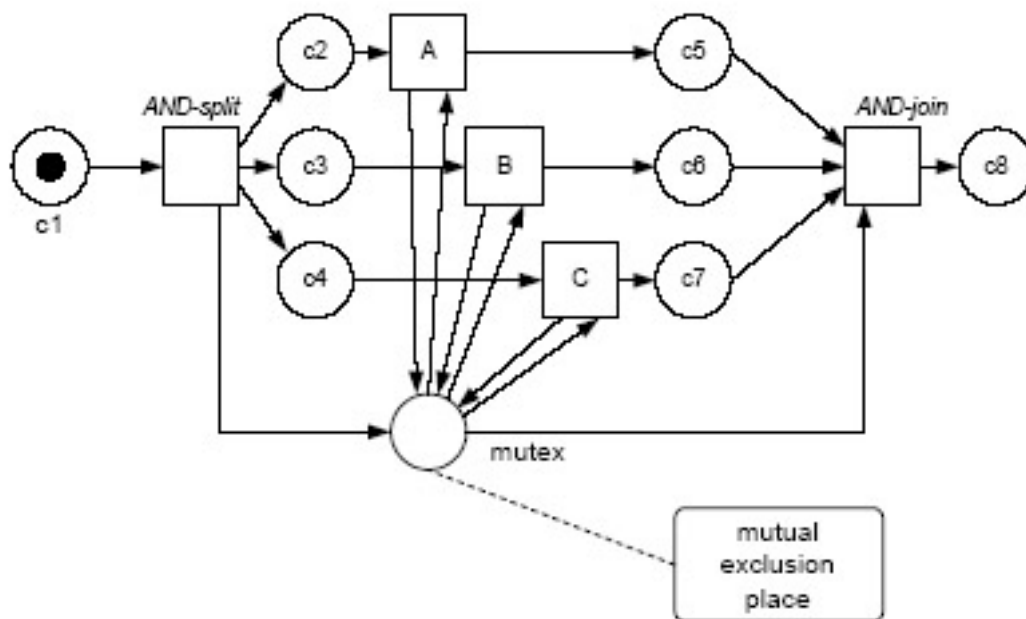
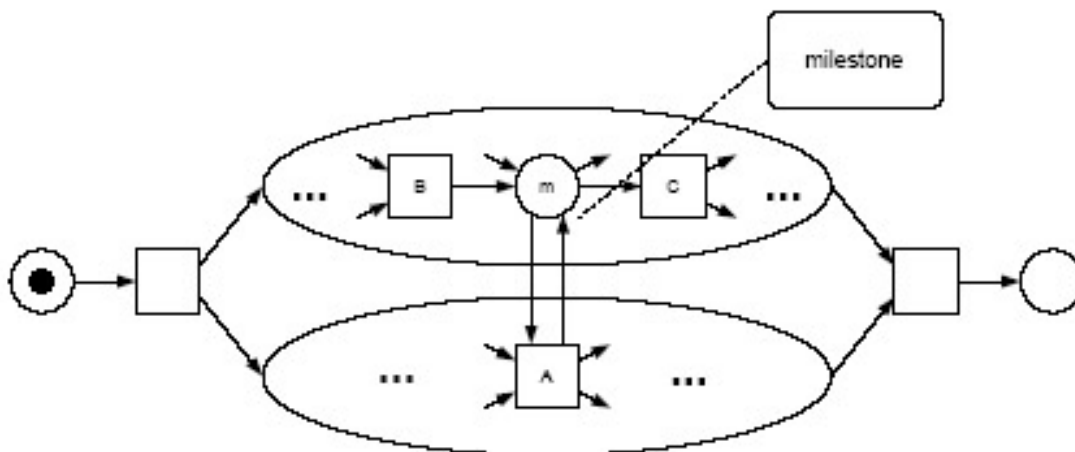


Figura 19: A execução de  $A$ ,  $B$  e  $C$  é intercalada adicionando-se um local de exclusão mútua

### 3.2.5.3. Padrão 18 (Milestone)

**Descrição:** A habilitação de uma atividade depende do caso estar em um estado especificado, isto é, a atividade é habilitada apenas se certo marco tenha sido alcançado e se ela não tenha expirado ainda. Considere três atividades, chamadas  $A$ ,  $B$  e  $C$ . A atividade  $A$  somente é habilitada se a atividade  $B$  foi executada e  $C$  não foi executada ainda, isto é,  $A$  não é habilitada antes da execução de  $B$  e não é habilitada depois da execução de  $C$ . A figura 20 ilustra este padrão. O estado no meio de  $B$  e  $C$  é modelado pelo local  $m$ . Este local é um marco para  $A$ . Note que  $A$  não remove o *token* de  $m$ : ele apenas testa a presença de um *token*.

**Sinônimos:** *Test arc, deadline, state condition, withdraw message.*



**Figura 20: Representação esquemática de um milestone**

**Problema:** O problema é similar ao problema mencionado no padrão 16 (*Deferred Choice*): existe uma competição entre um número de atividades e a execução de algumas atividades pode desabilitar outras. Na maioria dos sistemas de *workflow* (exceções notáveis são aqueles baseados em redes de Petri) uma vez que uma atividade se torna habilitada, não existe outro jeito de desabilitá-la a não ser de maneira programática. Um marco pode ser usado para testar se uma parte do processo está em um dado estado. Mecanismos de passagem de mensagens simples não serão capazes de suportar isto pois a desativação de um marco corresponde a retirada de uma mensagem. Este tipo de funcionalidade não é tipicamente oferecido pelos WFMS existentes.

**Implementação:**

- Considere três atividades *A*, *B* e *C*. A atividade *A* pode ser executada um número arbitrário de vezes antes da execução de *C* e depois da execução de *B*, conforme *Workflow A* na figura 21. Desta maneira um marco pode ser realizado usando o padrão 16 (*Deferred Choice*). Depois de executar *B* existe um *XOR-split* implícito, com duas possíveis sub-atividades subseqüentes: *A* e *C*. Se *A* é executada, então o mesmo *XOR-split* implícito é ativado de novo. Se *C* é executada, *A* é desabilitada pela construção *XOR-split* implícita. Esta solução é ilustrada pelo *Workflow B* na figura 21. Note que esta solução funciona apenas se a execução de *A* não for controlada por outras *threads* paralelas.
- Outra solução é usar a perspectiva de dados, por exemplo, introduzindo uma variável de *workflow* booleana *m*. Considere novamente três atividades *A*, *B* e *C*, sendo que à atividade *A* é permitido que seja executada entre as atividades *B* e *C*.



Inicialmente,  $m$  é definida como *false*. Depois da execução de  $B$ ,  $m$  é definida como *true*, e a atividade  $C$  define  $m$  como *false*. A atividade  $A$  é precedida por um laço que periodicamente checa se  $m$  é *true*. Se  $m$  é *true*, então  $A$  é ativada e se  $m$  é *false*, então se checa novamente depois de um determinado período. Esta solução é ilustrada no *Workflow C* na figura 21. Note que desta maneira um “*busy wait*” é introduzido e depois que  $A$  é habilitada ele não pode mãos ser bloqueado, isto é, a execução de  $C$  não influencia instâncias em execução ou habilitadas de  $A$ . Usando o padrão 19 (*Cancel Activity*),  $A$  pode ser retirada uma vez que  $C$  é iniciada. Variantes mais sofisticadas desta solução são possíveis usando-se gatilhos de bancos de dados, etc. Entretanto, uma desvantagem desta abordagem de solução é que uma parte essencial da perspectiva de processo é escondida dentro de atividades e aplicações. Além disso, a mescla de paralelismo e escolha pode levar a todos os tipos de problemas de concorrência.

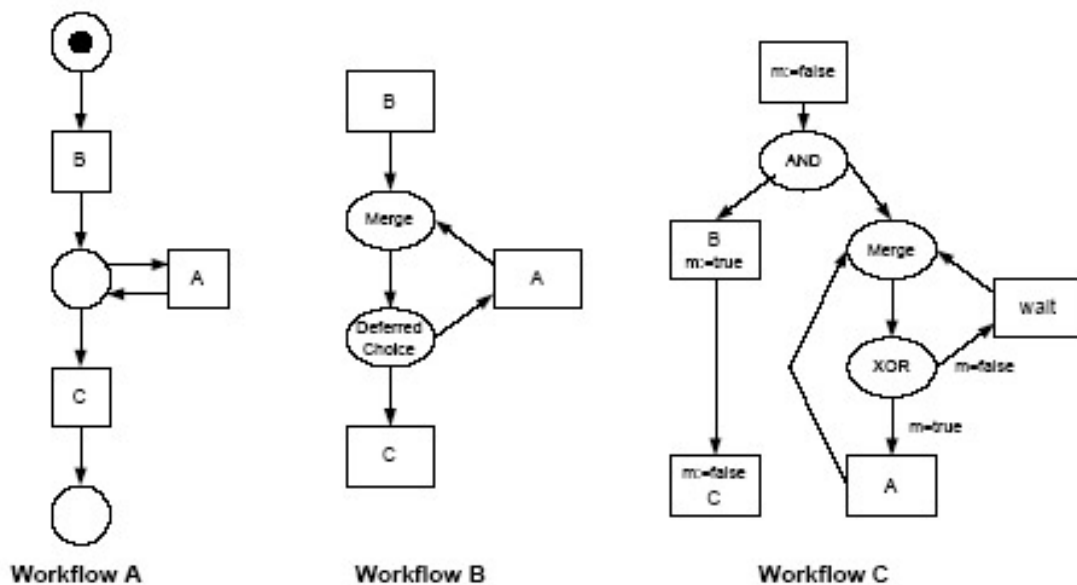


Figura 21: O padrão *milestone* nas suas mais simples formas: implementado usando uma rede de Petri (*Workflow A*), implementado usando *deferred choice* (*Workflow B*), e implementado usando um *busy wait* (*Workflow C*)

### 3.2.6. Padrões de Cancelamento

Nesta seção serão apresentados os padrões utilizados para cancelamento de atividades e instâncias de *workflow*.

### 3.2.6.1. Padrão 19 (Cancel Activity)

**Descrição:** Uma atividade habilitada é desabilitada, isto é, uma *thread* esperando pela execução de uma atividade é removida.

**Sinônimos:** *Withdraw activity*.

**Problema:** Apenas uma pequena quantidade de WFMS suporta a remoção de uma atividade diretamente na linguagem de modelagem de *workflow*, isto é, de maneira (semi-) gráfica.

**Implementação:**

- Se a linguagem de *workflow* suporta o padrão 16 (*Deferred Choice*), então é possível cancelar uma atividade adicionando-se uma então chamada “*shadow activity*”. Ambas, a atividade real e a atividade “sombra” são precedidas por uma *deferred choice*. Além do mais, a atividade sombra não requer interação humana e é disparada pelo sinal de cancelamento da atividade. Considere por exemplo uma linguagem de *workflow* baseada em redes de Petri. Uma atividade é cancelada removendo-se o *token* de cada um de seus locais de entrada. Os *tokens* são removidos executando-se outra atividade tendo o mesmo conjunto de locais de entrada. Note que a desvantagem desta solução é a introdução de atividades que não correspondem aos passos atuais do processo.
- Muitos WFMS suportam a retirada de atividades usando uma API que simplesmente remove a entrada correspondente no banco de dados, isto é, não é possível modelar o cancelamento de atividades de uma maneira direta e gráfica, mas dentro das atividades pode-se iniciar uma função que desabilita outra atividade.

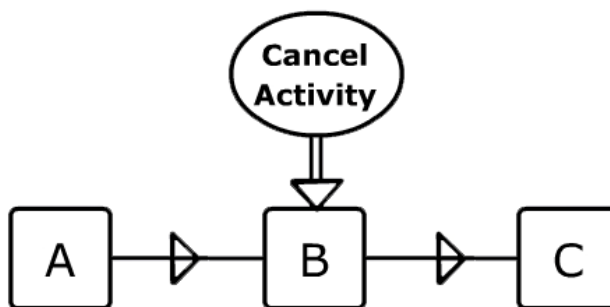


Figura 22: Implementação do padrão cancel activity

### 3.2.6.2. Padrão 20 (Cancel Case)

**Descrição:** Um caso, isto é, uma instância de *workflow*, é removido completamente (isto é, mesmo se partes do processo são instanciadas múltiplas vezes, todos os descendentes são removidos).

**Sinônimos:** *Withdraw case*.

**Problema:** WFMS tipicamente não suportam a retirada de um caso inteiro usando linguagem (gráfica) de *workflow*.

**Implementação:**

- O padrão 19 (*Cancel Activity*) pode ser repetido para cada atividade na definição do processo de *workflow*. Existe uma atividade disparando a retirada de cada atividade no *workflow*. Note que esta solução não é muito elegante, uma vez que o controle normal de fluxo é entrelaçado com todos os tipos de conexões unicamente introduzidas para remoção da instância de *workflow*.
- Similar ao padrão 19 (*Cancel Activity*), muitos WFMS suportam a retirada de casos usando uma API que simplesmente remove as entradas correspondentes do banco de dados.

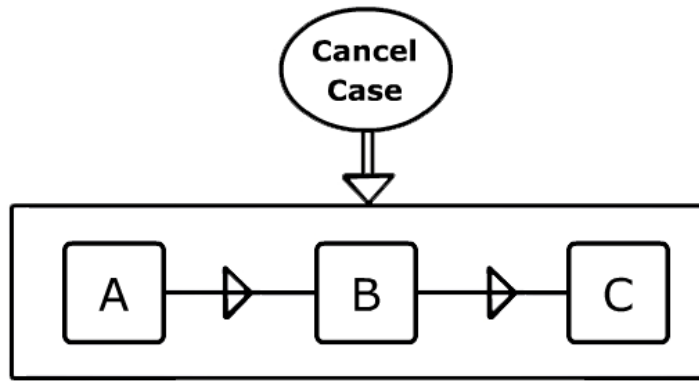


Figura 23: Implementação do padrão cancel case

## 4. Workflow Management Systems

Com o objetivo de identificar quais WFMS que implementam padrões de *workflow* estão disponíveis no mercado, e quais padrões são implementados por tais sistemas, realizou-se uma pesquisa com este foco.

As ferramentas estão divididas em duas categorias: WFMS de Código Aberto, que estão disponíveis sob licença de software livre, e WFMS Comerciais, cujas licenças são necessariamente adquiridas comercialmente.

O foco deste trabalho é a identificação de padrões de *workflow* em ferramentas de código aberto. As ferramentas pesquisadas são: JBoss jBPM, OpenWFE, OpenSymphony Workflow, WfMOpen, OpenFlow e ObjectWeb Bonita. As informações a respeito de WFMS de código aberto foram coletadas dos *websites* dos projetos, documentações, contato com desenvolvedores e das próprias aplicações, principalmente através dos casos de teste de unidade.

As informações de WFMS comerciais foram coletadas de [AALST 02a] e dos *websites* dos produtos.

Logo após a descrição dos WFMS, são apresentadas tabelas com o cruzamento das ferramentas com os padrões de *workflow*

### 4.1. WFMS de Código Aberto

#### 4.1.1. JBoss jBPM

JBoss jBPM é o mais poderoso WFMS de código aberto, e combina fácil desenvolvimento de aplicações de *workflow* com capacidades de integração de aplicações empresariais. JBoss jBPM pode ser usado tanto nos mais simples ambientes quanto escalado para tratar os mais complexos padrões de *workflow* em uma aplicação J2EE em *cluster*. [JBoss 05]

JBoss jBPM resolve a lacuna encontrada entre analistas de negócio e desenvolvedores, dando a eles uma linguagem comum – a *JBoss jBPM Process Definition Language* (jPdl) – com a qual são definidos os processos de negócio.

Processos de negócio, expressados nesta simples e poderosa linguagem, servem como entrada para o servidor JBoss jBPM, que mantém o estado, assim como gera registros das operações e realiza todas as ações automatizadas definidas no processo de negócio.

Os destaques de JBoss jBPM incluem:

- Licença de produto com custo zero (Código aberto)
- Núcleo da *engine* poderoso e simples
- Suporta todos os padrões de *workflow*
- Gestão de estados abrangente
- Liberdade de preparação para lidar tanto com ambientes simples e complexos
- Modelo de programação fácil
- Desenvolvimento dirigido a testes para *workflow*

Essencialmente, jBPM pode ser usado em qualquer programa Java como:

- Um aplicativo *web*, rodando em um *servlet container*, como JBoss e Tomcat
- Uma tarefa do *Ant*<sup>3</sup> – jBPM inclui *tasks* de *ant* para empacotar e preparar arquivos de processo
- Um teste de JUnit – A configuração padrão do jBPM é ideal para desenvolver e testar arquivos de processo. Ele usa banco de dados *hypersonic*, transitório e em memória, o que significa que não é necessário configurar um banco de dados para inicializá-lo. jBPM detecta que nenhuma tabela de seu banco de dados está presente e as cria automaticamente.
- Um aplicativo customizado – Quando é necessária a incorporação de gestão de *workflows*, é possível usar jBPM como um componente para a aplicação sem que os usuários do sistema percebam.

---

<sup>3</sup> *Apache Ant* é uma ferramenta de construção de sistemas baseada em Java. Ao invés de um modelo onde ele é estendido com comandos baseados em linha de comando, como os comandos Unix, *Ant* é estendido usando classes Java. Ao invés de escrever comandos em linha de comando, os arquivos de configuração são baseados em XML, chamando uma árvore alvo, onde várias tarefas são executadas. Cada tarefa é executada por um objeto que implementa uma interface particular de Tarefa. [APACHE 05]

### 4.1.2. OpenWFE

OpenWFE é uma *engine* de *workflow* de código aberto, implementada em Java, disponível sob a licença *BSD*. OpenWFE não é caracterizado apenas como uma engine de workflow, mas também como um WFMS completo, e apresenta, entre outras, os seguintes recursos:

- Um componente de *worklists*, para armazenar *workitems* (tarefas) para os participantes.
- Uma APRE (*Automatic Participant Runtime Environment*), permitindo implementar agentes automatizados para os fluxos.
- Droflo, uma interface *web* para modelar processos de negócio.
- Uma interface REST<sup>4</sup>, bem documentada, e várias bibliotecas para acessá-la.

OpenWFE é disponível atualmente somente como uma *suíte* completa, e pode rodar completamente de forma independente, sem nenhum servidor de aplicação, exigindo apenas um SDK Java (1.4.x) ou maior.

OpenWFE também várias bibliotecas de acesso externo para facilitar o acesso à interface REST através de uma variedade de linguagens, incluindo:

- OpenWFE-dotnet, uma biblioteca .Net escrita em C#, que permite acessar todas as funções REST através de uma API .Net fácil de implementar.
- OpenWFE-pyya é a equivalente, escrita em Python, e inclui uma versão em Python do OpenWFE APRE.
- OpenWFE-perl, um módulo atualmente em desenvolvimento, que irá permitir a integração com o OpenWFE através de aplicações ou *web services* escritos em Perl.

As definições de *workflow* são expressas em uma linguagem de definição de processos própria e extensível que, segundo os autores, foi “forjada” para afrontar os desafios ditados por cada um dos padrões de *workflow*. [OPENWFE 05]

---

<sup>4</sup> REST é uma interface baseada em HTTP, usando chamadas como GET e POST.

### 4.1.3. OpenSymphony Workflow – OSWorkflow

OSWorkflow é razoavelmente diferente da maioria dos sistemas de *workflow* disponíveis, tanto comercialmente quanto no mundo do código aberto. Segundo os autores, o que faz esta diferença é sua extrema flexibilidade. Por exemplo, OSWorkflow não possui ferramentas gráficas para desenvolver *workflows*, e a abordagem recomendada é escrever os descritores de *workflow* em XML “na mão”. É responsabilidade do desenvolvedor de aplicações proporcionar este tipo de integração, assim como qualquer integração com o código e bancos de dados existentes. Isto pode parecer um problema para alguém que está procurando por uma solução de *workflow* rápida *plug-and-play*, mas os autores acham que este tipo de solução nunca provê flexibilidade suficiente para cumprir todos os requisitos em uma aplicação já em andamento.

OSWorkflow pode ser considerado uma implementação de *workflow* de baixo nível. Situações como laços e condições, que seriam representadas por um ícone em outros sistemas de *workflow* precisam ser codificadas no OSWorkflow. Não é para dizer que código atual é necessário para implementar situações como estas, mas uma linguagem de *scripting* precisa ser empregada para estas condições específicas. Não é esperado que um usuário não técnico modifique um *workflow*. Os autores acreditam que apesar de alguns sistemas fornecerem interfaces gráficas que permitem a edição simples de *workflows*, as aplicações adjacentes ao *workflow* habitualmente terminam danificadas quando mudanças como esta são feitas. É melhor que estas mudanças sejam feitas por um desenvolvedor que é informado de cada mudança. Tendo dito isto, a última versão provê uma *GUI designer*, que pode ajudar com a edição do *workflow* [OPENSYMPHONY 05].

### 4.1.4. WfMOpen

WfMOpen é uma implementação de uma *engine* de *workflow* baseada em J2EE, como proposto pela WfMC e OMG. O componente de *workflow* é baseado em um conjunto de interfaces Java que definem uma API para um recurso de gestão de *workflow*. As interfaces *omgcore* básicas seguem a especificação *Workflow Management Facility Specification*, V1.2, da OMG, de muito perto, enquanto se faz



algumas modificações para adaptar o serviço CORBA para as práticas de projeto estabelecidas para uma API Java. Os *workflows* são especificados usando XPDL (*XML Process Definition Language*), da WfMC, com algumas extensões.

Apesar do *WfMCore* ter suas raízes nos padrões citados acima, a atual implementação transcende o domínio tradicional de *engines* de *workflow*. Por projeto, *WfMCore* fornece excelente escalabilidade e suporta tecnologias de integração úteis, incluindo SOAP. O componente de *workflow* pode, conseqüentemente, ser usado como o núcleo para qualquer implementação de aplicações baseadas em processos, e é apropriado para fornecer soluções para trabalhos relacionados a BPM [WFMOPEN 05].

#### 4.1.5. OpenFlow

OpenFlow é uma *engine* de *workflow* desenvolvida pela *Icube*, uma sociedade italiana voltada ao desenvolvimento de *software* livre, e é distribuída como tal. Ela é baseada em uma estrutura orientada a objetos, possui um sistema poderoso de tratamento de exceções e suporta o re-projeto dinâmico. Segundo os autores, estas características fazem de OpenFlow uma *engine* de *workflow* muito mais flexível do que as outras *engines* existentes. OpenFlow suporta padrões abertos (XML/XML-RPC) e os padrões da internet. Ele também facilita integração entre sistemas heterogêneos graças ao simples acesso que possui na maioria dos bancos de dados relacionais.

Openflow é baseado em atividades, multi-plataforma, baseado na *web*, inspirado na WfMC, construído em integrado com o servidor de aplicação Zope, e é um *software* totalmente grátis [OPENFLOW 05].

#### 4.1.6. ObjectWeb Bonita

Bonita é um sistema de *workflow* cooperativo flexível, de acordo com as especificações da WfMC, baseado no modelo de *workflow* proposto pelo time *ECOO*<sup>5</sup>, que incorpora a antecipação de atividades como um mecanismo mais flexível de execução de *workflow*. Bonita é de código aberto e pode ser baixado sob a licença LGPL. [BONITA 05]

O sistema fornece:

---

<sup>5</sup> *COOperation Environments*, um grupo francês que tem como principal objetivo projetar e implementar serviços básicos tanto quanto desenvolver uma metodologia a fim de permitir a criação de empreendimentos virtuais distribuídos. Sítio na internet: <http://www.loria.fr/equipes/ecoo/english/index.html>

- Um abrangente conjunto de ferramentas gráficas integradas para executar a concepção e definição de processo, a instanciação e controle deste processo, e a interação com os usuários e outras aplicações.
- Ambiente 100% baseado em navegador, com integração com *Web Services*, que utilizam ligações de dados através de SOAP e XML, a fim de encapsular métodos de negócio de *workflow* existentes e publicá-los em *web services* baseados em J2EE.
- Uma *engine* de *workflow* de terceira geração, baseada no modelo de antecipação de atividades. Esta flexibilidade permite um considerável aumento de velocidade nas fases de projeto e desenvolvimento de aplicações cooperativas.

## **4.2. WFMS Comerciais**

### **4.2.1. TIBCO Staffware Process Suite**

TIBCO Staffware é um dos principais WFMS, e era desenvolvido e distribuído pela Staffware PLC, que foi adquirida pela TIBCO Software Inc, em junho de 2004. Em 1998, o *Gartner Group* estimou que a representação do Staffware no mercado global era de 25%.

TIBCO Staffware Process Suite é um *software* de gerência de processos patenteados e abrangente, que pode ser introduzido sem problemas em uma infraestrutura de TI existente. TIBCO Staffware Process Suite é um conjunto de módulos de aplicação construídos em uma arquitetura aberta que é planejada para fornecer uma completa solução fim-a-fim de gerência de processos. Ele permite às organizações criar uma infra-estrutura de TI que seja baseada em seus processos de negócio – suas únicas maneiras de se fazer negócio. Separando a lógica de aplicação da camada de processo, ele permite a criação de uma camada que fornece uma abstração de processo, e remove os processos do controle das aplicações. Esta camada de abstração de processo, a camada independente de processo, é a chave para a flexibilidade e agilidade do processo. [TIBCO 05]

TIBCO Staffware Process Suite é composto dos seguintes principais componentes: [TIBCO 05]

- Modelagem – TIBCO Staffware Process Suite permite que pessoal não pertencente à equipe de TI e especialistas em negócio modelem processos de negócio através de um ambiente amigável e gráfico de modelagem, com suporte completo para controle de versões. O mapa de processo resultante é o guia para integrar pessoas, processos e aplicações.
- Execução – TIBCO Staffware Process Suite é equipado com o *TIBCO iProcess Engine*, uma poderosa *engine* de gerência de processos projetada para manipular volumes extremamente alto, transações de missão críticas através de múltiplos servidores enquanto mantém a integridade de transações individuais. TIBCO Staffware Process Suite também habilita abrangentes capacidades de integração de aplicações através da plataforma de integração TIBCO *BusinessWorks*, ou outra tecnologia de integração de terceiros.
- Regras – TIBCO Staffware Process Suite fornece uma ferramenta intuitiva para analistas de negócio usando o mesmo nível de prática de um bom usuário de *spreadsheet* para modelar, analisar, testar e gerenciar regras de negócio. A metáfora da tabela de decisão *spreadsheet-like* é prontamente aprendida em minutos.
- Análise – TIBCO Staffware Process Suite fornece uma ferramenta sofisticada para avaliar a efetividade e eficiência de todos os processos de negócio. Ele habilita gerenciamento para estabelecer e continuamente medir Indicadores de Desempenho Chave (KPIs) para execução e melhoria dos processos de saída.

#### 4.2.2. COSA BPM

COSA BPM é um *workflow management system* baseado em redes de Petri, desenvolvido pela empresa COSA GmbH. COSA BPM estabelece novos padrões para representar e controlar processos de negócio no meio eletrônico. Uma ênfase especial é colocada na criação de modelos de processos de maneira intuitiva e individual. [COSA 05]

A base de COSA BPM é uma *engine* de *workflow* poderosa. Ela garante uma fácil integração de aplicações já existentes dentro no processo de negócio controlado por *workflow*. COSA fornece flexibilidade ideal e tempos de ciclos curtos. Como resultado, ambas, qualidade e produtividade do processo são melhoradas consideravelmente. [COSA 05]

### 4.2.3. InConcert

InConcert foi estabelecido em 1996 como uma subsidiária da Xerox. Em 1999 ele foi comprado pela TIBCO Software. Hoje, InConcert foi incorporado ao sistema de integração de negócios da TIBCO chamado TIBCO BusinessWork Workflow [TIBCO 05].

Uma definição de *workflow* no InConcert era chamado de *job*. Um *job* continha nenhuma, uma ou mais atividades. Uma atividade podia ser simples ou composta. Uma atividade pode ser conectada a um número arbitrário de outras atividades, mas dependências circulares não são permitidas. Cada atividade possui uma condição de execução fixada a ela, e o valor padrão desta condição é *true*, de modo que atividades podem, de maneira geral, serem executadas. Se a condição de execução é avaliada para *false*, a atividade é omitida. Se uma atividade é omitida, então as atividades subsequentes não são omitidas automaticamente. Ramificações condicionais ou ramificações de caso podem ser concluídas por atividades paralelas com condições de execução diferentes. No InConcert ciclos arbitrários não são suportados, um ponto de terminação explícita não é necessário, não há provisão de múltiplas instâncias nem para a implementação direta de padrões baseados em estados, e os padrões de cancelamento não são suportados.

### 4.2.4. MQSeries/Workflow

MQSeries/Workflow é o sucessor do FlowMark, antiga oferta de *workflow* da IBM. FlowMark foi um dos primeiros produtos de *workflow* que era independente do gerenciamento de documentos e serviços de trabalho com imagens. Ele foi renomeado para MQSeries/Workflow depois de uma mudança de seu *middleware* proprietário para o *middleware* baseado no produto MQSeries.

No MQSeries/Workflow, o modelo de *workflow* consiste em atividades unidas por transições. Sua *engine* de *workflow* possui uma semântica de execução única, que propaga um *token False* para cada transição com uma condição sendo avaliada como falsa. Isto permite a cada atividade que possui mais de uma transição de entrada que realizem um *synchronizing merge*. A não ser o *synchronizing merge*, que é uma construção natural para MQSeries/Workflow, Não existem maneiras de implementar diretamente nenhum dos padrões de sincronização avançados.

O suporte para múltiplas instâncias é fornecido através do construtor *Bundle*, embora ele não seja adequado se o número de instâncias não é conhecido em nenhum ponto anterior para gerar as instâncias envolvidas. Laços arbitrários não são suportados. Um ponto de terminação explícita não é necessário e o processo de *workflow* irá terminar quando não houver nada mais a ser executado. Além disso, não existe caminho direto para modelar padrões baseados em estados e de cancelamento.

#### 4.2.5. Forté Conductor

Forté Conductor é uma *engine* de *workflow* que é um *add-on* ao ambiente de desenvolvimento da Forté, a Forté 4GL. A *engine* do conductor é baseada em um trabalho experimental realizado na *Digital Research* e sua linguagem de modelagem é poderosa e flexível. Forté Software foi adquirida em 1999 pela Sun Microsystems e subseqüentemente tornou-se parte integrante das *iPlanet E-Commerce Solutions*.

O modelo de *workflow* do Conductor consiste de um conjunto de atividades conectadas por transições, chamadas *routers*. Cada transição possui condições de transição associadas. Cada atividade possui um gatilho que determina a semântica daquela atividade, se existem mais de uma transição de entrada. Os gatilhos são flexíveis o suficiente para a fácil especificação de *OR-joins*, *AND-joins* e junção *N-out-of-M* embora a semântica de tal especificação é implícita e não é visível para o usuário final. Ciclos arbitrários são suportados, mas pontos de terminação explícita são necessários. Forté suporta a criação de múltiplas instâncias diretamente, através do uso de uma junção *multi-merge*, mas não suporta nenhum intermediário direto de sua sincronização subseqüente. Padrões baseados em estados não podem ser realizados, e o Forté não possui um construtor para cancelamento de atividade, mas cancelamento de caso está disponível através de sua semântica de término.

#### 4.2.6. Verve

Verve entrou no mercado de *workflow* no ano de 1998. Em 2000, foi adquirida pela Versata e foi renomeada para Versata Integration Server (VIS).

O que faz a *engine* de *workflow* Verve um produto de *workflow* interessante é que ela foi projetada desde seu início como uma *engine* de *workflow* embutida. A *engine* de *workflow* da Verve é muito poderosa e juntamente com outras características permite múltiplas instâncias e modificação dinâmica de instâncias em execução. O modelo de *workflow* da Verve consiste em atividades conectadas por transições. Cada transição possui uma condição de transição associada. Construções de roteamento extras como *synchronizer* e *discriminator* são suportadas, além de laços arbitrários. Um ponto de término explícito é necessário. Múltiplas instâncias são diretamente suportadas, através do uso de *multi-merge*, enquanto elas não necessitarem de sincronização subsequente. Não existe maneira direta de implementar padrões baseados em estados, e o que diz respeito aos padrões de cancelamento, *cancel case* é suportado através do término forçado pelas atividades “primeiras das últimas” que terminam.

#### 4.2.7. Visual WorkFlo

Visual WorkFlo é um dos líderes de mercado na indústria de *workflow*. Ele é parte do conjunto de aplicativos da Panagon chamado FileNet, que inclui também gerência de documentos e servidores de imagens. Visual WorkFlo é um dos mais antigos e melhores estabelecidos produtos no mercado. Desde sua introdução em 1994, ele foi conduzido para ganhar uma fatia respeitável das aplicações de *workflow* em todo o mundo. FileNet é uma corporação que se classifica entre as 60 maiores companhias de software no mundo.

A linguagem de modelagem de *workflow* é altamente estruturada e é uma coleção de atividades e elementos de roteamento, como ramificação (*XOR-split*), *while* (laço estruturado), *split* estático (*AND-split*), *Rendezvous* (*AND-join*), e *release*. Visual WorkFlo não suporta diretamente nenhum dos padrões de sincronização avançados. O suporte direto a múltiplas instâncias é possível através da construção *release*, enquanto não houver sincronização adicional requerida. Não existe maneira direta de implementar

qualquer um dos padrões baseados em estado, e não existe suporte explícito para padrões de cancelamento.

#### 4.2.8. I-Flow

I-Flow é um *workflow* oferecido pela Fujitsu, sucessor da *engine* de *workflow* da mesma empresa, o TeamWare, e predecessor da atual engine de workflow, o Interstage Business Process Manager [FUJITSU 05]. I-Flow é centrado na *web* e possui uma *engine* baseada em Java/CORBA contruída especialmente para fornecedores de *software* independentes e integradores de sistemas.

O modelo de *workflow* no I-Flow consiste em atividades e um conjunto de construções de roteamento conectadas por transições, chamada *arrows*. Construções de roteamento incluem nodo condicional (*XOR-split*), *OR-NODE (merge)*, e *AND-NODE* (sincronizador). O *AND-split* pode ser modelado implicitamente fornecendo-se uma atividade com mais de uma transição de saída. Múltiplas instâncias podem ser implementadas usando o *Chained Process Node*, que permite a invocação de sub-processos assíncronos. Laços arbitrários são permitidos, mas o processo necessita de um ponto de término explícito. Não existe maneira direta de implementar padrões baseados em estados. *Cancel case* é suportado, mas *cancel activity* não.

#### 4.2.9. SAP R/3 Workflow

SAP é o principal *player* no mercado de sistemas ERP. Seu pacote de software R/3 inclui um componente de *workflow* integrado. SAP *workflow* não pode ser confundido com EPCs (Event-driven Process Chains) encontradas em outras partes do sistema SAP. EPCs são usadas inteiramente para propósitos de modelagem de processos de negócio, e não para modelar *workflows* executáveis no SAP R/3 *runtime environment*. SAP R/3 Workflow impõe um número de restrições no uso de EPCs. EPCs que são usadas para modelagem de *workflow* consistem em um conjunto de funções (atividades), eventos e conectores (*AND*, *XOR*, *OR*). Entretanto, no SAP R/3 Workflow, nem todo o poder expressivo das EPCs pode ser usado, pois há um número de restrições sintáticas bastante similares às restrições impostas pelo FileNet Visual

Workflo. Não existe provisão direta para construções de sincronização avançadas, múltiplas instâncias, laços arbitrários e padrões baseados em estado e de cancelamento.



### 4.3. WFMS x Padrões de Workflow

As tabelas a seguir resumam os resultados da pesquisa realizada nos *workflow management systems*, em termos de suporte aos padrões de *workflow* propostos. Para cada combinação, é indicado se o produto suporta ou não o padrão. Como convenção, atribui-se o valor “Sim” para padrões suportados direta e/ou indiretamente pelo produto, e “Não” para padrões não suportados pelo produto.

A tabela 1 apresenta os WFMS de código aberto. Verificou-se que duas ferramentas implementam todos os padrões de *workflow*: o JBoss jBPM e o OpenWFE. A maneira com as quais estas ferramentas implementam os padrões será apresentada no próximo capítulo. As tabelas 2 e 3 apresentam os WFMS Comerciais disponíveis no mercado.

Padrão	Produto					
	jBPM	OpenWFE	OSWorkflow	WfMOpen	OpenFlow	Bonita
1 (seq)	Sim	Sim	Sim	Sim	Sim	Sim
2 (par-spl)	Sim	Sim	Sim	Sim	Sim	Sim
3 (synch)	Sim	Sim	Sim	Sim	Sim	Sim
4 (ex-ch)	Sim	Sim	Sim	Sim	Sim	Sim
5 (simple-m)	Sim	Sim	Sim	Sim	Sim	Sim
6 (m-choice)	Sim	Sim	Sim	Sim	Não	Sim
7 (sync-m)	Sim	Sim	Não	Sim	Não	Não
8 (multi-m)	Sim	Sim	Sim	Não	Sim	Não
9 (disc)	Sim	Sim	Sim	Não	Não	Não
10 (arb-c)	Sim	Sim	Sim	Sim	Sim	Sim
11 (impl-t)	Sim	Sim	Sim	Sim	Não	Sim
12 (mi-no-s)	Sim	Sim	Sim	Sim	Sim	Sim
13 (mi-dt)	Sim	Sim	Sim	Sim	Não	Sim
14 (mi-rt)	Sim	Sim	Sim	Não	Não	Sim
15 (mi-no)	Sim	Sim	Sim	Não	Não	Sim
16 (def-c)	Sim	Sim	Não	Sim	Sim	Não
17 (int-par)	Sim	Sim	Não	Não	Não	Não
18 (milest)	Sim	Sim	Não	Não	Não	Não
19 (can-a)	Sim	Sim	Sim	Não	Sim	Sim
20 (can-c)	Sim	Sim	Sim	Não	Sim	Não

Tabela 1: WFMS de Código Aberto x Padrões de Projeto

Padrão	Produto				
	Staffware	COSA	InConcert	MQSeries	Forté
1 (seq)	Sim	Sim	Sim	Sim	Sim
2 (par-spl)	Sim	Sim	Sim	Sim	Sim
3 (synch)	Sim	Sim	Sim	Sim	Sim
4 (ex-ch)	Sim	Sim	Sim	Sim	Sim
5 (simple-m)	Sim	Sim	Sim	Sim	Sim
6 (m-choice)	Não	Sim	Sim	Sim	Sim
7 (sync-m)	Não	Sim	Sim	Sim	Não
8 (multi-m)	Não	Não	Não	Não	Sim
9 (disc)	Não	Não	Não	Não	Sim
10 (arb-c)	Sim	Sim	Não	Não	Sim
11 (impl-t)	Sim	Não	Sim	Sim	Não
12 (mi-no-s)	Não	Sim	Não	Não	Sim
13 (mi-dt)	Sim	Sim	Sim	Sim	Sim
14 (mi-rt)	Não	Não	Não	Não	Não
15 (mi-no)	Não	Não	Não	Não	Não
16 (def-c)	Não	Sim	Não	Não	Não
17 (int-par)	Não	Sim	Não	Não	Não
18 (milest)	Não	Sim	Não	Não	Não
19 (can-a)	Sim	Sim	Não	Não	Não
20 (can-c)	Não	Não	Não	Sim	Não

Tabela 2: WFMS Comerciais x Padrões de Projeto (I)

Padrão	Produto			
	Verve	Vis. WF	I-Flow	SAP/R3
1 (seq)	Sim	Sim	Sim	Sim
2 (par-spl)	Sim	Sim	Sim	Sim
3 (synch)	Sim	Sim	Sim	Sim
4 (ex-ch)	Sim	Sim	Sim	Sim
5 (simple-m)	Sim	Sim	Sim	Sim
6 (m-choice)	Sim	Sim	Sim	Sim
7 (sync-m)	Não	Não	Não	Não
8 (multi-m)	Sim	Não	Não	Não
9 (disc)	Sim	Não	Não	Sim
10 (arb-c)	Sim	Sim	Sim	Não
11 (impl-t)	Não	Não	Não	Não
12 (mi-no-s)	Sim	Sim	Sim	Não
13 (mi-dt)	Sim	Sim	Sim	Sim
14 (mi-rt)	Não	Não	Não	Sim
15 (mi-no)	Não	Não	Não	Não
16 (def-c)	Não	Não	Não	Não
17 (int-par)	Não	Não	Não	Não
18 (milest)	Não	Não	Não	Não
19 (can-a)	Não	Não	Não	Sim
20 (can-c)	Não	Sim	Sim	Não

**Tabela 3: WFMS Comerciais x Padrões de Projeto (II)**

## 5. Padrões de Workflow no jBPM e OpenWFE

No capítulo anterior realizou-se um levantamento dos WFMS disponíveis. Identificou-se, com esta pesquisa, que duas ferramentas se propõem a implementar todos os padrões de *workflow*: o JBoss jBPM e o OpenWFE.

Neste capítulo estudaremos mais a fundo as ferramentas, identificando a maneira com que os padrões são implementados nelas. Como vimos no capítulo 3, um dos elementos que compõem um padrão de *workflow* são as sugestões de implementação. Pretende-se neste capítulo, também, analisar as implementações das ferramentas, traçando um paralelo e criticando a implementação, com base nas sugestões registradas nos padrões.

### 5.1. Padrão 01 (Sequence)

O padrão *sequence* é implementado no OpenWFE de maneira direta, através da marcação *sequence* na linguagem de definição de processos do OpenWFE, que cobre este padrão. A seguir é mostrado um exemplo da definição do padrão *sequence*, na linguagem de definição de processos do OpenWFE.

```
<sequence>
  <participant ref="a" />
  <participant ref="b" />
</sequence>
```

No exemplo, é declarada uma seqüência com duas tarefas. Com esta marcação, o processo passa por “a”, e depois por “b”.

No jBPM, o padrão *sequence* também é suportado de maneira direta. Na linguagem de definição de processos do jBPM, o jPDL, dentro da marcação *state*, do estado do workflow, pode-se indicar qual é o próximo estado, através da marcação *transition*, com o próximo estado como valor do atributo *to*. A seguir, um exemplo da definição de uma seqüência no jBPM.

```
<process-definition>
  <start-state name='inicio'>
    <transition to='a' />
  </start-state>
  <state name='a'>
```

```

    <transition to='b' />
</state>
<state name='b'>
    <transition to='c' />
</state>
<state name='c'>
    <transition to='fim' />
</state>
<end-state name='fim' />
</process-definition>

```

O padrão *sequence* é suportado de forma semelhante em todos os WFMS, por haver apenas uma implementação, que é a ligação incondicional entre as atividades. Comparando-se as duas ferramentas, o a definição do processo no OpenWFE é mais simples, pois a marcação *sequence* define as transições de forma implícita, de um participante para outro, e no jBPM é necessário que as transições sejam definidas.

## 5.2. Padrão 02 (*Parallel Split*)

O *parallel split* ocorre de maneira explícita. A linguagem de definição de processos do OpenWFE possui uma marcação *concurrency*, onde uma cópia do item de trabalho é enviado para cada um dos estados seguintes. A seguir é mostrado um exemplo da definição do padrão *parallel split*, na linguagem de definição de processos do OpenWFE.

```

<concurrency>
    <participant ref="a" />
    <participant ref="b" />
</concurrency>

```

O jBPM suporta o padrão *parallel split* através de *AND-splits* explícitos. Em jPDL, aninhado à marcação *fork*, pode ser feita a definição de quais estados devem ser atingidos pela divisão. A seguir é mostrado um exemplo de como é realizado o *AND-split* em jBPM.

```

<process-definition>
    <start-state name='inicio'>
        <transition to='and' />
    </start-state>
    <fork name='and'>
        <transition name='primeira' to='a' />
        <transition name='segunda' to='b' />
    </fork>
    <state name='a' />
    <state name='b' />
</process-definition>

```

No exemplo, depois da execução do estado “início”, é realizada uma transição para o *fork*. Do *fork*, partem duas transições, uma para o estado “a” e outra para o estado “b”.

Comparando-se as duas ferramentas, no OpenWFE, da mesma maneira que no padrão *sequence*, a definição do processo é mais simples do que no jBPM. Na linguagem de definição de processos do jBPM, o jPDL, fica mais claro do que no OpenWFE o uso do *AND-split*, através da marcação *fork*.

### 5.3. Padrão 03 (Synchronization)

No OpenWFE, a marcação *concurrency*, na linguagem de definição de processo, divide o fluxo em ramificações paralelas. Atributos da expressão *concurrency* são usados para definir como a sincronização deve ocorrer quando cada ramificação “responde” no final da *concurrency*. A seguir é mostrado um trecho de código especificando os atributos da expressão de sincronização (*sync*) genérica, com seus potenciais valores.

```
<concurrency
  sync="generic"
  count="x|*"
  merge="first|last|highest|lowest"
  merge-type="mix|override"
  remaining="cancel|forget"
>
  (...)
</concurrency>
```

O atributo *count*, quando omitido, possui como valor padrão o “\*”, que significa que a expressão *sync* comanda que se esperem todas as ramificações. Quando o valor atribuído a *count* for um número inteiro positivo, significa que o *sync* irá esperar enquanto o número de ramificações dados houverem respondido, e então ele irá ordenar que as marcações *concurrency* (ou *concurrent-iterator*) respondam para seus pais.

O atributo *merge* indica qual ramificação filho irá ganhar a prioridade no jogo de fusão de itens de trabalho.

No atributo *merge-type*, quando o valor “override” é definido, o item de trabalho que possui a prioridade sobrepõe completamente o outro item de trabalho. Quando o valor é definido como “mix”, os campos de itens de trabalho são considerados um após

o outro, o item de trabalho resultante é composto iniciando com os campos do item de trabalho de menos prioridade até a prioridade mais alta e a sobreposição ocorre no nível de campo.

O *jBPM* suporta o padrão *synchronization* através de *AND-joins* explícitos. A definição em jPDL deste padrão é mostrada no código abaixo.

```
<process-definition>
  <start-state name='inicio'>
    <transition to='and-split' />
  </start-state>
  <fork name='and-split'>
    <transition name='primeira' to='a' />
    <transition name='segunda' to='b' />
  </fork>
  <state name='a'>
    <transition to='and-join' />
  </state>
  <state name='b' />
    <transition to='and-join' />
  </state>
  <join name='and-join'>
    <transition to='fim' />
  </join>
  <end-state name='fim' />
</process-definition>
```

Depois da realização de um *fork*, todas as atividades paralelas possuem uma transição para um mesmo estado, onde é feita a sincronização. Este estado é declarado através da marcação *join*.

Neste padrão, assim como no padrão *parallel split*, a definição do *AND-join* no jPDL, do *jBPM*, é muito mais clara e direta, através da marcação *join*. O *OpenWFE*, apesar de não possuir uma marcação própria para realizar junções em sua linguagem de definição de processos, suporta este padrão com atributos da marcação *concurrency*, e permite uma certa flexibilidade no que diz respeito à sincronização.

#### **5.4. Padrão 04 (Exclusive Choice)**

No *OpenWFE*, o padrão *exclusive choice* é suportado pela marcação *if*. O projetista de *workflow* deve emular a exclusividade de escolha, especificando condições de transição exclusivas. A primeira marcação aninhada à marcação *if* deve ser do tipo *boolean*. A segunda marcação aninhada de *if* é executada se a expressão do tipo *boolean*

é identificada como verdadeira. Se não, a terceira marcação aninhada é executada. O trecho de código a seguir exemplifica a especificação de *exclusive choice*, na linguagem de definição de processos do OpenWFE.

```
<if>
  <equals field="assunto" value="aprovação crédito" />
  <participant ref="a" />
  <participant ref="b" />
</if>
```

No exemplo, se o campo “assunto” for igual a “aprovação crédito”, a atividade “a” é disparada. Caso contrário, a atividade “b” é disparada.

No jPBM, o projetista de *workflow* também deve emular a exclusividade de escolha, especificando condições de transição exclusivas. O padrão *exclusive choice* pode ser especificado, em jPDL, através da marcação *decision*. A seguir é mostrado um exemplo de especificação de *exclusive choice*, em jPDL.

```
<process-definition>
  <start-state name='start'>
    <transition to='a' />
  </start-state>
  <state name='a'>
    <transition to='xor' />
  </state>
  <decision name='xor'>
    <transition name='urgent' to='b'>
      <condition>prioridade==1</condition>
    </transition>
    <transition name='dont care' to='c'>
      <condition>prioridade==2</condition>
    </transition>
    <transition name='forget about it' to='d' />
  </decision>
  <state name='b' />
  <state name='c' />
  <state name='d' />
</process-definition>
```

Dentro da marcação *decision*, são especificadas as transições desejadas. Dentro das transições, existe uma condição que deve ser satisfeita, definida através da marcação *condition*. No exemplo, se o valor de “prioridade” for igual a 1, a transição irá para o estado “b”. Se for igual a 2, a transição irá para o estado “c”, e se não satisfizer nenhuma das condições, a transição irá para o estado “d”.



A implementação do padrão *exclusive choice* em ambas as ferramentas, jBPM e OpenWFE, é muito parecida, apesar da diferença das definições em suas linguagens de definição de processos.

## 5.5. Padrão 05 (Simple Merge)

No OpenWFE o padrão *simple merge* é implicitamente suportado pela expressão *if*. O trecho de código a seguir mostra um exemplo de especificação do padrão *simple merge*, na linguagem de definição de processos do OpenWFE.

```
<if>
  <equals field-value="category" other-value="new plant" />
  <participant ref="reviewer-b" />
  <participant ref="reviewer-c" />
</if>
```

Em jBPM, cada nodo possui um *merge* implícito à sua frente. Então não é necessário o uso do nodo de *merge*. jBPM suporta fusão tanto de caminhos alternativos de execução quanto de caminhos concorrentes de execução.

No primeiro exemplo de código apresentado, o nodo de merge é demonstrado exatamente como no padrão *simple merge*.

```
<process-definition>
  <start-state name="inicio">
    <transition to="a" name="para-a"/>
    <transition to="b" name="para-b"/>
  </start-state>
  <state name="a">
    <transition to="xor"/>
  </state>
  <state name="b">
    <transition to="xor"/>
  </state>
  <merge name="xor">
    <transition to="c"/>
  </merge>
  <state name="c"/>
</process-definition>
```

O segundo exemplo é uma variante implícita, onde é demonstrada a fusão de caminhos concorrentes.

```
<process-definition>
  <start-state name="inicio">
    <transition to="a" name="para-a"/>
    <transition to="b" name="para-b"/>
  </start-state>
```

```

<state name="a">
  <transition to="c"/>
</state>
<state name="b">
  <transition to="c"/>
</state>
<state name="c"/>
</process-definition>

```

No jBPM, o *simple merge* é definido de uma maneira mais nítida do que no OpenWFE. Não fica claro, na definição de processo do OpenWFE, como é feita a fusão de dois ou mais processos.

## 5.6. Padrão 06 (Multi-choice)

O OpenWFE suporta o padrão *multi-choice* através de uma combinação dos padrões *parallel split* (padrão 02) e *exclusive choice* (padrão 04). O trecho de código a seguir mostra a definição do padrão *multi-choice* na linguagem de definição de processos do OpenWFE.

```

<concurrency>
  <if>
    <equals field-value="assunto" other-value="aprovação crédito" />
    <participant ref="a" />
  </if>
  <if>
    <equals field-value="categoria" other-value="nova instalação" />
    <participant ref="b" />
    <participant ref="c" />
  </if>
</concurrency>

```

No jBPM, pode-se especificar condições nas transições, e o padrão *multi-choice* pode ser especificado diretamente. O projetista do *workflow* deve especificar as condições desejadas para cada transição. A seguir é mostrado um exemplo de como o padrão *multi-choice* é definido em jPDL. Logo após, é mostrado um exemplo de como é feita a especificação das condições das transições, em Java.

```

<process-definition>
  <start-state name="start">
    <transition to="a"/>
  </start-state>
  <state name="a">
    <transition to="multichoice"/>
  </state>
  <fork name="multichoice">
    <transition to="b" name="to b"/>
    <transition to="c" name="to c"/>
  </fork>
</process-definition>

```

```

</fork>
<state name="b">
  <transition to="syncmerge"/>
</state>
<state name="c">
  <transition to="syncmerge"/>
</state>
<join name="syncmerge">
  <transition to="end"/>
</join>
<end-state name="end"/>
</process-definition>

```

```

transitionNames = new ArrayList();
if ( scenario == 1 ) {
  transitionNames.add( "to b" );
} else if ( scenario == 2 ) {
  transitionNames.add( "to c" );
} else if ( scenario >= 3 ) {
  transitionNames.add( "to b" );
  transitionNames.add( "to c" );
}

```

No OpenWFE a condição de execução da ramificação é definida diretamente na linguagem de definição de processos. Já no jBPM, é necessário que haja programação por parte do projetista do *workflow*. Isto pode ser negativo no caso do analista não ter conhecimento em programação Java. Neste caso o analista deverá ser, necessariamente, um programador, ou deve aprender a programar na referida linguagem, para realizar as definições de processos que utilizam este padrão.

## 5.7. Padrão 07 (Synchronizing Merge)

O padrão *synchronizing merge* é suportado pelo OpenWFE indiretamente, através de uma combinação dos padrões *simple merge* (padrão 05) aninhados em uma *synchronization* (padrão 03). Este padrão é entendido como um fim para o padrão *multi-choice* (padrão 06).

No jBPM, o padrão *synchronization merge* é suportado utilizando-se *AND-joins*. O exemplo é o mesmo do padrão *multi-choice* (padrão 06), portanto quem dita quais são as transições a serem sincronizadas são as regras para transições do *multi-choice*.

```

<process-definition>
  <start-state name="start">
    <transition to="a"/>
  </start-state>
  <state name="a">
    <transition to="multichoice"/>
  </state>

```

```

<fork name="multichoice">
  <transition to="b" name="to b" />
  <transition to="c" name="to c" />
</fork>
<state name="b">
  <transition to="syncmerge" />
</state>
<state name="c">
  <transition to="syncmerge" />
</state>
<join name="syncmerge">
  <transition to="end" />
</join>
<end-state name="end" />
</process-definition>

```

Em ambas as ferramentas, é através de um *multi-choice* que as transições a serem sincronizadas partem. A implementação deste padrão no OpenWFE e no jBPM é equivalente, apesar da diferença nas linguagens de definição de processo.

## 5.8. Padrão 08 (Multi-merge)

Um exemplo de como o padrão *multi-merge* é implementado no OpenWFE é mostrado a seguir.

```

<process-definition name="padrao8">
  <sequence>
    <participant ref="a" />
    <concurrency>
      <if>
        <equals
          field-value="assunto"
          other-value="aprovação crédito"
        />
        <sequence>
          <participant ref="b" />
          <subprocess ref="d" />
        </sequence>
      </if>
      <if>
        <equals
          field-value="categoria"
          other-value="new plant"
        />
        <sequence>
          <participant ref="c" />
          <subprocess ref="d" />
        </sequence>
      </if>
    </concurrency>
  </sequence>

  <process-definition name="d">

```

```

    <participant ref="d" />
  </process-definition>
</process-definition>

```

Da mesma maneira que o padrão *multi-choice* (padrão 06), no jBPM o padrão *multi-merge* pode ser especificado, pelo projetista do *workflow*, através da especificação das condições desejadas para cada transição. A seguir é mostrado um exemplo de como o padrão *multi-merge* é definido em jPDL. Logo após, é mostrado um exemplo de como o projetista do *workflow* especifica as condições das transições.

```

<process-definition>
  <start-state name="start">
    <transition to="a"/>
  </start-state>
  <state name="a">
    <transition to="multichoice"/>
  </state>
  <fork name="multichoice">
    <transition to="b" name="to b"/>
    <transition to="c" name="to c"/>
  </fork>
  <state name="b">
    <transition to="multimerge"/>
  </state>
  <state name="c">
    <transition to="multimerge"/>
  </state>
  <merge name="multimerge">
    <transition to="d"/>
  </merge>
  <state name="d"/>
</process-definition>

```

```

transitionNames = new ArrayList();
if ( scenario == 1 ) {
  transitionNames.add( \"to b\" );
} else if ( scenario == 2 ) {
  transitionNames.add( \"to c\" );
} else if ( scenario >= 3 ) {
  transitionNames.add( \"to b\" );
  transitionNames.add( \"to c\" );
}

```

O OpenWFE e o jBPM implementam este padrão de maneira semelhante, definindo as condições de fusão para cada transição. A diferença está em como cada uma das ferramentas determina estas condições. No OpenWFE as condições são definidas diretamente na linguagem de definição de processos. Já no jBPM, elas são definidas usando-se a linguagem de programação Java. Como visto anteriormente, quando isso ocorre há a necessidade de que o projetista de *workflow* conheça a linguagem Java para que as condições da fusão sejam definidas.

## 5.9. Padrão 09 (Discriminator)

No OpenWFE o atributo *sync*, da marcação *concurrency*, quando é definido como “*generic*”, é usado para implementar o padrão *discriminator*. A seguir é mostrado um trecho de código com um exemplo de como o padrão *discriminator* é especificado.

```
<concurrency
  sync="generic"
  count="1"
  merge-type="mix"
  remaining="forget"
>
  <participant ref="alfred" />
  <participant ref="boris" />
  <participant ref="carlos" />
</concurrency>
```

No exemplo, o atributo *remaining* é definido para “*forget*”, o que significa que os dois participantes que responderam muito tarde não irão receber uma notificação de cancelamento. Eles serão capazes de responder, mas isto não terá outros efeitos no fluxo.

jBPM suporta este padrão de maneira direta, fornecendo uma construção *discriminator*. Na definição do processo, o *join* que fará o papel de *discriminator* deve ser definido como tal, através do método `setDiscriminator(true)`, da classe *Join*. Um exemplo da definição do padrão *discriminator* é mostrado no trecho de código a seguir.

```
ProcessDefinition pd = createProcessDefinition();

// configure the join as a discriminator
Join join = (Join) pd.getNode("discriminator");
join.setDiscriminator(true);
```

Sendo que o a definição de processo resultante de `createProcessDefinition()` é:

```
<process-definition>
  <start-state name="start">
    <transition to="a"/>
  </start-state>
  <state name="a">
    <transition to="multichoice"/>
  </state>
  <fork name="multichoice">
    <transition to="b" name="to b"/>
    <transition to="c" name="to c"/>
```

```

</fork>
<state name="b">
  <transition to="discriminator"/>
</state>
<state name="c">
  <transition to="discriminator"/>
</state>
<join name="discriminator">
  <transition to="d"/>
</join>
<state name="d"/>
</process-definition>

```

```

transitionNames = new ArrayList();
if ( scenario == 1 ) {
  transitionNames.add( "to b" );
} else if ( scenario == 2 ) {
  transitionNames.add( "to c" );
} else if ( scenario >= 3 ) {
  transitionNames.add( "to b" );
  transitionNames.add( "to c" );
}

```

## 5.10. Padrão 10 (Arbitrary Cycles)

O OpenWFE suporta ciclos arbitrários. O código a seguir mostra uma implementação do padrão *arbitrary cycles* na linguagem de definição de processos do OpenWFE.

```

<process-definition name="pattern10" revision="1.5.1pre2" >
  <sequence>
    <participant ref="a" />

    <set variable="participant_list" value="b, c, d" />
    <set variable="//position" value="0" />
    <set field="agreed" value="true" type="boolean" />

    <subprocess ref="node" />

    <participant ref="e" />
  </sequence>

  <process-definition name="node">
    <sequence>
      <set variable="node_participant"
        value="${call:elt('${position}', '${participant_list}')}" />
      <if>
        <not>
          <equals variable-value="node_participant"
            other-value="" />
        </not>
        <!-- then -->

```

```

<sequence>
  <participant ref="{node_participant}" />
  <if>
    <equals field-value="agreed" other-value="false" />
    <!-- then -->
    <inc variable="//position" value="-1" />
    <!-- else -->
    <inc variable="//position" value="1" />
  </if>
  <if>
    <lesser-than variable-value="//position"
      other-value="0" />
    <!-- then enforce low limit at 0 -->
    <set variable="//position" value="0" />
  </if>
  <subprocess ref="node" />
</sequence>
</if>
</sequence>
</process-definition>
</process-definition>

```

O jBPM suporta o padrão *arbitrary cycles* da seguinte maneira: o processo é definido como no exemplo de código abaixo, e o *token* raiz envia um comando *signal* para indicar a próxima transição do fluxo. É usado `root.signal("forward")` para acionar a transição que avança no fluxo, e `root.signal("back")` para acionar a transição que volta, realizando o laço.

```

<process-definition>
  <start-state name="start">
    <transition to="a"/>
  </start-state>
  <state name="a">
    <transition to="b"/>
  </state>
  <state name="b">
    <transition to="c"/>
  </state>
  <state name="c">
    <transition to="d" name="forward"/>
    <transition to="b" name="back"/>
  </state>
  <state name="d">
    <transition to="e" name="forward"/>
    <transition to="c" name="back"/>
  </state>
  <state name="e"/>
</process-definition>

```

### 5.11. Padrão 11 (Implicit Termination)



O OpenWFE suporta o padrão *implicit termination* diretamente. Um processo de negócio é concluído quando não há mais nada a ser feito. No OpenWFE não existem as expressões *break* ou *exit*.

O jBPM permite que se verifique se a instância de processo foi finalizada. O método `setTerminationImplicit(true)`, da classe `ProcessDefinition`, em conjunto com o método `hasEnded()`, da classe `ProcessInstance` permitem esta verificação.

O problema da implementação deste padrão no jBPM é que o processo pode ficar pendente se não houver o uso do método `setTerminationImplicit(true)`. No OpenWFE esta questão não preocupa, uma vez que o processo que deve ser terminado através de terminação explícita não fica pendente, e sim é considerado finalizado.

## 5.12. Padrão 12 (MI Without Synchronization)

O padrão *multiple instances without synchronization* é tratado no OpenWFE como uma geração de novas *threads* de controle. O exemplo a seguir mostra como este padrão pode ser especificado.

```
<process-definition name="padrao12">
  <sequence>
    <participant ref="a" />
    <subprocess ref="b" />
    <participant ref="c" />
  </sequence>

  <process-definition name="b">
    <sequence>
      <participant ref="b" />
      <if>
        <equals field-value="spawn" other-value="true" />
        <subprocess ref="x" forget="true" />
      </if>
      <if>
        <equals field-value="loop" other-value="true" />
        <subprocess ref="b" />
      </if>
    </sequence>
  </process-definition>

  <process-definition name="x">
    <participant ref="x" />
  </process-definition>
</process-definition>
```

No exemplo, “b” é um sub-processo, com uma chave para criação de instâncias (*spawn*) e uma chave para iteração (*loop*). Na chave *spawn*, se seu valor for *true*, o sub-processo “x” é iniciado, e neste sub-processo a instância é criada. A chave *loop* controla a iteração. Se seu valor for *true*, o sub-processo “b” é iniciado, recursivamente. *Loop* será *true* enquanto houverem instâncias a serem criadas.

Este padrão é implementado no jBPM como uma combinação de tarefas não sincronizadas e avaliação das tarefas em tempo de execução. A especificação do padrão *multiple instances without synchronization* na jPDL, se dá através da marcação *action*, aninhada à marcação *event*, por sua vez aninhada à marcação *node*, que indica o nodo onde ocorrerá a geração das tarefas. O trecho de código a seguir mostra um exemplo da especificação deste padrão, em jPDL.

#### Geração das Instâncias:

```
public static class CreateTasks implements ActionHandler {
    private static final long serialVersionUID = 1L;
    public void execute(ExecutionContext executionContext) {
        TaskMgmtDefinition taskMgmtDefinition = (TaskMgmtDefinition)
            executionContext.getDefinition(TaskMgmtDefinition.class);
        Task task = taskMgmtDefinition.getTask("undress");

        TaskMgmtInstance tmi = executionContext.getTaskMgmtInstance();
        for (int i = 1; i < nbrOfTasks + 1; i++) {
            tmi.createTaskInstance(task, executionContext.getToken());
        }
    }
}
```

#### Definição do Processo:

```
<process-definition>
  <start-state name='a'>
    <transition to='b' />
  </start-state>
  <node name='b'>
    <event type='node-enter'>
      <action class='package.CreateTasks' />
    </event>
    <transition to='c' />
  </node>
  <state name='c' />
  <task name='undress' />
</process-definition>
```

Na implementação deste padrão, vemos novamente no jBPM a necessidade da programação em Java, para a criação das instâncias, enquanto no OpenWFE a

implementação da geração de múltiplas instâncias é feita na própria linguagem de definição de processos. Isto é uma vantagem para o OpenWFE, pois não é necessário que o projetista de *workflow* aprenda a linguagem Java.

### 5.13. Padrão 13 (MI With a Priori Design Time Knowledge)

No OpenWFE, o padrão *multiple instances with a priori design time knowledge* pode ser especificado de duas maneiras. A primeira é utilizando o padrão *parallel split* (padrão 02). Quando todas as atividades forem completadas, basta utilizar uma construção de sincronização. O trecho de código a seguir ilustra a opção de uso do *parallel split*.

```
<concurrency>
  <participant ref="first-activity" />
  <participant ref="first-activity" />
  <participant ref="first-activity" />
</concurrency>
```

A outra maneira é utilizar a marcação *concurrent-iterator*, da linguagem de definição de processos do OpenWFE. O exemplo a seguir apresenta esta opção.

```
<process-definition name="padrao13">
  <sequence>
    <participant ref="a" />
    <participant ref="b" />
    <concurrent-iterator
      on-value="1, 2, 3"
      to-field="index"
    >
      <participant ref="x" />
    </concurrent-iterator>
    <participant ref="c" />
  </sequence>
</process-definition>
```

O jBPM suporta diretamente o padrão *multiple instances with a priori design time knowledge*, através da marcação *task-node*, em jPDL, que permite que sejam definidas as tarefas, e para qual transição o fluxo segue, depois de cumpridas as tarefas. Aninhadas à marcação *task-node*, são especificadas as tarefas, através da marcação *task*, e a próxima transição do fluxo, através da marcação *transition*. A seguir é mostrado um exemplo da definição deste padrão, na jPDL.

```
<process-definition name='rotina diaria? '>
  <start-state name='a'>
    <transition to='b' />
```

```

</start-state>
<state name='b'>
  <transition to='t' />
</state>
<task-node name='t'>
  <task name='acordar' />
  <task name='trabalhar' />
  <task name='dormir' />
  <transition to='c' />
</task-node>
<state name='c' />
</process-definition>

```

Em ambas as ferramentas este padrão é suportado de maneira simples e direta. No OpenWFE há a possibilidade de se utilizar duas implementações, dependendo da necessidade e da complexidade do projeto de *workflow*. No jBPM também é clara e objetiva a implementação deste padrão.

#### 5.14. Padrão 14 (MI With a Priori Runtime Knowledge)

No OpenWFE, o padrão *multiple instances with a priori runtime knowledge* é implementado através da marcação *concurrent-iterator*, onde pode-se definir um campo chamado *iterator\_list*, cujo conteúdo pode ser um atributo do tipo *String*, como por exemplo 'a, b, c, d'. Quando o *concurrent-iterator* é avaliado, o valor deste campo é transformado em uma lista de iteração. Quatro *workitems* serão passados para o próximo participante, com o campo *\_\_subject\_\_* definido respectivamente como 'a', 'b', 'c', e 'd'. O separador padrão é ',' (vírgula). O trecho de código a seguir mostra o uso de *concurrent-iterator*.

```

<sequence>
  <participant ref="alice" />
  <concurrent-iterator
    on-field-value="iteration_list"
    to-field="__subject__"
  >
    <participant ref="bob" />
  </concurrent-iterator>
</sequence>

```

No jBPM, o padrão *multiple instances with a priori runtime knowledge* é especificado, em jPDL, através da marcação *task-node*. Um *task-node* seria um nodo onde várias instâncias de tarefas podem ser criadas. Quando todas as tarefas criadas em tempo de execução forem finalizadas, a transição para o próximo estado é disparada. O trecho de código a seguir mostra um exemplo da especificação deste padrão, em jPDL.

### Geração das Instâncias:

```
public static class CreateTasks implements ActionHandler {
    private static final long serialVersionUID = 1L;
    public void execute(ExecutionContext executionContext) {
        TaskMgmtDefinition taskMgmtDefinition = (TaskMgmtDefinition)
            executionContext.getDefinition(TaskMgmtDefinition.class);
        Task task = taskMgmtDefinition.getTask("ver filme");

        TaskMgmtInstance tmi = executionContext.getTaskMgmtInstance();
        for (int i = 1; i < nbrOfTasks + 1; i++) {
            tmi.createTaskInstance(task, executionContext.getToken());
        }
    }
}
```

### Definição do Processo:

```
<process-definition>
  <start-state name='a'>
    <transition to='b' />
  </start-state>
  <state name='b'>
    <transition to='t' />
  </state>
  <task-node name='t' create-tasks='false'>
    <event type='node-enter'>
      <action class='package.CreateTasks' />
    </event>
    <task name='ver filme' />
    <transition to='c' />
  </task-node>
  <state name='c' />
</process-definition>
```

### Método para finalizar uma instância de tarefa no *task-node*:

```
public static void endOneTask(Token token) {
    TaskMgmtInstance tmi =
        (TaskMgmtInstance)token.getProcessInstance().getInstance(TaskMgmtInsta
            nce.class);
    TaskInstance taskInstance = (TaskInstance)
        tmi.getUnfinishedTasks(token).iterator().next();
    taskInstance.end();
}
```

O padrão *multiple instances with a priori runtime knowledge* é suportado no OpenWFE usando somente sua linguagem de definição de processos. Mais uma vez o jBPM apresenta a necessidade de implementação de regras em Java, agora não somente para criação de instâncias, mas também ára finalizar uma instância de tarefa no *task-node*.

### 5.15. Padrão 15 (MI Without a Priori Runtime Knowledge)

Como o OpenWFE suporta sub-processos, ele implementa facilmente o padrão *multiple instances without a priori runtime knowledge*. Um exemplo de como o OpenWFE implementa este padrão é mostrado a seguir.

```
<process-definition name="multiple instances">
  <subprocess ref="launch" />

  <process-definition name="launch">
    <subprocess ref="activity" forget="true" />
    <participant ref="b" />
    <if>
      <equals
        field-value="fire_more_instances_of_activity"
        other-value="true"
      />
      <subprocess ref="launch" />
    </if>
  </process-definition>

  <process-definition name="activity">
    <participant ref="a" />
  </process-definition>
</process-definition>
```

No jBPM, a implementação do padrão *multiple instances without a priori runtime knowledge* é praticamente a mesma do padrão *multiple instances with a priori runtime knowledge*, exceto pelo atributo *signal*, com o valor “*last-wait*”, na marcação *task-node*. Neste padrão é possível que se criem instâncias de tarefas e as finalize. Quando todas as tarefas forem finalizadas, a transição para o próximo estado é disparada. O trecho de código a seguir mostra um exemplo da especificação deste padrão, em jPDL.

Geração das Instâncias:

```
public static class CreateTasks implements ActionHandler {
  private static final long serialVersionUID = 1L;
  public void execute(ExecutionContext executionContext) {
    TaskMgmtDefinition taskMgmtDefinition = (TaskMgmtDefinition)
      executionContext.getDefinition(TaskMgmtDefinition.class);
    Task task = taskMgmtDefinition.getTask("ver filme");

    TaskMgmtInstance tmi = executionContext.getTaskMgmtInstance();
    for (int i = 1; i < nbrOfTasks + 1; i++) {
      tmi.createTaskInstance(task, executionContext.getToken());
    }
  }
}
```

```

    }
}

```

Definição do Processo:

```

<process-definition>
  <start-state name='a'>
    <transition to='b' />
  </start-state>
  <state name='b'>
    <transition to='t' />
  </state>
  <task-node name='t' create-tasks='false'>
    <event type='node-enter'>
      <action class='package.CreateTasks' />
    </event>
    <task name='ver filme' />
    <transition to='c' />
  </task-node>
  <state name='c' />
</process-definition>

```

Método para adicionar uma instância de tarefa no task-node:

```

private void addOneTask(Token token) {
    TaskMgmtDefinition tmd = (TaskMgmtDefinition)
        processDefinition.getDefinition(TaskMgmtDefinition.class);
    Task task = tmd.getTask("watch movie amadeus");

    TaskMgmtInstance tmi =
        token.getProcessInstance().getTaskMgmtInstance();
    tmi.createTaskInstance(task, token);
}

```

Método para finalizar uma instância de tarefa no task-node:

```

public static void endOneTask(Token token) {
    TaskMgmtInstance tmi =
        (TaskMgmtInstance)token.getProcessInstance().getInstance(TaskMgmtInsta
        nce.class);
    TaskInstance taskInstance = (TaskInstance)
        tmi.getUnfinishedTasks(token).iterator().next();
    taskInstance.end();
}

```

## 5.16. Padrão 16 (Deferred Choice)

Na linguagem de definição de processos do OpenWFE, a marcação *concurrency* pode possuir o atributo *remaining*. Se o valor deste atributo for *cancel*, quando uma das atividades for executada, as todas as outras são canceladas. A seguir é apresentado um

trecho de código com a definição de um exemplo de implementação do padrão *deferred choice*.

```
<concurrency
  sync="generic"
  count="1"
  remaining="cancel"
>
  <participant ref="uma-atividade" />
  <participant ref="outra-atividade" />
</concurrency>
```

O jBPM suporta o padrão *deferred choice* uma vez que é possível que se indique qual a ramificação deve ser seguida, através do método `signal()` da classe `Token`. Um exemplo de especificação do padrão *deferred choice* é mostrado a seguir.

```
<process-definition>
  <start-state name="inicio">
    <transition to="a"/>
  </start-state>
  <state name="a">
    <transition to="b" name="primeira"/>
    <transition to="c" name="segunda"/>
  </state>
  <state name="b"/>
  <state name="c"/>
</process-definition>
```

Escolha da ramificação a ser tomada:

```
ProcessInstance pi = new ProcessInstance( pd );
Token root = pi.getRootToken();
...
root.signal("primeira"); // toma a transição de nome 'primeira'
```

A implementação deste padrão no OpenWFE permite que a ramificação a ser seguida simplesmente seja executada, e a partir daí o fluxo segue por ela. No jBPM, há um controle maior, pois é necessário que se mande um sinal para que a transição da ramificação seja executada.

## 5.17. Padrão 17 (Interleaved Parallel Routing)

O roteamento paralelo pode ser realizado através de mais de uma maneira no OpenWFE. Serão apresentadas duas delas. A primeira é apresentada no trecho de código a seguir.

```
<concurrency>
  <when>
```



```

<undefined variable-value="/mutex" />
<sequence>
  <set variable="/mutex" value="set" />
  <participant ref="physical-tester" />
  <unset variable="/mutex" />
</sequence>
</when>
<when>
  <undefined variable-value="/mutex" />
  <sequence>
    <set variable="/mutex" value="set" />
    <participant ref="medical-tester" />
    <unset variable="/mutex" />
  </sequence>
</when>
</concurrency>

```

A condição preferencial é colocada no topo da lista de concorrência. Este fragmento se aproveita do recurso ‘/’ variável. Esta barra indica que a variável precisa ser definida na raiz da definição do fluxo, sendo deste modo disponível para cada expressão abaixo dela. Outra definição é mostrada no trecho de código a seguir.

```

<sequence>
  <participant ref="a" />
  <iterator
    on-value="{call:shuffle('b, c, d')}"
    to-variable="p"
  >
    <participant ref="{p}" />
  </iterator>
  <participant ref="e" />
</sequence>

```

O exemplo utiliza a função *shuffle*, que reordena randomicamente uma lista, no exemplo, ‘b, c, d’. A decisão nesta implementação é deixada por conta da *engine* de fluxo, que decide (randomicamente) qual participante recebe o *workitem*.

O roteamento paralelo intercalado é diretamente suportado pelo jBPM, através das marcações *interleave-start* e *interleave-end*, na jPDL. Um exemplo da definição do padrão *interleaved parallel routing* é mostrado a seguir.

```

<process-definition>
  <start-state name='a'>
    <transition to='startinterleaving' />
  </start-state>
  <interleave-start name='startinterleaving'>
    <transition name='b' to='b' />
    <transition name='c' to='c' />
    <transition name='d' to='d' />
  </interleave-start>
  <state name='b'>
    <transition to='endinterleaving' />

```

```

</state>
<state name='c'>
  <transition to='endinterleaving' />
</state>
<state name='d'>
  <transition to='endinterleaving' />
</state>
<interleave-end name='endinterleaving'>
  <transition name='back' to='startinterleaving' />
  <transition name='done' to='e' />
</interleave-end>
<state name='e' />
</process-definition>

```

O padrão *interleaved parallel routing* é implementado de forma mais clara pelo jBPM do que pelo OpenWFE. O jBPM leva vantagem por possuir marcações específicas para o suporte a este padrão, no caso *interleave-start* e *interleave-end*. As implementações do padrão *interleaved parallel routing* no OpenWFE são alternativas interessantes para a resolução da questão de roteamento intercalado.

### 5.18. Padrão 18 (Milestone)

No OpenWFE, o padrão *milestone* é implementado através da utilização do padrão *deferred coice* (Padrão 16). O código a seguir mostra um exemplo da definição deste padrão.

```

<process-definition name="milestone">
  <sequence>
    <participant ref="b" />
    <set variable="/m" value="true" />
    <concurrency
      sync="generic"
      count="1"
      remaining="cancel"
    >
      <subprocess ref="a" />
      <sequence>
        <participant ref="c" />
        <set variable="/m" value="false" />
      </sequence>
    </concurrency>
  </sequence>

  <process-definition name="a">
    <if>
      <equals variable-value="/m" other-value="true" />
      <sequence>
        <participant ref="a" />
        <subprocess ref="a" />
      </sequence>
    </if>
  </process-definition>
</process-definition>

```

```
</process-definition>
```

No exemplo, recursividade é usada para implementar o padrão. Enquanto o participante “b” não tratar seu item de trabalho, não atingindo assim o marco (*milestone*), o participante “a” trabalha repetidamente em seu item de trabalho. A sincronização através do padrão *deferred choice* na concorrência entre “a” e “c” garante que tão logo que “c” responda, a ramificação de “a” será cancelada.

O jBPM possui uma marcação *milestone-node*, para o tratamento do padrão *milestone*. Um exemplo da especificação deste padrão é mostrado a seguir.

```
<process-definition>
  <start-state name="start">
    <transition to="fork"/>
  </start-state>
  <fork name="fork">
    <transition to="b" name="m"/>
    <transition to="d" name="d"/>
  </fork>
  <state name="b">
    <transition to="m"/>
  </state>
  <milestone-node name="m">
    <transition to="c"/>
  </milestone-node>
  <state name="c">
    <transition to="join"/>
  </state>
  <state name="d">
    <transition to="join"/>
    <event type="node-leave">
      <action/>
    </event>
  </state>
  <join name="join">
    <transition to="end"/>
  </join>
  <end-state name="end"/>
</process-definition>
```

Da mesma maneira que no padrão anterior, o jBPM possui uma marcação específica, em jPDL, para o suporte ao padrão *milestone*, o que torna direta sua implementação. O OpenWFE utiliza a primeira implementação proposta na definição do padrão *milestone*, visto no capítulo 3.

## 5.19. Padrão 19 (Cancel Activity)

O padrão *cancel activity* pode ser executado no OpenWFE de duas maneiras. A primeira é através do uso de uma construção como a de *deferred choice*, onde um item de trabalho voltando de uma ramificação cancela as outras ramificações. A outra maneira é através do uso de uma ferramenta de controle. No OpenWFE, esta ferramenta é executada na linha de comando. Uma vez autenticado como administrador pode-se usar comandos como “*list*”, “*freeze*”, “*unfreeze*”, “*cancel*” e “*cancelx*”.

No jBPM, o padrão *cancel activity* é suportado através do método `end()`, da classe `Token`. Primeiramente, pega-se o estado que se quer cancelar e o atribui a um *token*, através da chamada de método `root.getChild(nomeDoEstado)`, sendo que *root* é o *token* do nodo raiz da instância de processo. Posteriormente, chama-se o método `token.end()`, que irá cancelar a atividade. Um exemplo é mostrado a seguir.

```
<process-definition>
  <start-state name="start">
    <transition to="f1"/>
  </start-state>
  <fork name="f1">
    <transition to="a" name="a"/>
    <transition to="f2" name="f2"/>
  </fork>
  <state name="a"/>
  <fork name="f2">
    <transition to="b" name="b"/>
    <transition to="c" name="c"/>
  </fork>
  <state name="b"/>
  <state name="c"/>
</process-definition>
```

Trecho de código do cancelamento de atividade:

```
ProcessInstance pi = new ProcessInstance(processDefinition);
Token root = pi.getRootToken();
Token tokenA = root.getChild("a");
tokenA.end();
```

No OpenWFE, por utilizar o padrão *deferred choice* para o cancelamento de atividade, é necessário que haja uma transição para uma “*shadow activity*”. No jBPM, não é necessário que hajam transições para que o cancelamento seja feito, pois a atividade é cancelada diretamente. O uso de uma ferramenta para cancelamento de atividade, como é o caso do OpenWFE, é interessante quando se quer cancelar uma atividade sem que haja interação com o *workflow*. Isso pode ocorrer num ponto onde o

*workflow* está emperrado ou quando uma atividade não será executada, e o administrador, externo ao processo, constata que a atividade deverá ser cancelada.

### **5.20. Padrão 20 (*cancel Case*)**

No OpenWFE, o padrão *cancel case* é implementado como o comando “*cancel*” da ferramenta de controle descrita no padrão *cancel activity* (padrão 19). Este comando de cancelamento remove uma instância de fluxo inteira. Como está implementado hoje, ela não afeta fluxos filhos e fluxos pais.

O padrão *cancel case*, no jBPM, é implementado de forma semelhante ao padrão *cancel activity* (padrão 19), exceto que ao invés de se cancelar um estado, cancela-se toda uma instância de processo. Todas as atividades da instância de processo são canceladas automaticamente. O trecho de código a seguir mostra um exemplo de implementação deste padrão.

```
ProcessInstance pi = new ProcessInstance(processDefinition);  
pi.end();
```

Como no padrão *cancel activity*, através apenas da chamada de método o cancelamento é executado. Ao contrário disso, o OpenWFE não possui construções que cancelem instâncias de processo em sua linguagem de definição de processos. A desvantagem disso é que não há como cancelar uma instância inteira durante a execução de um processo, sendo necessária a intervenção do administrador do *workflow*.

## 6. Conclusão

Neste trabalho foi realizado um estudo dos padrões de *workflow* e sua aplicação à reutilização de software nas corporações. A adoção do conceito de padrões é uma das abordagens mais recentes em Engenharia de Software e de extrema utilidade quando aplicada a *Workflows*.

Os padrões de workflow efetivamente servem como guia e auxiliam no desenvolvimento de *Workflow Management Systems* e Sistemas de Gerenciamento de Processos de Negócio, fornecendo estratégias concretas de como os processos de negócio podem ser modelados em um sistema computacional. Torna-se muito mais fácil desenvolver e avaliar sistemas complexos, como os de gerenciamento de *workflows* tendo como base estes padrões.

Pesquisando-se sobre padrões de *workflow*, foi constatado que eles são amplamente usados como base para desenvolvimento e/ou avaliação de WFMS.

No que diz respeito ao trabalho desenvolvido, o mesmo permitiu avaliar uma gama importante de ferramentas para o desenvolvimento de *workflows*, sendo que as que foram estudadas com mais profundidade foram jBPM e OpenWFE.

A razão que nos levou à escolha destas duas ferramentas foi o fato de as mesmas anunciarem o suporte a todo o conjunto de padrões de *workflow*. Este aspecto foi comprovado pelo estudo realizado.

As duas ferramentas apresentam um grau de flexibilidade bastante alto, podendo ser aplicados no desenvolvimento de sistemas baseados na tecnologia Java.

Do ponto de vista de aplicações corporativas, a ferramenta jBPM, por fazer uso do container JBoss, pode ser mais facilmente adaptada ao desenvolvimento de sistemas que sejam baseadas nesta tecnologia.

No que diz respeito ao suporte aos padrões de *workflow*, existem muitos casos em que as duas ferramentas implementam de forma semelhante os padrões. Em outros casos, as peculiaridades de cada ferramenta sobressaem, pela adoção de estratégias que melhor se adaptem às suas características.

As duas ferramentas permitem a utilização de estratégias alternativas de implementação, com base no conhecimento adquirido na outra ferramenta, o que pode ser útil para cobrir eventuais limitações observadas na ferramenta em uso.

Com base nos resultados obtidos, consideramos ter atingido o principal objetivo deste trabalho. Entretanto, à medida que os estudos foram evoluindo ao longo deste período, novas idéias foram surgindo as quais, pelas limitações de tempo impostas pelo cronograma do projeto realizado, não puderam ser completamente desenvolvidas. Sendo assim, deixamos estas idéias como sugestões para futuros trabalhos envolvendo este tema.

Os tópicos propostos neste contexto são os seguintes:

- Modelagem de um *framework* para desenvolvimento de *workflows* baseado em padrões de *workflow*;
- Implementação do *framework* proposto;
- Aplicação efetiva dos padrões de *workflow* nas ferramentas;
- Avaliação de outras ferramentas, propondo alternativas para os padrões não suportados por elas, formando uma documentação sobre ferramentas, baseada nos padrões de *workflow*;
- Proposta de implementações alternativas para os padrões nas ferramentas;
- Identificação de novos padrões, baseado num conhecimento mais avançado de processos de negócio e *workflow*;
- Identificação de outras alternativas de implementação, além das propostas.

## Referências Bibliográficas

[AALST 02a] AALST, Wil van der.; HOFSTEDE, Arthur ter.; KIEPUSZEWSKI, Bartek; BARROS, Alistair. Workflow Patterns. 2002a. Disponível por <http://tmitwww.tn.tue.nl/research/patterns/download/wfs-pat-2002.pdf>

[AALST 02b] AALST, Wil van der.; HEE, Kees M. Workflow Management: Models, Methods and Systems. EUA, The MIT Press, 2002b.

[AALST 03] AALST, Wil van der.; HOFSTEDE, A.; KIEPUSZEWSKI, B.; BARROS, A. Advanced Workflow Patterns. 2003. Disponível por <http://tmitwww.tn.tue.nl/research/patterns/download/coopis.pdf>

[AMARAL 97] AMARAL, Vinícius L. Técnicas de Modelagem de Workflow. Porto Alegre: CPGCC da UFRGS, 1997. (TI – 622).

[APACHE 05] Apache Software Foundation. Disponível por <http://ant.apache.org/manual/index.html>. Capturado em Abril de 2005.

[BONITA 05] Bonita: Workflow Cooperative System. Disponível por <http://bonita.objectweb.org/>. Capturado em Abril de 2005.

[COOPER 98] COOPER, James W. The Design Patterns Java Companion. EUA, Addison-Wesley, 1998.

[COSA 05] COSA GmbH. Disponível por <http://eng.cosa.de/>. Capturado em Abril de 2005.

[DAVENPORT 94] DAVENPORT, T. Reengenharia de Processos. 5 ed. Rio de Janeiro, Campus.1994.

[FISCHER 02] FISCHER, Layna. Workflow Handbook 2002. EUA, Future Strategies Inc., 2002.



[FUJITSU 05] Fujitsu Limited. Disponível por <http://www.fujitsu.com/global/services/software/interstage/products/bpm/>. Capturado em Abril de 2005.

[GAMMA 95] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. Design Patterns: Elements of Reusable Object Oriented Software. EUA, Addison-Wesley, 1995.

[GEORGAKOPOULOS 97] GEORGAKOPOULOS, D. TSALGATIDOU, A. “Technology and Tools for Comprehensive Business Process Lifecycle Management” In: Proceedings of the NATO Advanced Study Institute on Workflow. Vol. 164. pp. 356-395. Istanbul, Turkey. August. 1997.

[JBOSS 05] JBoss Inc. Disponível por <http://www.jboss.com>. Capturado em Abril de 2005.

[MEDINA-MORA 92] MEDINA-MORA, R.; WINOGRAD, T., FLORES, R.; FLORES, F. The action workflow approach to workflow management technology, Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'92), ACM Press, Toronto, Ontario, 1992.

[OPENFLOW 05] OpenFlow. Disponível por <http://www.openflow.it/EN/>. Capturado em Abril de 2005.

[OPENSYPHONY 05] OpenSymphony. Disponível por <http://www.opensymphony.com/osworkflow/>. Capturado em Abril de 2005.

[OPENWFE 05] OpenWFE. Disponível por <http://web.openwfe.org/display/openwfe/Home>. Capturado em Abril de 2005.

[SCHEIDT 03] SCHEIDT, Neiva. Introdução de Suporte Gerencial Baseado em Workflow e CMM a um Ambiente de Desenvolvimento de Software. Florianópolis,

2003. 129 p. Dissertação (Mestrado em Ciências da Computação) - Universidade Federal de Santa Catarina.

[SILVA 01] SILVA, André Valadares. Modelagem de Processos para Implementação de Workflow: uma avaliação crítica. Rio de Janeiro, 2001 XII. 405 p. (COPPE/UF RJ. M.Sc., Engenharia de Produção, 2001) Tese - Universidade Federal do Rio de Janeiro, COPPE.

[THOM 01] THOM, Lucinéia Heloisa. Associando Estrutura Organizacional e Modelagem de Workflow. Porto Alegre, 2001. 46p. Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2001.

[TIBCO 05] TIBCO Software Inc. Disponível por <http://www.tibco.com/>

[WHITE 04] WHITE, Stephen A. Process Modeling Notations and Workflow Patterns. EUA, 2004. Disponível por [http://www.omg.org/bp-corner/bp-files/Process\\_Modeling\\_Notations.pdf](http://www.omg.org/bp-corner/bp-files/Process_Modeling_Notations.pdf)

[WfMC 95] WORKFLOW MANAGEMENT COALITION. The Workflow Reference Model. 1995. Disponível por <http://www.wfmc.org/standards/docs/tc003v11.pdf>. Capturado em Setembro de 2004.

[WfMC 99] WORKFLOW MANAGEMENT COALITION. Workflow Management Coalition Terminology & Glossary. 1999. Disponível por [http://www.wfmc.org/standards/docs/TC-1011\\_term\\_glossary\\_v3.pdf](http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf). Capturado em Maio de 2005.

[WFMOPEN 05] WfMOpen. Disponível por <http://wfmopen.sourceforge.net/>. Capturado em Abril de 2005.

## Anexo 1 – Artigo

# UM ESTUDO SOBRE PADRÕES DE WORKFLOW E SUAS IMPLEMENTAÇÕES EM WORKFLOW MANAGEMENT SYSTEMS

Fábio Dall'Oglio da Cunha

fabio@inf.ufsc.br

Universidade Federal de Santa Catarina

### RESUMO

*Este trabalho de conclusão de curso teve por objetivo estudar Padrões de Workflow e analisar como estes padrões são implementados em Workflow Management Systems. Para tanto, desenvolveu-se um estudo específico sobre workflow, processos de negócio, padrões de projeto, padrões de workflow e workflow management systems. Realizou-se um aprofundamento em padrões de workflow, onde foram descritos e estudados em detalhes todos os padrões propostos para resolverem problemas em processos de negócio em sistemas computacionais. Fez-se ainda um estudo sobre workflow management systems, onde foram analisadas as principais ferramentas de gerenciamento de workflow, tanto comerciais quanto de código aberto, disponíveis no mercado. A partir deste estudo, identificaram-se ferramentas de workflow que implementam todos os padrões propostos. Por fim, elaborou-se uma descrição e análise detalhada de como os padrões de workflow são implementados nestas ferramentas.*

### Palavras-Chave:

Processos de Negócio, Workflow, Padrões de Projeto, Padrões de Workflow, Workflow Management Systems, WFMS, jBPM, OpenWFE

### 1. INTRODUÇÃO

Estamos numa era onde a competitividade entre empresas nunca esteve tão acirrada. Empresas tradicionais têm modificado radicalmente sua maneira de atuar no mercado, utilizando-se de estratégias de marketing, reduzindo o ciclo de vida de seus principais produtos, aumentando a velocidade de produção em suas fábricas, geralmente através da reconfiguração de seus processos de fabricação, e se esforçando cada vez

mais para atender da melhor maneira possível às necessidades de seus clientes. [6]

Segundo Davenport, um dos motivadores para estas transformações é a Tecnologia da Informação (TI) [3]. A razão disso é que a tecnologia da informação tem dado grandes saltos evolutivos nos últimos anos, resultando na criação de maneiras completamente novas de organizar processos de negócio. O desenvolvimento de pacotes genéricos de software para gerenciar processos de negócio, no caso os *Workflow Management Systems* (WFMS), é particularmente importante no que diz respeito a isto [1]

Segundo Amaral, *Workflow* pode ser definido como uma coleção de tarefas organizadas para realizar um processo de negócio. Uma tarefa pode ser executada por um sistema de computador, por um agente humano ou pela combinação destes dois [2]. Sistemas de *Workflow* não substituem as já tradicionais ferramentas de automação de escritório, como processadores de texto e planilhas eletrônicas, nem os sistemas corporativos da empresa, como sistemas de contas a pagar, contabilidade, folha de pagamento, planejamento e controle de produção etc. Ao invés disso permitem uma utilização integrada dessas diversas ferramentas para a execução otimizada de todo o processo.

Este trabalho tem por objetivo identificar e estudar *Workflow Management Systems* (WFMS) disponíveis no mercado, principalmente os de código aberto, que implementem os Padrões de Workflow. Pretende-se verificar como estes WFMS implementam os Padrões de Workflow propostos e se os mesmos dão suporte adequado ao desenvolvimento de *Workflow Management Systems* próprios.

Os objetivos específicos deste projeto foram:

- Estudo de tópicos relacionados a processos de negócios em empresas.
- Estudo aprofundado da tecnologia Workflow, Workflow Management Systems, Padrões de Projeto e Padrões de

Workflow, que será peça fundamental na execução deste projeto.

- Pesquisa de trabalhos envolvendo os tópicos citados anteriormente, que possam servir de base à proposta a ser desenvolvida no contexto deste projeto;
- Elaboração de um estudo aprofundado das ferramentas de workflow de código aberto que suportam os padrões de workflow, a resultar deste trabalho.

## 2. WORKFLOW E CONCEITOS RELACIONADOS

A seguir, as definições das tecnologias que serão utilizadas no projeto.

### 2.1. Processo de Negócio

Processos de negócio são coleções de uma ou mais atividades ligadas que concretizam um objetivo de negócio ou uma diretriz a ser alcançada, como, por exemplo, o fechamento de um contrato de negócio, e/ou a satisfação de uma necessidade específica de um cliente [4].

Processos de negócio são descrições das atividades de uma organização, focadas no mercado. Isto é, processos de negócio são compilações de atividades que dão suporte às funções organizacionais críticas, ao realizar um objetivo ou diretriz a ser alcançada [5].

O ciclo de vida de um processo de negócio envolve tudo que diz respeito à captura do processo em uma representação computadorizada para automatizar o mesmo, como, por exemplo, implementando um processo através de Workflow. Isto tipicamente inclui atividades explícitas de medição, análise, e melhoria do processo. A necessidade de gerenciar efetivamente o ciclo de vida do processo de negócio, como, por exemplo, aplicando Business Process Management (BPM), tem conduzido ao desenvolvimento de novas conceitos ferramentas interoperáveis que suportam aspectos complementares de gerenciamento de processos de negócio.

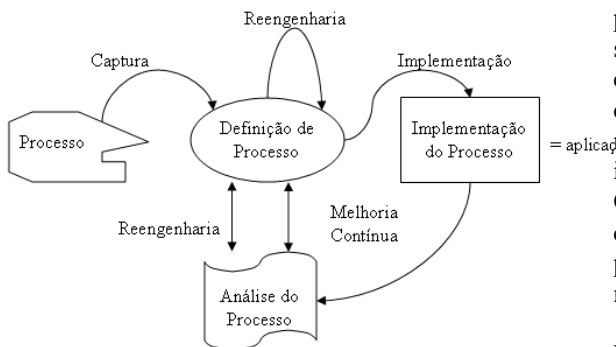


Figura 1: Ciclo de Vida dos Processos de Negócio

### 2.2. Workflow

Segundo a Workflow Management Coalition, Workflow é a facilitação ou automação de um processo de negócio, como um todo ou em parte. Workflow é preocupado com a automação de procedimentos onde documentos, informações ou tarefas são passados entre participantes do processo, de acordo com um conjunto de regras que se queiram alcançar, ou contribuir para, um objetivo de negócio como um todo [7].

Workflow Management System (WFMS) é um sistema para definição, criação e gerência da execução de fluxos de trabalho através do uso de software, capaz de interpretar a definição de processos, interagir com seus participantes e, quando necessário, invocar ferramentas e aplicações [WfMC95].

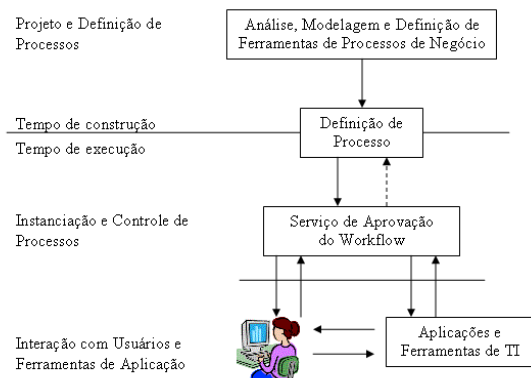


Figura 2: Características de Sistemas Workflow

## 3. PADRÕES DE WORKFLOW

O trabalho de pesquisa de Wil van der Aalst, Arthur ter Hofstede, Bartek Kiepuszewski, e Alistair Barros resultou na identificação de 21 padrões que descrevem o comportamento de processos de negócio. A base lógica para o desenvolvimento dos padrões foi descrever as possíveis capacidades que um Workflow deve ter durante a execução de processos de negócios [8].

Estes padrões, chamados Padrões de Workflow [1], podem ser usados para examinar WFMS aos quais se pretende utilizar ou podem servir como um conjunto de idéias de como implementar requisitos de Workflow em um sistema próprio que já esteja rodando na empresa ou que se pretenda implementar.

Os padrões variam de muito simples a muito complexos e cobrem os comportamentos que podem ser capturados dentro da maioria dos modelos de processo de negócio [8].

Eles estão divididos nas seguintes categorias:

- Padrões de Controle Básico: Sequence, Parallel Split, Synchronization, Exclusive Choice e Simple Merge.

- Padrões de Ramificação e Sincronização Avançados: Multiple Choice, Synchronizing Merge, Multiple Merge, Discriminator e N-out-of-M-join.
- Padrões Estruturais: Arbitrary Cycles e Implicit Termination.
- Padrões Envolvendo Múltiplas Instâncias: MI without synchronization, MI with a priori known design time knowledge, MI with a priori known runtime knowledge e MI with no a priori runtime knowledge.
- Padrões Baseados em Estados: Deferred Coice, Interleaved Parallel Routing e Milestone.
- Padrões de Cancelamento: Cancel Activity e Cancel Case.

Um Padrão de Workflow básico possui os seguintes elementos, além do nome:

1. Uma descrição, que explica o funcionamento do padrão.
2. Uma lista de sinônimos, que contém a lista de outros nomes pelos quais o padrão também é conhecido.
3. Alguns exemplos, que exemplifica situações, através da descrição de situações hipotéticas ou metáforas, para que se possa visualizar o que se quer atingir com o padrão.

Em alguns casos de padrões mais complexos, podem aparecer os seguintes elementos:

4. O problema, que descreve porque a construção é difícil de ser implementada em muitos WFMS disponíveis hoje em dia.
5. Possíveis estratégias de implementação, que também é referenciado como solução, e descreve como, assumindo um conjunto de primitivas, o procedimento requerido pode ser realizado.

No projeto, foram apresentados e descritos os 21 padrões, conforme as descrições acima.

#### 4. WORKFLOW MANAGEMENT SYSTEMS

Com o objetivo de identificar quais WFMS que implementam padrões de *workflow* estão disponíveis no mercado, e quais padrões são implementados por tais sistemas, realizou-se uma pesquisa com este foco.

As ferramentas estão divididas em duas categorias: WFMS de Código Aberto, que estão disponíveis sob licença de software livre, e WFMS Comerciais, cujas licenças são necessariamente adquiridas comercialmente.

O foco deste trabalho é a identificação de padrões de *workflow* em ferramentas de código aberto. As ferramentas pesquisadas são: JBoss jBPM, OpenWFE, OpenSymphony Workflow, WfMOpen,

OpenFlow e ObjectWeb Bonita. As informações a respeito de WFMS de código aberto foram coletadas dos *websites* dos projetos, documentações, contato com desenvolvedores e das próprias aplicações, principalmente através dos casos de teste de unidade.

As informações de WFMS comerciais foram coletadas de [1] e dos *websites* dos produtos.

Verificou-se que duas ferramentas implementam todos os padrões de *workflow*: o JBoss jBPM e o OpenWFE, e a maneira com as quais estas ferramentas implementam os padrões foi estudada a fundo.

#### 5. PADRÕES DE WORKFLOW NO JBPM E OPENWFE

No capítulo anterior realizou-se um levantamento dos WFMS disponíveis. Identificou-se, com esta pesquisa, que duas ferramentas se propõem a implementar todos os padrões de *workflow*: o JBoss jBPM e o OpenWFE.

Como dito no capítulo anterior, estudou-se mais a fundo as ferramentas, identificando a maneira com que os padrões são implementados nelas. Como visto no capítulo 3, um dos elementos que compõem um padrão de *workflow* são as sugestões de implementação. Realizou-se uma análise das implementações das ferramentas, traçou-se um paralelo e foi criticada a implementação, com base nas sugestões registradas nos padrões.

#### 6. CONCLUSÃO

Neste trabalho foi realizado um estudo dos padrões de *workflow* e sua aplicação à reutilização de software nas corporações. A adoção do conceito de padrões é uma das abordagens mais recentes em Engenharia de Software e de extrema utilidade quando aplicada a *Workflows*.

Os padrões de *workflow* efetivamente servem como guia e auxiliam no desenvolvimento de *Workflow Management Systems* e Sistemas de Gerenciamento de Processos de Negócio, fornecendo estratégias concretas de como os processos de negócio podem ser modelados em um sistema computacional. Torna-se muito mais fácil desenvolver e avaliar sistemas complexos, como os de gerenciamento de *workflows* tendo como base estes padrões.

Pesquisando-se sobre padrões de *workflow*, foi constatado que eles são amplamente usados como base para desenvolvimento e/ou avaliação de WFMS.

No que diz respeito ao trabalho desenvolvido, o mesmo permitiu avaliar uma gama importante de ferramentas para o desenvolvimento de *workflows*, sendo que as que foram estudadas com mais profundidade foram jBPM e OpenWFE.

A razão que nos levou à escolha destas duas ferramentas foi o fato de as mesmas anunciarem o

suporte a todo o conjunto de padrões de *workflow*. Este aspecto foi comprovado pelo estudo realizado. As duas ferramentas apresentam um grau de flexibilidade bastante alto, podendo ser aplicados no desenvolvimento de sistemas baseados na tecnologia Java.

Do ponto de vista de aplicações corporativas, a ferramenta jBPM, por fazer uso do container JBoss, pode ser mais facilmente adaptada ao desenvolvimento de sistemas que sejam baseadas nesta tecnologia.

No que diz respeito ao suporte aos padrões de *workflow*, existem muitos casos em que as duas ferramentas implementam de forma semelhante os padrões. Em outros casos, as peculiaridades de cada ferramenta sobressaem, pela adoção de estratégias que melhor se adaptem às suas características.

As duas ferramentas permitem a utilização de estratégias alternativas de implementação, com base no conhecimento adquirido na outra ferramenta, o que pode ser útil para cobrir eventuais limitações observadas na ferramenta em uso.

## 7. TRABALHOS FUTUROS

Com base nos resultados obtidos, consideramos ter atingido o principal objetivo deste trabalho. Entretanto, à medida que os estudos foram evoluindo ao longo deste período, novas idéias foram surgindo as quais, pelas limitações de tempo impostas pelo cronograma do projeto realizado, não puderam ser completamente desenvolvidas. Sendo assim, deixamos estas idéias como sugestões para futuros trabalhos envolvendo este tema.

Os tópicos propostos neste contexto são os seguintes:

- o Modelagem de um framework para desenvolvimento de workflows baseado em padrões de workflow;
- o Implementação do framework proposto;
- o Aplicação efetiva dos padrões de workflow nas ferramentas;
- o Avaliação de outras ferramentas, propondo alternativas para os padrões não suportados por elas, formando uma documentação sobre ferramentas, baseada nos padrões de workflow;
- o Proposta de implementações alternativas para os padrões nas ferramentas;
- o Identificação de novos padrões, baseado num conhecimento mais avançado de processos de negócio e Workflow;
- o Identificação de outras alternativas de implementação, além das propostas.

## 8. REFERÊNCIAS BIBLIOGRÁFICAS

[1] AALST, Wil van der.; HOFSTEDE, Arthur ter.; KIEPUSZEWSKI, Bartek; BARROS, Alistair. Workflow Patterns. 2002a. Disponível por

<http://tmitwww.tn.tue.nl/research/patterns/download/wfs-pat-2002.pdf>

[2] AMARAL, Vinícius L. Técnicas de Modelagem de Workflow. Porto Alegre: CPGCC da UFRGS, 1997. (TI – 622).

[3] DAVENPORT, T. Reengenharia de Processos. 5 ed. Rio de Janeiro, Campus.1994.

[4] GEORGAKOPOULOS, D. TSALGATIDOU, A. “Technology and Tools for Comprehensive Business Process Lifecycle Management” In: Proceedings of the NATO Advanced Study Institute on Workflow. Vol. 164. pp. 356-395. Istanbul, Turkey. August. 1997.

[5] MEDINA-MORA, R.; WINOGRAD, T., FLORES, R.; FLORES, F. The action workflow approach to workflow management technology, Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW’92), ACM Press, Toronto, Ontario, 1992.

[6] SILVA, André Valadares. Modelagem de Processos para Implementação de Workflow: uma avaliação crítica. Rio de Janeiro, 2001 XII. 405 p. (COPPE/UFRJ. M.Sc., Engenharia de Produção, 2001) Tese - Universidade Federal do Rio de Janeiro, COPPE.

[7] WORKFLOW MANAGEMENT COALITION. The Workflow Reference Model. 1995. Disponível por <http://www.wfmc.org/standards/docs/tc003v11.pdf>. Capturado em Setembro de 2004.

[8] WHITE, Stephen A. Process Modeling Notations and Workflow Patterns. EUA, 2004. Disponível por [http://www.omg.org/bp-corner/bp-files/Process\\_Modeling\\_Notations.pdf](http://www.omg.org/bp-corner/bp-files/Process_Modeling_Notations.pdf)