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

Evolução de Esquemas de *Workflow* Representados em XML

por

FÁBIO ZSCHORNACK

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

Profa. Dra. Nina Edelweiss
Orientadora

Porto Alegre, abril de 2003.

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Zschornack, Fábio

Evolução de Esquemas de *Workflow* Representados em XML / por Fábio Zschornack. — Porto Alegre: PPGC da UFRGS, 2003.

91 f.: il.

Dissertação (mestrado) — Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2003. Orientadora: Edelweiss, Nina.

1. Workflow. 2. Evolução de workflow. 3. XML. I. Edelweiss, Nina. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitora: Prof^{a}. Wrana Maria Panizzi
Pró-Reitor de Ensino: Prof. José Carlos Ferraz Hennemann
Pró-Reitora Adjunta de Pós-Graduação: Prof^{a}. Jocélia Grazia
Diretor do Instituto de Informática: Prof. Philippe Olivier Alexandre Navaux
Coordenador do PPGC: Prof. Carlos Alberto Heuser
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

"The more that things change,
the more they stay the same."
— Rush (Circumstances, 1978)

Agradecimentos

Primeiramente, agradeço a Deus por tudo: pelo azul celeste e por nuvens que há também, pelas rosas no caminho e os espinhos que elas têm, pela escuridão da noite e pela estrela que brilhou, pelas coisas do futuro e por tudo que já passou, pela certeza da vida que virá[1]! Nunca conseguirei agradecer o suficiente pelo que fizeste por mim!

— Agora, aos humanos —

Um agradecimento especial à professora Nina Edelweiss, pela paciência e pelo bom-humor de sempre e por ter me acolhido como orientando num momento difícil do meu mestrado. Suas palavras de incentivo e coragem foram decisivas para que eu pudesse chegar até aqui. Mais do que uma orientadora, uma grande amiga! Obrigado, professora!

À minha esposa, Denise, pelo apoio dado mesmo sem perceber e por ter me agüentado durante esses dois últimos anos. A tua presença ao meu lado me deixa mais confiante e alegre para poder seguir em frente. “(...) Perdi as chaves (que cabeça a minha!), agora vai ter que ser para toda a vida!”[2] Te amo!

A meus pais, Lidio e Elmira, que mesmo longe, estão sempre perto, em pensamento e oração, torcendo pelo meu sucesso! Aos meus irmãos, Felipe, Tiago e Talita, pelos momentos alegres e pela compreensão!

Quero também agradecer a duas pessoas que muito me ajudaram neste mestrado: Ademir e Patrícia. Obrigado pelo incentivo, pelo sofá-cama e pelas sessões de música em discos de vinil, do velho e bom rock’n’roll!

Aos amigos do Instituto de Informática, meu muito obrigado! Não teria condições de citar todos, mas de maneira especial agradeço ao grupo Versões e Tempo, pelas trocas de idéias, em especial à Renata Galante. Também ao pessoal dos laboratórios 213 e 215, pelas risadas e pela conversa fora! Aos funcionários, pela presteza e amizade!

A todas as pessoas que me incentivaram ou me repreenderam durante essa caminhada, fica também o meu agradecimento sincero!

A CAPES, pela bolsa de estudos fornecida.

— E por que não ao *software*? —

Finalmente, gostaria de fazer um agradecimento especial ao LaTeX e ao BibTeX, por realizarem para mim a tarefa de formatação deste trabalho e “por não possuirem vírus de macro”.

[1]adaptado do Hinário Luterano, nº 222.

[2]Engenheiros do Hawaii (3x4, 1999)

Sumário

Lista de Abreviaturas 7
Lista de Figuras 8
Lista de Tabelas 10
Resumo 11
Abstract 12
1 Introdução 13
1.1 Motivação 13
1.2 Objetivos do Trabalho 14
1.3 Estrutura do Trabalho 14
2 Base Conceitual 15
2.1 Conceitos sobre *Workflow* 15
2.1.1 Conceitos Básicos 15
2.1.2 Padrões da WfMC 16
2.2 Evolução de *Workflow* 17
2.2.1 Proposta de Casati et al. 17
2.2.2 O Ambiente ADEPT 19
2.2.3 Proposta de Kradolfer 20
2.2.4 Considerações sobre Evolução de *Workflow* 21
2.3 *Workflow* e XML 22
2.3.1 XPDL 22
2.3.2 Wf-XML 22
2.3.3 WSFL 23
2.3.4 XRL 24
2.3.5 WQM 25
2.3.6 Considerações sobre *Workflow* e XML 26
2.4 Considerações Finais 28
3 Representação e Alteração de Esquemas de *Workflow* em XML 30
3.1 Linguagem para Representação de *Workflow* em XML 30
3.1.1 Motivação 30
3.1.2 *Workflows* Estruturados 31
3.1.3 Elementos da Linguagem 33
3.1.4 Exemplo de Modelagem 37
3.1.5 Considerações sobre a Linguagem de Representação de *Workflow* 37
3.2 Alteração de Esquemas de *Workflow* em XML 39
3.2.1 Propostas de Linguagens para Alteração de Documentos XML 39
3.2.2 Operações de Modificação 41
3.2.3 Considerações sobre Alteração de Esquemas 44
3.3 Considerações Finais 46

4 Representação e Migração de Instâncias 47
4.1 Representação das Instâncias 47
4.1.1 Estrutura Básica 48
4.1.2 Estados das Atividades 49
4.1.3 Informações sobre os Esquemas de Base 51
4.1.4 Exemplo de Instância 51
4.2 Migração de Instâncias para outro Esquema 52
4.2.1 Compatibilidade Total 53
4.2.2 Compatibilidade Parcial 54
4.2.3 Migração de Instâncias Compatíveis 55
4.2.4 Adaptação de Instâncias Incompatíveis 57
4.3 Considerações Finais 58

5 Versionamento de Esquemas de *Workflow* 59
5.1 Conceitos 59
5.2 Estrutura do Esquema Versionado 60
5.3 Estados das Versões de Esquema de *Workflow* 61
5.4 Análise Temporal das Versões de Esquema 64
5.5 Operações sobre Esquemas e Versões 66
5.5.1 Operação de Criação de Esquema 66
5.5.2 Operação de Derivação de Versões 66
5.5.3 Operações de Mudança de Estados 67
5.6 Considerações Finais 69

6 Estudo de Caso 70
6.1 Descrição do Contexto 70
6.2 Modelagem do Estudo de Caso 70
6.3 Considerações Finais 81

7 Conclusão 84

Anexo XML *Schema* da Linguagem de Representação de *Workflow* 86

Referências 88

Lista de Abreviaturas

ADEPT	Application Development Based on Encapsulated Premodeled Process Templates
DOM	Document Object Model
DTD	Document Type Definition
RET	Representação Explícita de Transições
RIT	Representação Implícita de Transições
W3C	World Wide Web Consortium
WfMC	Workflow Management Coalition
WFMS	Workflow Management System
WQM	Workflow Query Model
WSFL	Web Services Flow Language
XML	Extensible Markup Language
XRL	Exchangeable Routing Language
XPDL	XML Process Definition Language

Lista de Figuras

FIGURA 2.1 – Modelo de referência da WfMC, adaptado de Workflow Management Coalition (1995) 17
FIGURA 2.2 – Separação das instâncias do esquema e adequação das mesmas quanto às políticas progressivas (CASATI et al., 1998) 19
FIGURA 2.3 – Árvore de versões do esquema de *workflow* v (adaptado de Kradolfer e Geppert (1999)) 20
FIGURA 2.4 – Intercâmbio entre diferentes ferramentas de modelagem de *workflow* (adaptado de Workflow Management Coalition (2002)) 23
FIGURA 2.5 – Parte de um documento XPDL 24
FIGURA 2.6 – Exemplo de uma mensagem em Wf-XML 24
FIGURA 2.7 – Exemplo de um serviço *Web* em WSFL 25
FIGURA 2.8 – Exemplo de um processo em XRL 26
FIGURA 2.9 – Exemplo de um processo WQM em XML Query Algebra 26
FIGURA 2.10 – Representação de *workflow* em XML com transições explícitas (b) e implícitas (c), a partir de um modelo gráfico (a) 27
FIGURA 2.11 – Fluxo que não pode ser modelado pela abordagem RIT 28

FIGURA 3.1 – Estruturas definidas por Kiepuszewski, Hofstede e Bussler (2000) . 32
FIGURA 3.2 – Estrutura básica da linguagem de representação de *workflow* 33
FIGURA 3.3 – Elemento `sequence` 34
FIGURA 3.4 – Elemento `parallel` 35
FIGURA 3.5 – Elemento `conditional` 36
FIGURA 3.6 – Elemento `loop` 36
FIGURA 3.7 – Representação de múltipla escolha (AALST et al., 2002) 37
FIGURA 3.8 – Exemplo de um fluxo de controle representado na linguagem proposta 38
FIGURA 3.9 – Documento de exemplo (a) e após uma modificação (b) 40
FIGURA 3.10 – Operação necessária para modificação de acordo com a extensão de XQuery 40
FIGURA 3.11 – Operação necessária para modificação de acordo com a linguagem XUpdate 40
FIGURA 3.12 – Operação de inserção 42
FIGURA 3.13 – Operação de remoção 43
FIGURA 3.14 – Operação de movimentação 44
FIGURA 3.15 – Operação de troca de estrutura 44
FIGURA 3.16 – Facilidade de alteração de esquema de *workflow* de acordo com a linguagem de representação 45

FIGURA 4.1 – Representação do elemento `loop` em um esquema (a) e em uma instância (b) 49
FIGURA 4.2 – Diagrama de estados das atividades 50
FIGURA 4.3 – Exemplo de uma instância 52
FIGURA 4.4 – Árvores do esquema e da instância 56
FIGURA 4.5 – Instância migrada para outro esquema 57

FIGURA 5.1 – Esquema versionado de *workflow* em árvore 61
FIGURA 5.2 – Esquema versionado de *workflow* em XML 62

FIGURA 5.3 – Diagrama dos estados possíveis para uma versão de esquema de *workflow* 63
FIGURA 5.4 – Gráfico do ciclo de vida de versões em função do tempo 65
FIGURA 5.5 – Representação dos estados na versão de esquema 65

FIGURA 6.1 – Fluxo de controle do processo analisado 71
FIGURA 6.2 – Esquema de *workflow* de montagem de computadores (`PC1.xml`) . 72
FIGURA 6.3 – Modificações aplicadas sobre a versão PC1 (`mod1.xml`) 73
FIGURA 6.4 – Versão derivada de PC1 (`PC2.xml`) 74
FIGURA 6.5 – Modificações aplicadas sobre a versão PC2 (`mod2.xml`) 76
FIGURA 6.6 – Versão derivada de PC2 (`PC3.xml`) 77
FIGURA 6.7 – Modificações aplicadas sobre a versão PC2 (`mod3.xml`) 78
FIGURA 6.8 – Versão derivada de PC2 (`PC4.xml`) 79
FIGURA 6.9 – Instância baseada em PC2 81
FIGURA 6.10 – Árvores da versão PC3 e da instância 189 82
FIGURA 6.11 – Esquema versionado PC 82

Lista de Tabelas

TABELA 5.1 – Ações possíveis sobre os estados de uma versão de esquema de *workflow* 64
TABELA 5.2 – Estados que aceitam operações sobre versões 68
TABELA 5.3 – Aplicação das operações sobre esquemas e versões de esquema . . 68

Resumo

Sistemas de gerência de *workflow* estão sendo amplamente utilizados para a modelagem e a execução dos processos de negócios das organizações. Tipicamente, esses sistemas interpretam um *workflow* e atribuem atividades a participantes, os quais podem utilizar ferramentas e aplicativos para sua realização. Recentemente, XML começou a ser utilizada como linguagem para representação dos processos bem como para a interoperação entre várias máquinas de *workflow*.

Os processos de negócio são, na grande maioria, dinâmicos, podendo ser modificados devido a inúmeros fatores, que vão desde a correção de erros até a adaptação a novas leis externas à organização. Conseqüentemente, os *workflows* correspondentes devem também evoluir, para se adequar às novas especificações do processo. Algumas propostas para o tratamento deste problema já foram definidas, enfocando principalmente as alterações sobre o fluxo de controle. Entretanto, para *workflows* representados em XML, ainda não foram definidos mecanismos apropriados para que a evolução possa ser realizada.

Este trabalho apresenta uma estratégia para a evolução de esquemas de *workflow* representados em XML. Esta estratégia é construída a partir do conceito de versionamento, que permite o armazenamento de diversas versões de esquemas e a consulta ao histórico de versões e instâncias. As versões são representadas de acordo com uma linguagem que considera os aspectos de evolução. As instâncias, responsáveis pelas execuções particulares das versões, também são adequadamente modeladas. Além disso, é definido um método para a migração de instâncias entre versões de esquema, no caso de uma evolução. A aplicabilidade da estratégia proposta é verificada por meio de um estudo de caso.

Palavras-chave: Workflow, evolução de workflow, XML.

TITLE: "EVOLUTION OF XML WORKFLOW SCHEMATA"

Abstract

Workflow management systems are being widely used by organizations for modeling and enactment of their business processes. Typically, these systems recognize a workflow description and assign its activities to workflow participants, based on the control flow. The participants may use tools and applications during the workflow execution. In this context, XML arose as a language for process modeling as well as for interoperation among workflow engines.

Business processes are mostly dynamic. They must be modified due to many reasons, as errors correction or adaptation to new external laws. The corresponding workflows must also evolve in order to reflect the new process specifications. Some proposals concerning this question were already made, most of them focusing on changes on the control flow. However, there is not a suitable mechanism to deal with the evolution of XML workflows.

This work presents an evolution strategy for XML workflow schemata. The strategy is based on schema versioning concepts allowing the storage of versions as well as queries over histories of both versions and instances. A language that considers evolution issues is proposed in order to model workflow schema versions. Instances, as particular executions of schema versions, are also suitably modeled. In addition, a method for instance migration between versions is defined, in case of an evolution. Finally, a study case verifies the appliance of the whole strategy.

Keywords: workflow, workflow evolution, XML.

1 Introdução

1.1 Motivação

A cada dia, as organizações buscam melhorar seus procedimentos de negócio, visando maior qualidade dos produtos e satisfação de seu público-alvo. Para que tais objetivos possam ser alcançados, buscam agilidade na realização das tarefas, uma vez que, num mundo de intensa competição, isso pode determinar o sucesso ou o fracasso da empresa.

As empresas utilizam, atualmente, sistemas de informação para uma melhor tomada de decisão sobre seus negócios. No passado os processos se caracterizavam por ser tipicamente manuais e a dificuldade de coordenação dos mesmos era proporcional à sua complexidade, ou seja, quanto mais tarefas e pessoas envolvidas em um processo, mais difícil sua eficiente coordenação. Com o advento dos computadores, alguns dos processos manuais puderam ser automatizados, resultando em uma diminuição do tempo necessário para que uma determinada tarefa fosse realizada. Apesar disso, o problema da coordenação de tarefas ainda permanecia. Uma primeira tentativa foi passar aos programas aplicativos a tarefa de coordenação, o que causava um aumento da sua complexidade.

Nesse contexto, surgiu a necessidade de um sistema que pudesse gerenciar a realização das tarefas, tanto manuais quanto automatizadas, e também coordenar as mesmas, buscando uma maior eficiência do processo. Assim, começaram a nascer os sistemas de gerência de *workflow*, específicos para essas finalidades, isolando-as das aplicações. Um *workflow* pode ser considerado como uma implementação de um processo de negócio, de tal forma que suas atividades possam ser descritas e organizadas. Um *workflow* (também chamado de esquema de *workflow*) é executado por um sistema de *workflow*, criando instâncias específicas a cada caso de execução.

Existem vários formatos e linguagens para a representação de *workflow*, tais como Redes de Petri (AALST, 1998), textuais (WORKFLOW MANAGEMENT COALITION, 1999a) e outros mais específicos, baseados em elementos de fluxogramas (CASATI et al., 1995; SADIQ; ORLOWSKA, 1996). Há pouco tempo, começou-se a utilizar também a linguagem XML (*Extensible Markup Language*) para a modelagem de *workflows*, por ser uma linguagem flexível e também porque esta linguagem é muito utilizada no intercâmbio de informações entre organizações. Isso possibilitou o surgimento de uma nova forma de negócios: o comércio eletrônico. Os processos de comércio eletrônico geralmente são inter-organizacionais, ou seja, ultrapassam a fronteira de uma empresa em particular, envolvendo várias delas, de forma que informações devem ser passadas entre as mesmas através de um formato adequado. Neste ponto a linguagem XML é útil, servindo como formato de interoperabilidade (AALST; KUMAR, 2000; KUMAR; ZHAO, 2002).

Seja no âmbito de uma empresa ou entre diversas organizações, os processos estão em constante mudança, visando a um melhor resultado para o usuário final. Muitos são os motivos que causam alterações nos processos. Entre eles podem ser citados: novos requisitos externos, melhoria de desempenho, entre outros. Por isso, é necessário que os *workflows* possam também se adequar às mudanças ocorridas, ou seja, o sistema que gerencia o *workflow* deve repassar àqueles as mudanças ocorridas nos processos. Esse fenômeno é chamado de evolução de *workflow*, sendo caracterizado por operações que são aplicadas sobre uma representação para deixá-la compatível com a nova realidade.

Vários trabalhos foram propostos para evolução a partir da utilização de métodos ou formas de representação específicos, como por exemplo Casati et al. (1998), Reichert e

Dadam (1998), Kradolfer (2000), entre outros. No entanto, para *workflows* representados em XML, ainda não se tem um mecanismo adequado para tratar sua evolução.

1.2 Objetivos do Trabalho

O objetivo deste trabalho consiste em definir um mecanismo de representação e evolução de *workflows* em XML, de forma a contemplar características de versionamento e representação temporal. O versionamento é importante para o armazenamento de toda a história de evolução de um determinado esquema de *workflow*. Na parte de representação temporal são tratados aspectos como tempos de vida de esquemas, instâncias e atividades.

Para que o objetivo descrito seja alcançado, as seguintes tarefas são desenvolvidas neste trabalho:

- é definida uma linguagem para representação de *workflow* em XML, permitindo que os esquemas criados a partir dela sejam facilmente alterados e que obedecem a critérios de correção;
- é apresentada uma linguagem adequada para a modificação de esquemas de *workflow* definidos de acordo com a linguagem proposta;
- é definido um mecanismo para armazenamento de versões de esquema;
- é proposto um método para migração de instâncias entre versões de esquemas.

1.3 Estrutura do Trabalho

Esta dissertação contém 7 capítulos, incluído o presente, estando organizada da seguinte maneira:

- o capítulo 2 traz uma revisão de conceitos e tecnologias, necessária para apresentar o estado da arte nas áreas de evolução de *workflow* e representação em XML;
- no capítulo 3, é apresentada uma proposta de linguagem para representação de *workflow* em XML, definida de forma a permitir uma evolução de esquemas mais facilitada. Além disso, é mostrada uma linguagem para a modificação de documentos, possibilitando a criação de novos esquemas;
- o capítulo 4 descreve a forma de representação de instâncias de acordo com a linguagem descrita no capítulo 3, bem como apresenta uma estratégia para migração de instâncias entre dois esquemas;
- no capítulo 5, é apresentado um modelo para versionamento de esquemas de *workflow*, em XML, permitindo que várias versões possam estar ativas ao mesmo tempo e que todas sejam armazenadas para fins históricos;
- no capítulo 6, é apresentado e modelado um estudo de caso, para verificação da aplicabilidade das estratégias de evolução de esquemas em XML;
- por fim, o capítulo 7 estabelece algumas conclusões e apresenta propostas de trabalhos futuros que podem complementar este trabalho.

2 Base Conceitual

Neste capítulo é feita uma apresentação de conceitos e propostas das tecnologias que servirão de base para o modelo de evolução de *workflows* que será apresentado a partir do capítulo 3. A seção 2.1 trata de conceitos gerais da tecnologia de *workflow* e padrões que estão sendo desenvolvidos. A seção 2.2 apresenta algumas propostas para o problema da evolução de *workflow*. A seção 2.3 mostra algumas linguagens e modelos encontrados na literatura, que integram a tecnologia de *workflow* e XML. Por fim, a seção 2.4 traz algumas considerações sobre os conceitos e tecnologias discutidos neste capítulo.

2.1 Conceitos sobre *Workflow*

Nesta seção são abordados alguns conceitos relacionados à tecnologia de *workflow* e de processos de negócio, para que possam ser usados no restante do trabalho.

2.1.1 Conceitos Básicos

Segundo Workflow Management Coalition (1999b), ***workflow*** é a automação, total ou parcial, de um processo de negócio, durante o qual documentos, informações ou tarefas são passados de um participante para outro para a realização de alguma ação, de acordo com um conjunto de regras procedimentais. Georgakopoulos, Hornick e Sheth (1995) definem *workflow* como sendo uma coleção de tarefas organizada de maneira a executar processos de negócio. Um **processo de negócio**, por sua vez, pode ser entendido como um conjunto de uma ou mais atividades relacionadas, que coletivamente atingem um objetivo de negócios, dentro do contexto de uma estrutura organizacional que define papéis funcionais e relações. Tais atividades podem ser humanas, como reuniões e entrevistas, ou automatizadas, como a impressão de um documento.

De maneira simplificada, pode-se entender um *workflow* como sendo uma representação computacional de um processo de negócio. Essa representação é descrita por meio de alguma simbologia, basicamente constituída por elementos gráficos ou textuais. A forma gráfica de representação de processos de negócio é a mais compreensível, pois permite que as atividades que compõem um processo sejam dispostas e arranjadas visualmente. Às atividades são relacionados papéis, que são conjuntos de participantes responsáveis pela sua execução parcial ou total.

Para que um *workflow* possa ser definido e executado, com participantes realizando as atividades, tem-se um **sistema de gerência de *workflow*** (WFMS - *Workflow Management System*) que, conforme Workflow Management Coalition (1995; 1999b), é um sistema de *software* que define, cria e gerencia a execução de *workflows*, executando em uma ou mais máquinas de *workflow*, que é capaz de interpretar a definição do processo, interagir com os participantes do *workflow* e, quando necessário, invocar o uso de ferramentas e aplicações de tecnologia de informação.

Segundo Georgakopoulos, Hornick e Sheth (1995), a gerência de *workflow* envolve desde a modelagem dos processos até a sincronização das atividades e dos participantes que realizam os processos. São destacadas as seguintes etapas na gerência de *workflow*:

1. **modelagem do processo e especificação do *workflow*** – são requeridos modelos de *workflow* e metodologias para se capturar um processo como uma especificação de *workflow*;

2. **reengenharia do processo** – são necessárias metodologias para a otimização do processo modelado;

3. **implementação e automação do *workflow*** – metodologias e tecnologias para a utilização de sistemas de informação e usuários que implementem, executem e controlem as tarefas descritas na especificação do *workflow*.

Barthelmess e Wainer (1995) destacam a distinção entre dois níveis em um WFMS: o **nível de descrição** e o **nível de execução**. No nível de descrição, obtém-se a representação do processo de negócio. Essa representação tipicamente é chamada de **esquema de *workflow***, que é composto pelas atividades e pela ordem de execução das mesmas, pelos papéis responsáveis e pelo conjunto de informações necessárias. Por outro lado, no nível de execução, este esquema gera **instâncias** que são ativadas pelo WFMS e representam casos em particular que obedecem à estrutura do esquema. Idealmente, deseja-se que os dois níveis sejam ativados separadamente para um mesmo processo de negócio. No entanto, isso pode não acontecer em alguns casos, pelo fato de, em tempo de modelagem, ser impossível de se prever como se comporta o processo.

É também apresentado em Georgakopoulos, Hornick e Sheth (1995) uma categorização dos *workflows*, levando-se em consideração principalmente a estruturação e o nível de automação dos processos. Dessa maneira, existem três categorias de *workflow*:

- ***workflows ad-hoc*** – são processos que possuem baixa estruturação e nos quais a coordenação das atividades é feita manualmente, muitas vezes em tempo de execução. Por isso, não se caracterizam por serem processos repetitivos e geralmente não são críticos para o sucesso da organização;

- ***workflows* administrativos** – envolvem um pouco mais de estruturação. São processos mais previsíveis e que suportam automação. Porém, são de tarefas razoavelmente simples, que não necessitam de complexos sistemas de informação;

- ***workflows* de produção** – são processos que envolvem alta estruturação, por tratarem-se de processos repetitivos e que são essenciais para a organização. Um WFMS é extremamente importante para este tipo de *workflow*, visto que geralmente existem muitas atividades e vários agentes que devem ser acionados.

2.1.2 Padrões da WfMC

A WfMC (*Workflow Management Coalition*) é uma organização internacional formada em 1993, que congrega desenvolvedores, usuários, universidades e grupos de pesquisa, cujo principal objetivo é estabelecer normas e padrões para a tecnologia de *workflow*.

Nesse sentido, a WfMC desenvolveu um modelo de referência e uma série de interfaces que interligam os componentes de um WFMS, de maneira que diferentes sistemas de gerência possam ser interoperáveis entre si (WORKFLOW MANAGEMENT COALITION, 1995). Os componentes ligados a essas interfaces são ferramentas para definição de processos (Interface 1), usuários (Interface 2), aplicações (Interface 3), outros WFMSs (Interface 4) e ferramentas de administração (Interface 5). A figura 2.1, adaptada de Workflow Management Coalition (1995), ilustra os componentes e as interfaces definidas pela WfMC.

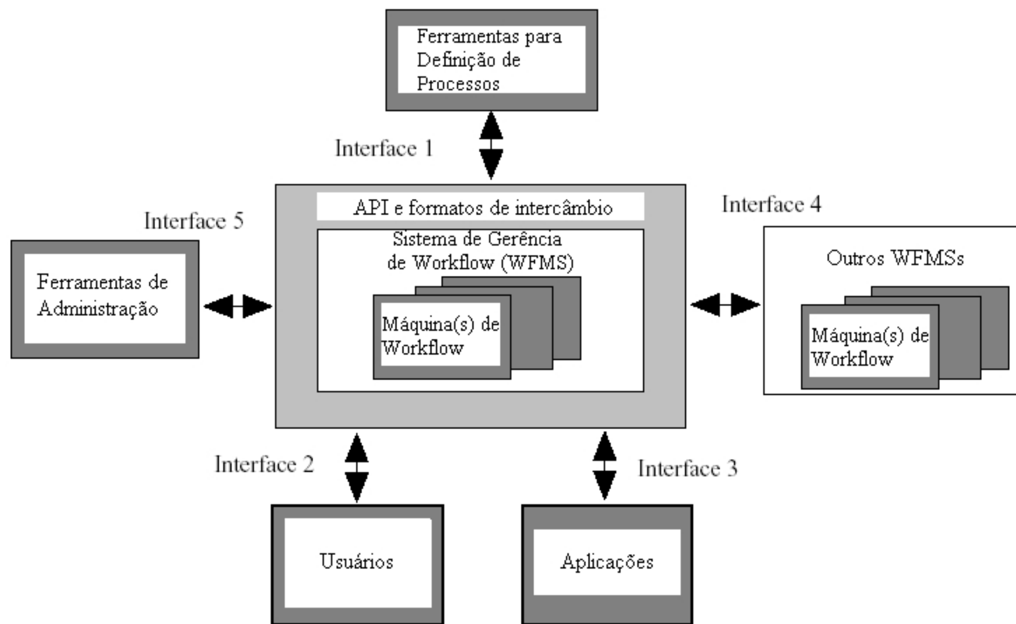


FIGURA 2.1 – Modelo de referência da WfMC, adaptado de Workflow Management Coalition (1995)

2.2 Evolução de *Workflow*

Segundo Reichert e Dadam (1998), um WFMS somente pode ser utilizado de maneira segura e adequada se os processos de negócio possuírem alta estruturação e não necessitarem de modificação em tempo de execução. No entanto, é sabido que a rigidez pode eliminar o dinamismo e a capacidade das organizações de competir num ambiente de mercado competitivo (SADIQ; MARJANOVIC; ORLOWSKA, 2000). Assim, é necessário que ocorra um balanço entre esses dois lados, de forma que haja um equilíbrio entre a definição de uma seqüência de atividades e a possibilidade de modificar essa definição quando necessário.

Existem vários fatores que causam mudanças em processos de negócios:

- **fatores internos** – tais como situações não previstas durante a modelagem, correção de erros e aumento de desempenho;
- **fatores externos** – tais como definição de novas leis governamentais, imposição de novos requisitos por parte do mercado, entre outros.

Algumas propostas extraídas da literatura tentam resolver esses problemas, possibilitando que um *workflow* possa ser alterado sob certas condições. Nas seções a seguir, serão apresentadas algumas dessas propostas.

2.2.1 Proposta de Casati et al.

O modelo de evolução proposto por Casati et al. (1998) parte do princípio de que o processo de evolução de *workflow* possui dois aspectos a serem considerados:

- **evolução estática** – se refere às modificações realizadas sobre a descrição do *workflow*, ou seja, seu esquema;

- **evolução dinâmica** – se refere ao problema da gerência das instâncias ativas de um esquema que foi modificado.

Para o problema da evolução estática, os autores apresentam um conjunto de primitivas, considerado completo, mínimo e consistente. É considerado completo pois estas primitivas são capazes de modificar um esquema de *workflow* qualquer em outro esquema qualquer. É mínimo pois consegue modificar um esquema com o menor conjunto possível de primitivas e é considerado consistente pois é garantido que o esquema será modificado sem erros de compilação e/ou execução. Neste último aspecto, são definidos dois tipos de consistência: a consistência estrutural e a consistência comportamental. O primeiro tipo de consistência refere-se à parte estática do *workflow*, ou seja, o esquema. Dessa forma, a consistência estrutural impõe que um esquema correto, ao sofrer uma modificação, resulte em outro esquema correto. Já o segundo tipo está relacionado à parte dinâmica do *workflow*, ou seja, às instâncias. Assim, a consistência comportamental garante que, ao ser aplicada qualquer modificação a uma instância em execução, o resultado será a instância baseada no novo esquema e ainda correta.

A evolução dinâmica é considerada um problema crucial no contexto de evolução de *workflow*, uma vez que, ocorrendo uma evolução estática pela utilização do conjunto de primitivas de modificação, soluções como reiniciar todas as instâncias ativas (*abort*) ou deixá-las terminar segundo o esquema antigo (*flush*) podem não ser adequadas, pois, invariavelmente, todo o conjunto de instâncias sofreria o mesmo tratamento. Com a utilização de *abort*, muito trabalho já realizado pelas instâncias seria desperdiçado, enquanto que *flush* geraria resultados incorretos. Dessa maneira, os autores propõem algumas políticas que podem ser aplicadas a cada instância em particular, garantindo assim a consistência comportamental das mesmas.

As políticas a serem aplicadas às instâncias levam em consideração o estado de cada instância, ou seja, diferentes decisões podem ser tomadas dependendo do caso. Além disso, múltiplas versões de esquema podem co-existir. As políticas (denominadas pelos autores como progressivas) são as seguintes:

- **concorrente ao término** – as instâncias em execução continuam executando sob a versão de esquema antigo, enquanto que novas instâncias podem ser executadas segundo o novo esquema;

- **migração ao novo esquema** – neste tipo de política, as instâncias são migradas para a nova versão do esquema. No entanto, deve-se verificar se a instância é totalmente compatível com o novo esquema (migração incondicional) ou se a mesma precisa sofrer alguns ajustes para se adequar a ele (migração condicional). Informalmente, uma instância é considerada compatível com determinado esquema se a mesma seguiu algum caminho presente na nova representação. De maneira prática, se a instância está num ponto anterior à modificação, então ela pode ser considerada compatível com o novo esquema, uma vez que o caminho de execução que ela tomou existe na nova representação. Por outro lado, se o ponto em que a instância está é posterior ao ponto de alteração, então deve-se verificar se a mesma tomou algum caminho que seja válido na nova representação, caso contrário, algumas ações como *rollback* ou compensações de atividades devem ser tomadas;

- **migração a um *workflow ad-hoc*** – nesta política, o administrador do *workflow* pode definir um procedimento diferenciado para uma determinada instância, de forma a não comprometer sua execução. Isso pode ser alcançado por meio da

definição de um conjunto específico de atividades, que causariam um resultado similar ao do novo esquema definido;

- **abortar** – a instância é totalmente desfeita.

É importante destacar que as políticas progressivas concorrente ao término e abortar são essencialmente diferentes do que *flush* e *abort*, respectivamente. Isso porque as políticas permitem que mais de um esquema atue ao mesmo tempo, possibilitando que um número maior de instâncias sejam completadas com sucesso, de acordo com as novas especificações, sem comprometer todas as instâncias em conjunto, o que não era possível com a utilização de *flush* e/ou *abort*.

Uma vez ocorrendo a mudança, é necessário que as instâncias sejam agrupadas de acordo com a política à qual elas serão submetidas. Esse processo pode ser automático (no caso de migração incondicional) ou manual, quando o administrador deve decidir o que fazer em cada caso. Após esse processo, somente seguirão o esquema antigo aquelas instâncias que utilizaram a política concorrente ao término, enquanto que o novo esquema conterá, além de suas próprias instâncias, aquelas que utilizaram a política de migração ao novo esquema. A figura 2.2 apresenta as características discutidas.

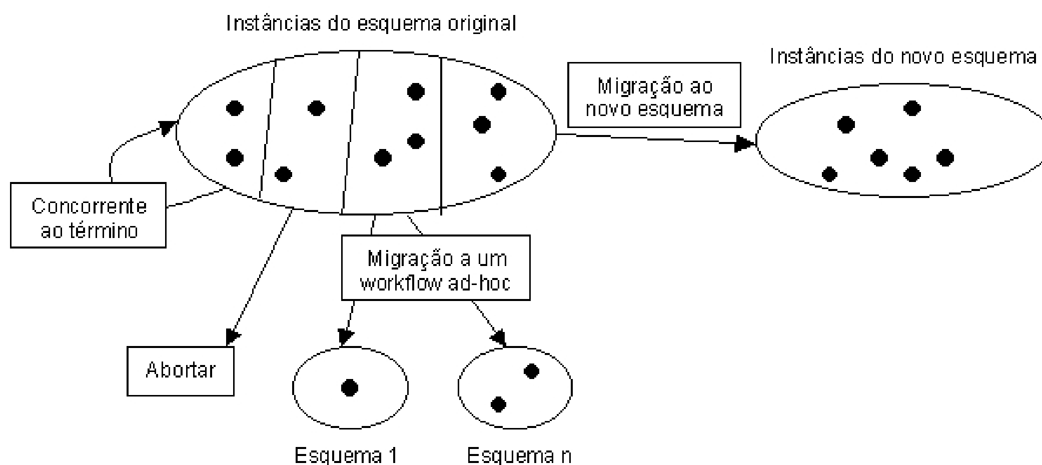


FIGURA 2.2 – Separação das instâncias do esquema e adequação das mesmas quanto às políticas progressivas (CASATI et al., 1998)

2.2.2 O Ambiente ADEPT

O ambiente ADEPT (*Application Development Based on Encapsulated Premodeled Process Templates*), conforme definido por Reichert e Dadam (1998), é um modelo formal para a definição de *workflows*, sendo que a este modelo se soma um conjunto de operações de modificação, ADEPT$_{flex}$, considerado completo e mínimo. O enfoque principal desta proposta é a alteração dinâmica de instâncias de *workflow*, ao contrário da proposta de Casati et al. (1998), que destacava as mudanças nos esquemas de *workflow*. Os autores citam como exemplo um ambiente hospitalar, cujos processos precisam ser flexíveis dependendo da urgência do atendimento aos pacientes.

Os argumentos destacados nessa proposta para a modificação de instâncias giram em torno de aspectos como: impossibilidade, em alguns casos, do processo ser completamente modelado; eventos que não foram previstos em tempo de definição; ajustes específicos em determinada instância (*ad-hoc workflow*).

Outra contribuição dessa proposta é um estudo aprofundado das implicações no fluxo de dados a partir de mudanças nas instâncias. É proposto um conjunto de regras que verificam se ocorrem violações, como por exemplo, a leitura de valores de dados inválidos e o problema de escrita-sobre-escrita (*lost update*) entre tarefas em paralelo. Além disso, algumas regras para a correção do fluxo de controle também são especificadas.

A aplicação de mudanças permanentes e temporárias também foi discutida nessa proposta. Tipicamente, as alterações aplicadas sobre um determinado esquema valem daquele momento em diante. No entanto, além do enfoque ser dado às modificações nas instâncias, o modelo permite a ocorrência de laços de repetição. Com a utilização de mudanças temporárias, pode-se aplicar alterações dentro de um laço que somente valem para aquela volta, sendo desfeitas no próximo retorno.

2.2.3 Proposta de Kradolfer

A proposta de Kradolfer, diferentemente das anteriores, é inteiramente baseada no conceito de versionamento de esquemas. Dessa forma, um esquema modificado pode continuar sendo válido, juntamente com o novo esquema criado. Isso possibilita a criação de uma árvore de versões, como ilustra a figura 2.3, a partir de sucessivas alterações nos esquemas constituintes (KRADOLFER; GEPPERT, 1999; KRADOLFER, 2000).

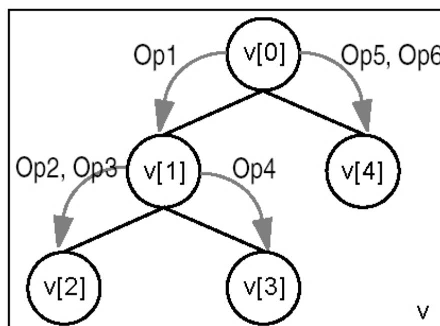


FIGURA 2.3 – Árvore de versões do esquema de *workflow* v (adaptado de Kradolfer e Geppert (1999))

Como efeito imediato, as instâncias podem ou não ser migradas para a nova versão de esquema, dependendo do estado da instância. Assim sendo, pode-se ter, num determinado instante de tempo, várias versões de esquema ativas, com instâncias executando segundo cada uma delas.

O autor também propõe a utilização de invariantes de esquema, de forma a garantir a correção do mesmo, tanto antes quanto depois de uma modificação.

Uma versão de esquema é composta por *workflows* complexos e/ou atividades simples. Um *workflow* complexo é aquele que possui *subworkflows* internos a ele, ou seja, possui subprocessos. Esses *subworkflows* são versões de outros *workflows*, sendo portanto associados a outras árvores de versões existentes. Isso possibilita que as versões possam ser reusadas em vários esquemas, o que Kradolfer defende em sua tese de doutorado (KRADOLFER, 2000). Já uma atividade simples não possui subdivisões, sendo considerada atômica.

Kradolfer também definiu uma taxonomia de operações para a modificação de esquemas e de versões de esquemas. Deve-se assegurar que as modificações mantêm o

esquema correto, ou seja, respeitam as invariantes. Elas se resumem, basicamente, a quatro grandes operações:

- **criação de versão** – qualquer modificação efetuada sobre uma versão existente cria uma nova versão, derivada da primeira;
- **remoção de versão** – essa operação somente pode ser aplicada caso a versão seja folha da árvore e não possua instâncias em execução;
- **criação de esquema** – quando um novo esquema de *workflow* é criado, automaticamente é incluída a raiz da árvore de versões (primeira versão);
- **remoção de esquema** – um esquema pode ser removido se não possuir instâncias nem for referenciado por outro esquema.

Além disso, o autor também apresentou regras para migração de instâncias entre versões de esquemas. De acordo com essa proposta, uma instância de uma versão de *workflow* somente pode ser migrada para outra versão se, no momento da migração, ela executou alguma seqüência válida de acordo com a nova versão de esquema. Se naquele momento a instância tiver executado ou deixado de executar alguma atividade anterior, a migração não é possível. Também foi analisada a migração de instâncias entre diferentes versões, tanto descendentes quanto ascendentes.

2.2.4 Considerações sobre Evolução de *Workflow*

As três propostas de evolução de *workflow* apresentadas anteriormente são apenas algumas dentre várias existentes na literatura. Existem outras alternativas que serão resumidas a seguir.

Ellis, Keddara e Rozenberg (1995) definiram um modelo matemático, baseado em Redes de Petri. O processo de mudança, nesse modelo, se dá por substituição de pedaços de uma rede, não existindo operações específicas de modificação. Além disso, também estabeleceram critérios para correção dos esquemas e das instâncias. Esta foi uma das primeiras propostas para a área de evolução de *workflow*.

Joeris e Herzog (1999) também definiram sua proposta de evolução baseada em versionamento de esquemas, da mesma forma que Kradolfer (seção 2.2.3). Além disso, discutem temas como granulosidade do versionamento e estratégias para propagação de instâncias. No entanto, os aspectos de migração são mais restritos.

Weske (2001) apresenta um conceito de mapeamento entre instâncias e esquemas, de forma que uma instância somente pode ser adaptada de um esquema para outro se houver um mapeamento entre a instância e o novo esquema. Além disso, a proposta também se baseia no reuso de *subworkflows*.

S. Sadiq, W. Sadiq e Orlowska (2001) apresentam uma proposta para a representação de processos que necessitem de flexibilidade na composição do fluxo de controle. O conceito de flexibilidade nesse contexto está ligado à definição de um esqueleto mínimo de esquema, de forma que a especificação total do modelo seja feita em tempo de execução. Dessa maneira, o administrador pode utilizar-se do chamado *pocket* de flexibilidade, que é uma coleção de atividades cuja seqüência de execução é definida enquanto a instância está ativa.

Apesar do número variado de alternativas, ainda falta uma proposta que seja eficaz para o problema apresentado, especialmente devido à natureza dos processos. Enquanto

alguns são totalmente automatizados e possuem atividades que podem ser desfeitas, outros são intrinsicamente manuais. Outros, apesar da automatização, não permitem que atividades sejam desfeitas. Em suma, pode-se dizer que as características dos processos modelados determinam o sucesso ou o fracasso das alternativas de evolução.

2.3 *Workflow* e XML

A linguagem XML é um padrão de declaração de documentos, proposto pela W3C (*World Wide Web Consortium*) (WORLD WIDE WEB CONSORTIUM, 2000). Dentre as suas muitas utilidades, ela pode ser entendida como uma meta-linguagem, pois permite a descrição de novas linguagens baseadas na sua sintaxe, nos mais diferentes domínios de aplicação. Isso é possível por meio da utilização de DTDs (*Document Type Definition*) ou XML *Schemas*, que definem a estrutura que será seguida pelos documentos.

Uma das áreas à qual XML está sendo aplicada atualmente é a gerência de *workflow*. De acordo com os documentos da WfMC, duas de suas interfaces estão utilizando XML como base de suas linguagens. Além disso, alguns outros projetos de pesquisa também estão indo na mesma direção, enfocando os mais variados aspectos.

A seguir, serão apresentadas algumas propostas de linguagens e modelos que utilizam XML para representar aspectos da tecnologia de *workflow*. Três dessas propostas vêm da indústria (XPDL, Wf-XML e WSFL), enquanto que as outras duas são acadêmicas (XRL e WQM). Ao final, um comparativo entre elas é realizado.

2.3.1 XPDL

XPDL (*XML Process Definition Language*) é uma linguagem desenvolvida pela WfMC para a representação de processos de negócio em XML (WORKFLOW MANAGEMENT COALITION, 2002). Esta linguagem integra a chamada Interface 1 da WfMC, responsável pela parte de intercâmbio entre diferentes ferramentas de modelagem de *workflow*. Sendo assim, a XPDL funciona como um modelo comum, de forma que qualquer ferramenta pode importar um processo descrito nessa linguagem ou exportar um processo em sua linguagem nativa.

A figura 2.4 ilustra como XPDL é utilizada no processo de intercâmbio. Cada uma das ferramentas, a partir de sua representação interna, exporta seu processo para XPDL, sendo possível sua utilização por outra(s) ferramenta(s).

A linguagem XPDL é composta por vários elementos XML, como por exemplo atividades, transições, participantes, dados, parâmetros, entre outros. Todas essas informações são colocadas dentro de um elemento `Package`, conforme a figura 2.5. A figura apresenta um trecho de um documento em XPDL, mostrando a declaração de uma atividade e de uma transição. É importante observar que um mesmo pacote pode conter vários processos de *workflow*.

2.3.2 Wf-XML

A linguagem Wf-XML, assim como XPDL, também foi desenvolvida pela WfMC (WORKFLOW MANAGEMENT COALITION, 2001). Porém, seu propósito é diferente desta última. Enquanto XPDL é apropriada para a modelagem dos processos, Wf-XML é utilizada para a interoperação entre diferentes máquinas de *workflow*, integrando a Interface 4 da WfMC. Uma determinada máquina de *workflow*, por exemplo, pode solicitar que

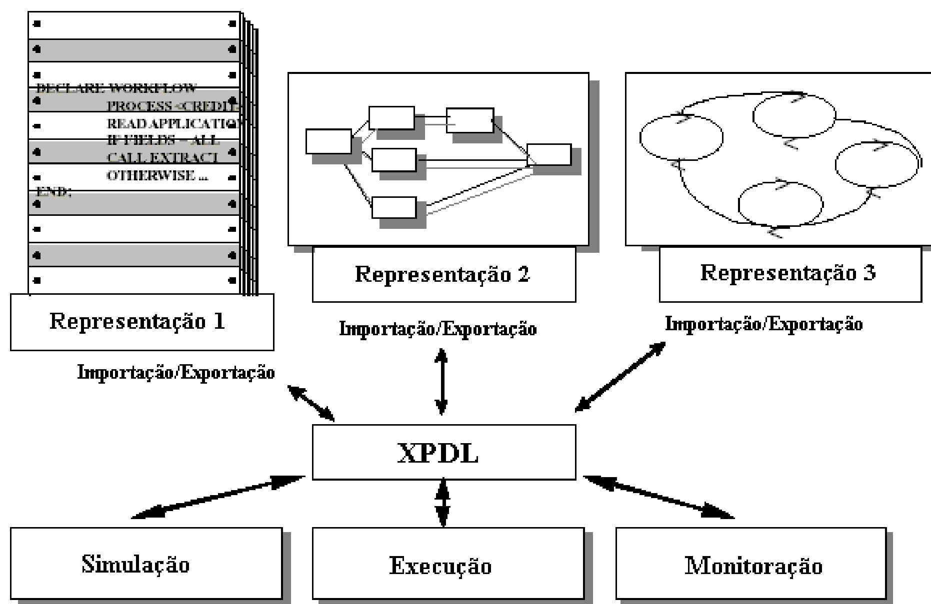


FIGURA 2.4 – Intercâmbio entre diferentes ferramentas de modelagem de *workflow* (adaptado de Workflow Management Coalition (2002))

outra máquina execute determinados (sub-)processos, esperando ou não que os mesmos terminem para continuar sua execução. Essa passagem do controle de execução entre diferentes máquinas de *workflow* é realizada utilizando-se o mecanismo de mensagens, devidamente descritas nos documentos XML.

Basicamente, existem três modelos de interoperabilidade definidos pela WfMC, que são implementados pela linguagem Wf-XML:

- **encadeada** – o controle é passado de uma máquina de *workflow* para outra, sendo que a primeira pode continuar executando outras atividades enquanto a segunda está trabalhando;
- **aninhada** – o controle é passado de uma máquina de *workflow* para outra, no entanto, a primeira deve esperar até que a segunda retorne o controle para continuar sua execução;
- **paralelo sincronizado** – neste tipo de interoperabilidade, dois processos podem ser executados paralelamente em duas máquinas diferentes, mas são sincronizados em pontos determinados, o que é útil para o controle das execuções dos mesmos.

A figura 2.6 mostra um esqueleto de uma mensagem entre duas máquinas de *workflow* quaisquer.

2.3.3 WSFL

A linguagem WSFL (*Web Services Flow Language*) foi desenvolvida pela IBM, com a finalidade de descrever composições de serviços *Web*. De acordo com IBM Corporation (2001), dois tipos de composições são considerados:

- **modelos de fluxo** – determinam seqüências de execuções entre serviços *Web*;

```
<Package ...>
   ...
   <Participants>
      ...
   </Participants>
   <WorkflowProcesses>
      <WorkflowProcess Id="1" Name="" ...>
         ...
         <Applications>
            ...
         </Applications>
         <Activities>
            <Activity Id="10">
               ...
            </Activity>
            ...
         </Activities>
         <Transitions>
            <Transition Id="101" From="10" To="11"/>
            ...
         </Transitions>
      </WorkflowProcess>
      ...
   </WorkflowProcesses>
</Package>
```

FIGURA 2.5 – Parte de um documento XPDL

```
<WfMessage xmlns="..." Version="1.1">
   <WfTransport>
      ...
   </WfTransport>
   <WfMessageHeader>
      ...
   </WfMessageHeader>
   <WfMessageBody>
      ...
   </WfMessageBody>
</WfMessage>
```

FIGURA 2.6 – Exemplo de uma mensagem em Wf-XML

- **modelos globais** – como os serviços *Web* interagem uns com os outros.

Embora em contextos diferentes, WSFL e XPDL são bastante semelhantes, uma vez que determinam atividades a serem executadas e uma seqüência que deve ser obedecida para que o serviço *Web*/processo possa ser realizado com sucesso. Isso pode ser constatado na figura 2.7, que apresenta um esboço de um modelo de fluxo de serviço *Web*.

2.3.4 XRL

XRL (*Exchangeable Routing Language*) é um ambiente que permite a definição de *workflows* e o posterior intercâmbio dos processos e documentos entre várias organizações participantes de comércio eletrônico. Foi definida em termos de Redes de Petri, sendo possível sua formalização e verificação (AALST; KUMAR, 2000).

A linguagem é composta de construções que podem ser usadas como blocos básicos para o projeto de esquemas para aplicações inter-organizacionais, ou seja, fora do escopo de uma só empresa. XRL é baseada em XML, possuindo estruturas específicas para a definição de roteamentos com semânticas próprias. Tais construções são dos seguintes

```
<flowModel ...>
   <serviceProvider ...>
      ...
   </serviceProvider>
   <activity name="...">
      ...
   </activity>
   ...
   <controlLink source="..." target="..."/>
   <dataLink.../>
</flowModel>
```

FIGURA 2.7 – Exemplo de um serviço *Web* em WSFL

tipos:

- **tarefa** – relaciona-se com uma ação a ser realizada. Esse elemento possui uma série de atributos, como nome, papel, documentos necessários, tempos de início e fim, entre outros;
- **seqüência** – os elementos dentro de uma seqüência devem ser executados na ordem em que estão dispostos;
- **paralelismo** – os elementos dentro de uma construção de paralelismo podem ser executados ao mesmo tempo. Três tipos são oferecidos: o paralelismo com sincronismo total (todos os elementos em paralelo devem terminar suas execuções para que o fluxo continue), o paralelismo com sincronismo parcial (o fluxo pode continuar quando alguns elementos terminarem) e paralelismo sem sincronismo (não existe espera entre os elementos paralelos);
- **condicional** – somente os elementos que satisfizerem determinada condição serão executados;
- **repetição** – construção que permite a execução repetida dos elementos internos. Neste caso, uma condição para o laço deve ser testada.

Além dessas construções básicas, XRL também oferece elementos de espera (*wait*), de eventos, de *timeouts*, de término, entre outros.

A figura 2.8 ilustra um exemplo de um processo descrito em XRL.

2.3.5 WQM

A linguagem WQM (*Workflow Query Model*) define, da mesma forma que XRL, estruturas para a representação do fluxo de controle de um processo. São elas: **sequence**, **choice**, **parallel** e **loop**. No entanto, esta linguagem tem como objetivo primordial a realização de consultas, tanto sobre os esquemas quanto sobre as execuções de instâncias. O processo é especificado utilizando-se XML Query Algebra (CHRISTOPHIDES; HULL; KUMAR, 2001).

As consultas a serem realizadas pode ser reunidas, basicamente, em três classes:

- **simples** – as consultas simples são relacionadas a execuções de instâncias em particular. Exemplos de consultas: qual o estado de uma atividade, quantas vezes um determinado *loop* foi executado, etc. São consideradas consultas locais, pois podem ser realizadas sobre os dados já armazenados nas instâncias;

```
<route>
   <sequence>
      <task name="..." start_time="..." end_time="..."/>
      <parallel_sync>
         <task .../>
         <task .../>
      </parallel_sync>
      <condition condition="...">
         <true>
            ...
         </true>
         <false>
            ...
         </false>
      </condition>
   </sequence>
</route>
```

FIGURA 2.8 – Exemplo de um processo em XRL

- **relacionadas** – esta classe verifica aspectos mais amplos, considerando esquemas e instâncias. Exemplos: qual o relacionamento entre duas atividades, qual o tempo estimado para uma instância, etc. Estas consultas são mais complexas, pois envolvem a utilização de funções específicas para a obtenção das respostas desejadas;
- **construção de esquema** – esta classe pode ser usada para a construção de esquemas de *workflow*, a partir da utilização de componentes de uma biblioteca de *templates*.

A figura 2.9 ilustra um processo hipotético representado em WQM.

```
Route[
  Data_list[...],
  Sequence[
    Task[@name["..."]]
    Parallel[
      Task[@name["..."]]
      Task[@name["..."]]
    ]
    Choice[@condition["..."]
      ...
    ]
  ]
]
```

FIGURA 2.9 – Exemplo de um processo WQM em XML Query Algebra

Os autores deste modelo consideram que a utilização do mesmo permite um incremento no projeto e na eficiência de *workflows*, tanto em tempo de construção quanto em tempo de execução, uma vez que, a partir das consultas realizadas sobre os esquemas e as instâncias, várias anomalias podem ser detectadas e corrigidas.

2.3.6 Considerações sobre *Workflow* e XML

A seção 2.3 apresentou algumas propostas existentes que relacionam as áreas de *workflow* e XML. Pode-se perceber que, dentre todas elas, somente Wf-XML não se relaciona com a parte de modelagem de processos de negócio, estando diretamente ligada à

interoperabilidade dos mesmos entre diferentes máquinas. Sendo assim, as considerações que são feitas aqui são aplicadas às outras propostas (XPDL, WSFL, XRL e WQM).

A partir dos exemplos e pedaços de código apresentados nessa seção, pode-se realizar um comparativo entre as quatro linguagens, no que tange à forma de representação dos elementos e do fluxo de controle respectivo. Dessa forma, pode-se inseri-las em uma das abordagens a seguir (ZSCHORNACK; EDELWEISS, 2002):

- **representação explícita de transições (RET)** – a característica fundamental das linguagens pertencentes a este grupo é a presença de algum elemento XML que denote explicitamente uma transição, ligando uma atividade a outra diretamente. Para isso, tal elemento deve possuir atributos que indiquem o elemento do qual ele "parte" (origem) e o elemento no qual ele "chega" (destino). Exemplos de linguagens baseadas nesta abordagem são XPDL (atributos `from` e `to`) e WSFL (atributos `source` e `target`);

- **representação implícita de transições (RIT)** – os modelos pertencentes a esta abordagem, ao contrário dos modelos RET, não possuem transições explícitas, mas são compostos de estruturas que determinam como será construído o fluxo de controle do *workflow*. Cada estrutura possui uma semântica própria que, obviamente, deve ser conhecida pela máquina de *workflow*. XRL e WQM são exemplos de modelos que fazem parte dessa abordagem.

A figura 2.10 apresenta parte de um processo hipotético, representado de três maneiras equivalentes: gráfica (a), baseado na RET (b) e baseado na RIT (c). As representações não estão utilizando a sintaxe de alguma ferramenta em particular, sendo definidas livremente, seguindo as particularidades das abordagens.

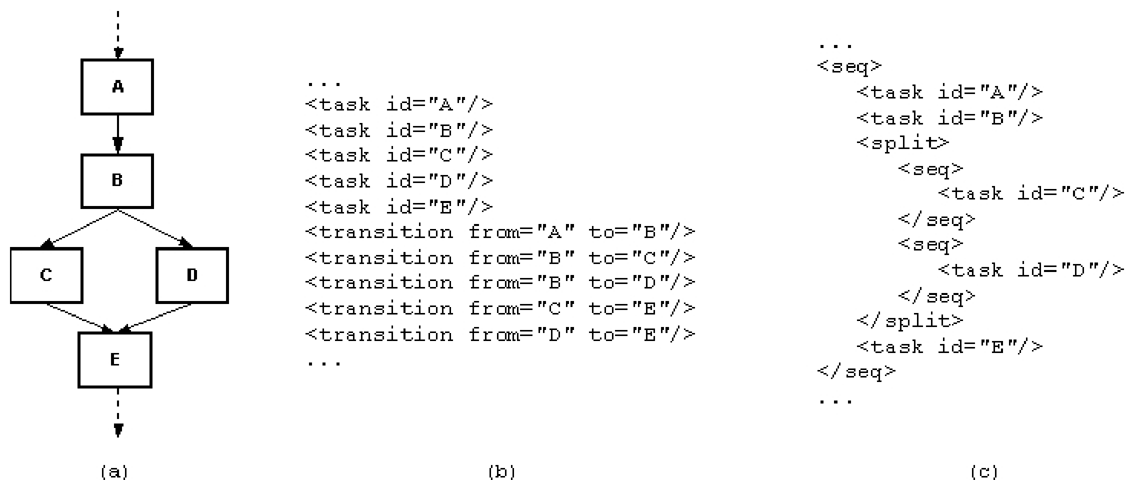


FIGURA 2.10 – Representação de *workflow* em XML com transições explícitas (b) e implícitas (c), a partir de um modelo gráfico (a)

Uma análise das linguagens apresentadas nesta seção nos mostra que as abordagens anteriormente descritas possuem aspectos positivos e negativos. Em alguns critérios de comparação, as abordagens são opostas uma em relação a outra. Tomando como critério a liberdade de modelagem, observa-se que as linguagens que representam as transições explicitamente oferecem mais recursos ao modelador de processos para construir um *workflow*, pois praticamente não impõem restrições quanto às possíveis combinações de estru-

turas. Este aspecto pode ser considerado como uma desvantagem das linguagens baseadas em RIT, pois estas oferecem somente estruturas pré-definidas para a modelagem dos processos. A figura 2.11 ilustra um exemplo de fluxo que pode ser modelado segundo a abordagem RET, mas que as linguagens que se baseiam em transições implícitas são incapazes de representar em XML, pelo menos na forma em que se apresenta, por problemas de aninhamento de estruturas.

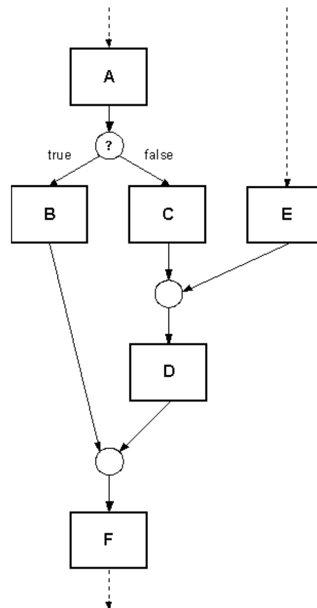


FIGURA 2.11 – Fluxo que não pode ser modelado pela abordagem RIT

A verificação de correção dos esquemas é outro aspecto no qual as duas abordagens são opostas. Nos modelos baseados em RET, tal tarefa é complexa, porque uma DTD, neste caso, tem pouca força para especificar se um documento é correto ou não (determina somente se o documento é válido). Por exemplo, a partir de uma DTD ou *Schema*, não é possível a realização de verificações de valores de atributos. Por outro lado, nos modelos que representam implicitamente as transições, a correção é uma tarefa mais simples, pois a DTD ou *Schema* é capaz de especificar como as estruturas devem ser dispostas no documento XML. Assim, o processo de validação serve também como verificação de correção.

2.4 Considerações Finais

Os conceitos e propostas apresentados neste capítulo dão uma idéia da importância do tema "evolução de *workflow*". Pela natureza dinâmica dos processos de negócio, os *workflows* necessitam ser modificados, assim como *softwares*, em virtude de mudanças de requisitos. Dessa forma, a construção de *workflows* estáticos ou de difícil alteração causa inúmeros problemas, uma vez que os resultados não serão os esperados pelos usuários.

Mesmo sendo uma linguagem puramente textual, a utilização de XML como formato para representação de *workflow* se constitui em uma boa alternativa, pois facilita o intercâmbio de processos entre várias partes. No entanto, mesmo que a modelagem do processo sirva apenas para o âmbito de uma organização, os aspectos positivos se destacam. Documentos XML podem ser interpretados por muitos *softwares* em várias platafor-

mas, ao contrário de sistemas específicos e/ou proprietários. Além disso, XML oferece uma determinada estruturação de elementos e permite a definição de novas linguagens, orientadas ao domínio específico.

Apesar de existirem algumas propostas de modelagem e representação de *workflow* em XML, sejam acadêmicas ou industriais, ainda não se tem conhecimento de alguma que possua, como objetivo principal, o tratamento da evolução. Esse tópico, inclusive, é raramente citado nas publicações relatadas na seção 2.3. Por não existir essa preocupação, alguns modelos tornam essa tarefa muito complexa, especialmente os que seguem a abordagem RET.

A partir do capítulo 3, será apresentada uma proposta que visa tratar de forma unificada a representação e a evolução de *workflow* em XML, utilizando para tanto vários dos conceitos e idéias apresentadas neste capítulo.

3 Representação e Alteração de Esquemas de *Workflow* em XML

No capítulo 2, algumas propostas para modelagem de *workflow* em XML foram apresentadas. No entanto, como frisado na seção 2.4, essas propostas não foram desenvolvidas levando-se em consideração a evolução dos esquemas. Diante disso, uma vez que é grande a possibilidade de um esquema ser modificado, deve-se ter um mecanismo que permita o tratamento adequado tanto de esquemas quanto de instâncias.

O presente capítulo apresenta duas linguagens: a primeira, definida na seção 3.1, permite a representação de *workflows* em XML que respeitam algumas propriedades essenciais para o processo de evolução; a segunda, descrita na seção 3.2, permite a alteração de documentos XML e, no contexto deste trabalho, de esquemas de *workflow*. Por fim, na seção 3.3 são feitas algumas considerações sobre este capítulo.

3.1 Linguagem para Representação de *Workflow* em XML

3.1.1 Motivação

Nesta seção descreve-se uma linguagem para a representação de esquemas de *workflow* em XML. A linguagem, entre outros aspectos, é apropriada para a aplicação de mudanças, tendo sido desenvolvida para esta finalidade, possibilitando assim a evolução de esquemas de *workflow*. Para que tal objetivo fosse alcançado, a linguagem baseou-se em algumas propriedades:

- **correção** – não deixar com que o usuário crie fluxos não permitidos ou com erros, o que pode ser garantido por meio de regras sintáticas;
- **simplicidade** – capacidade de se representar um *workflow* utilizando um número mínimo de estruturas de modelagem;
- **clareza** – relaciona-se com o entendimento que o usuário possui do processo modelado, ou seja, a linguagem deve permitir que o usuário compreenda facilmente o processo, apesar da natureza textual de XML;
- **facilidade de alteração** – possibilidade de alterar um esquema de *workflow* facilmente, sem causar erros de correção.

A definição de uma linguagem específica se faz necessária, pois as linguagens apresentadas na seção 2.3 não satisfazem ou satisfazem apenas parcialmente algumas destas propriedades.

Linguagens baseadas em RET (XPDL e WSFL, por exemplo) possuem poucos mecanismos para verificação de correção. Um exemplo deste problema pode ser a ligação, por meio de um elemento `transition`, entre duas atividades, sendo que uma delas (ou até as duas) pode não existir. A informação de fonte e destino da transição é armazenada em atributos específicos deste elemento. Sendo assim, uma DTD ou *Schema* não teria condições de verificar se o valor do atributo corresponde a um valor existente de atividade. Pode-se constatar, portanto, que a correção é um aspecto muito importante e deve ser levado em consideração.

Além disso, quando se pretende alterar um *workflow*, outras dificuldades podem surgir. Isso porque a tarefa de modificação pode-se transformar numa maçante alteração de valores de atributos, de maneira praticamente manual, o que, novamente, pode levar a esquemas inconsistentes e/ou incorretos.

Por tais argumentos, as linguagens baseadas em RET são inadequadas para a representação de *workflows* que primem pela correção e pela facilidade de alteração, duas das propriedades descritas.

Por outro lado, a linguagem XRL também não é a mais adequada, uma vez que é orientada a instâncias, ou seja, cada *workflow* representado nessa linguagem constitui-se num caso isolado. Se for necessário alterar muitas instâncias, este trabalho deve ser realizado caso a caso. Por sua vez, WQM é uma linguagem que visa a realização de consultas, não sendo, assim como XRL, originalmente definida para o tratamento da evolução de esquemas.

A linguagem definida neste trabalho somente modela o fluxo de controle do *workflow*, ou seja, as atividades e as ligações entre elas, desconsiderando o fluxo de dados (que informações serão manipuladas?) e o modelo organizacional (quem realizará as atividades?) do mesmo. Essa decisão foi tomada visto que é no fluxo de controle de um *workflow* que se concentra a grande maioria das alterações que são realizadas, o que é demonstrado pelas propostas de evolução apresentas na seção 2.2. Essa política foi adotada pois a intenção não é a de se ter uma linguagem completa ou que represente o maior número possível de estruturas, mas trabalhar sobre um conjunto menor e garantir que as mudanças nos esquemas possam ser aplicadas adequadamente sobre este conjunto. Sendo assim, esta linguagem contém estruturas suficientes, sendo apropriada para a aplicação de evolução em seus esquemas.

A base conceitual e teórica escolhida para a linguagem de representação de *workflow* em XML é calcada numa abordagem estruturada, que será apresentada a seguir.

3.1.2 *Workflows* Estruturados

***Workflows* estruturados** é uma classe de *workflows* definida por Kiepuszewski, Hofstede e Bussler (2000), com algumas características especiais. Esta classe vem a ser um subconjunto de uma classe genérica, a dos ***workflows* arbitrários**, de maneira que todo esquema definido como um *workflow* estruturado é também um *workflow* arbitrário, mas o inverso pode não ser verdadeiro.

Um *workflow* arbitrário pode ser considerado, intuitivamente, como um *workflow* que não apresenta nenhuma espécie de restrição quanto à modelagem do fluxo de controle, ou seja, o *workflow* pode ser livremente projetado.

Por outro lado, um *workflow* estruturado possui uma série de restrições sintáticas que limitam a modelagem dos processos. Basicamente, o que se procura evitar com o uso dessas restrições são problemas como *deadlocks* e ausência de sincronismo entre ramos de execução, que são problemas que podem acontecer com modelos totalmente livres (SADIQ; ORLOWSKA, 1999).

As restrições sintáticas são representadas por estruturas específicas de modelagem, as quais devem ser arranjadas de forma totalmente aninhada, sem sobreposição. Estas estruturas são as seguintes:

- **seqüência** – nesta estrutura, as atividades serão executadas linearmente, ou seja, uma atividade só poderá executar se a anterior tiver terminado;

- **paralelismo** – as atividades podem ser executadas ao mesmo tempo, sendo sincronizadas ao final;
- **decisão** – somente serão habilitados os ramos de execução que respeitarem determinadas condições;
- **repetição** – as atividades podem executar de maneira repetitiva, enquanto/até que uma determinada condição for verdadeira/seja falsa.

A figura 3.1 apresenta uma esquematização destas estruturas.

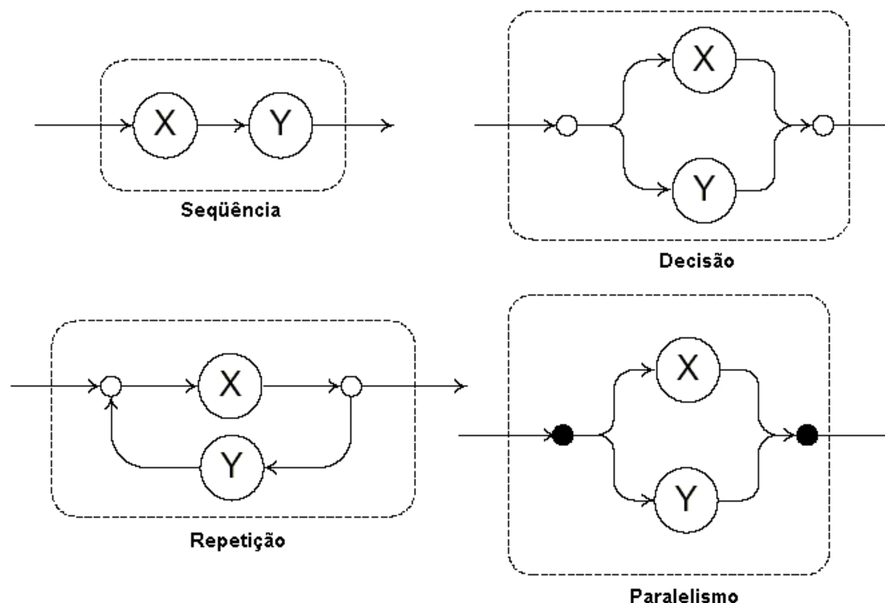


FIGURA 3.1 – Estruturas definidas por Kiepuszewski, Hofstede e Bussler (2000)

A existência de estruturas específicas resulta em outras restrições como, por exemplo, a obrigatoriedade de um único elemento de início e um único elemento de fim de fluxo. Por outro lado, a utilização somente destas estruturas constitui uma limitação de poder de expressão dos modelos, uma vez que não permite a definição de qualquer fluxo, como no caso dos *workflows* arbitrários. Por exemplo, o fluxo representado na figura 2.11 (página 28) é um *workflow* arbitrário, porém o mesmo não pode ser caracterizado como um *workflow* estruturado, pois o ramo que contém a atividade E "entra" na estrutura condicional, sobrepondo dessa maneira esta estrutura. Esse tipo de limitação, no entanto, pode ser contornado em alguns casos, por meio de transformações nos fluxos arbitrários, de forma que se tornem equivalentes aos *workflows* estruturados. Alguns casos de transformações de *workflows* arbitrários para *workflows* estruturados podem ser encontrados em Kiepuszewski, Hofstede e Bussler (2000).

A partir da descrição desse tipo de *workflow*, é possível traçar uma analogia desta classe com documentos XML, no nível da sintaxe. Da mesma forma que a classe dos *workflows* estruturados, XML também se utiliza de blocos ou elementos completamente aninhados, não sendo permitida sobreposição entre eles. Além disso, a obrigatoriedade de início e fim únicos pode ser comparada ao elemento raiz de um documento XML, que engloba todos os outros elementos. Por outro lado, as classes de *workflows* arbitrários e

workflows estruturados possuem estreita relação com as abordagens RET e RIT, respectivamente, descritas na seção 2.3.6.

Diante disso, complementado pelas vantagens apresentadas pelos modelos que seguem a abordagem RIT, decidiu-se seguir esta estratégia de representação de *workflows*, utilizando a classe dos *workflows* estruturados como base para o desenvolvimento de uma linguagem que, entre outras características, pudesse preencher as propriedades propostas na seção 3.1.1. Na seqüência, os elementos da linguagem definida serão descritos em detalhes.

3.1.3 Elementos da Linguagem

A linguagem para a representação de *workflow* em XML apresentada nesta seção é baseada na abordagem RIT e nos *workflows* estruturados, sendo que os elementos da linguagem equivalem às estruturas apresentadas na seção 3.1.2. A linguagem foi definida em XML *Schema*. A figura 3.2 apresenta sua estrutura básica. No anexo 1 é apresentado o XML *Schema* completo da linguagem.

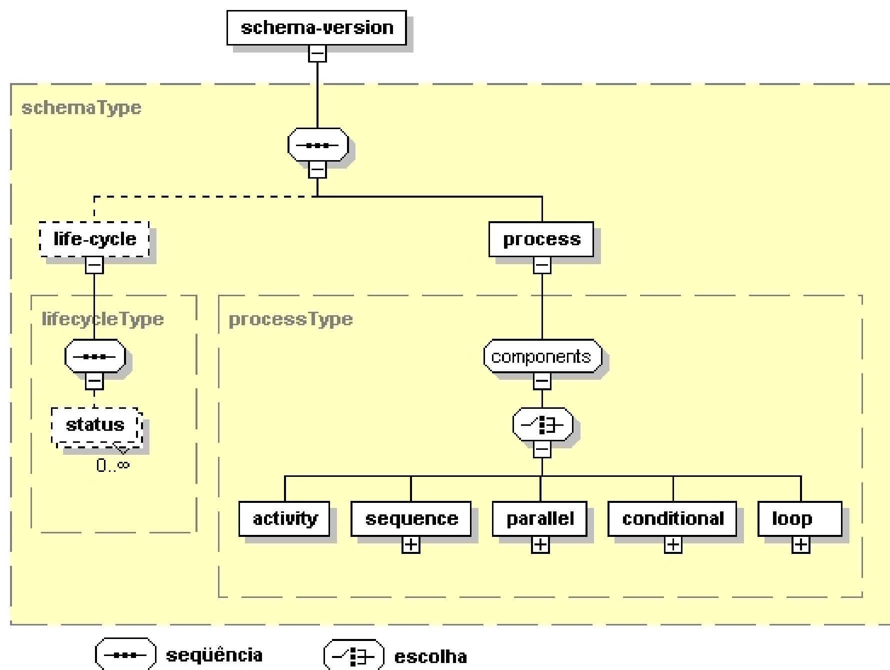


FIGURA 3.2 – Estrutura básica da linguagem de representação de *workflow*

De acordo com a figura 3.2, o elemento **`schema-version`** é a raiz da árvore, ou seja, engloba todos os outros elementos da linguagem. Este elemento possui dois filhos imediatos: **`life-cycle`** e **`process`**. O elemento `life-cycle` pode conter elementos **`status`**. O elemento `process`, por sua vez, é composto por um dos seguintes elementos: **`activity`**, **`sequence`**, **`parallel`**, **`conditional`** ou **`loop`**. Nessa figura, elementos obrigatórios são representados por retângulos brancos cheios, enquanto que elementos opcionais são denotados por retângulos brancos tracejados. Dessa forma, o elemento `life-cycle` (e conseqüentemente `status`) é considerado opcional para o construtor de esquemas, pois, como será apresentado no capítulo 5, este elemento refere-se a parte de versionamento de esquemas e, particularmente, ao ciclo de vida da versão. Dessa maneira, `life-cycle` e `status` são manipulados pelo sistema de gerência de *workflow*

que coordena as versões de esquema.

A seguir são apresentados os elementos e estruturas que compõem a parte da descrição do fluxo de controle da linguagem, ou seja, o elemento `process` e seus filhos.

Process

O elemento **`process`** é o elemento que engloba o processo de negócio em si, ou seja, todo o fluxo de controle. Com esse elemento, garante-se uma das restrições dos *workflows* estruturados, que exige que um processo tenha pontos únicos de início e de fim. Este elemento aceita somente um elemento filho imediato, que pode ser uma das estruturas básicas ou uma atividade, apresentadas a seguir.

Activity

O elemento **`activity`** representa a menor unidade de trabalho da linguagem, ou seja, uma atividade propriamente dita. Este elemento é responsável pela parte "executiva" da linguagem, enquanto que os outros elementos são apenas roteadores. É comparável ao elemento `task`, presente em XPDL, XRL e WQM. Seu formato geral é:

```
<activity name="A"/>
```

O elemento `activity` é composto de um atributo **`name`**, que indica o nome da atividade, para fins descritivos. Conforme apresentado, seu formato é o de um elemento vazio (folha XML), não possuindo filhos em seu interior.

Sequence

O elemento **`sequence`** representa uma das quatro estruturas básicas definidas em Kiepuszewski, Hofstede e Bussler (2000), sendo equivalente à estrutura seqüencial. Pode aceitar várias ocorrências de qualquer estrutura da linguagem em seu interior, excetuando-se `process`. Os elementos colocados dentro de `sequence` serão executados em seqüência, ou seja, na ordem em que estão dispostos no documento.

A figura 3.3 ilustra um modelo gráfico e um trecho de documento XML equivalente, contendo um elemento `sequence` e, em seu interior, algumas atividades.

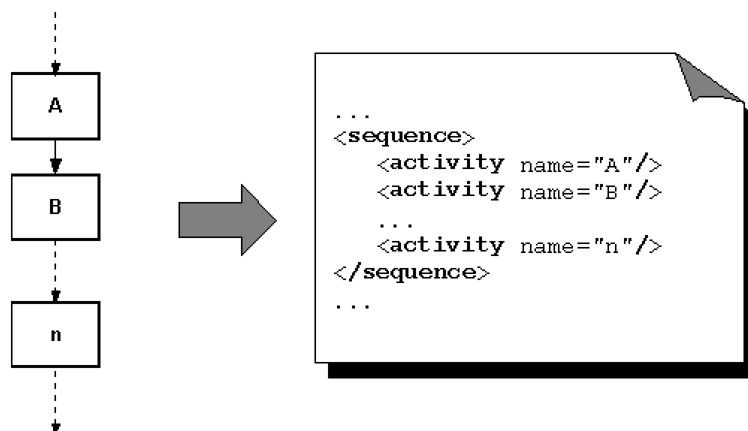


FIGURA 3.3 – Elemento `sequence`

Parallel

O elemento **`parallel`** é equivalente à estrutura paralela dos *workflows* estruturados, sendo responsável pela execução concomitante de seus elementos internos.

Em seu interior, vários elementos podem ser inseridos, assim como acontece com sequence. Porém, neste caso, os filhos de parallel serão executados em paralelo, ao mesmo tempo. Na linguagem XRL, são disponibilizadas três formas de roteamento paralelo: parallel_sync, parallel_part_sync e parallel_no_sync, sendo que parallel_sync é a correspondente a parallel na presente linguagem, uma vez que a classe dos *workflows* estruturados somente modela paralelismo total e as outras formas de paralelismo podem causar ausência de sincronismo.

A figura 3.4 apresenta um roteamento paralelo gráfico e sua estrutura correspondente, que possui algumas atividades, sendo que todas são executadas em paralelo.

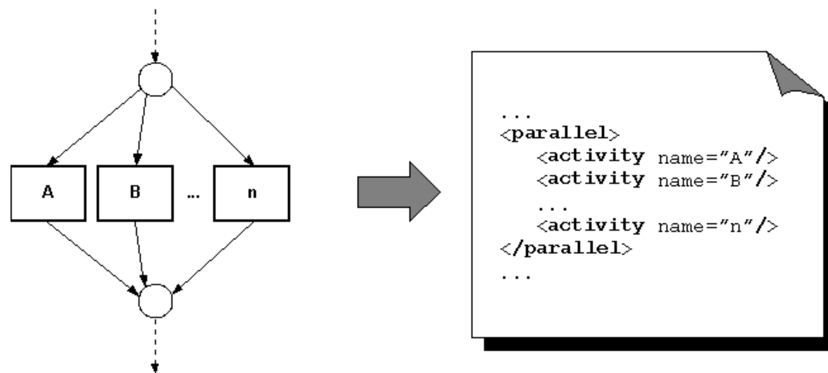


FIGURA 3.4 – Elemento parallel

Conditional

O elemento **conditional** é a representação da estrutura de decisão, e sua semântica é selecionar um dos dois possíveis ramos existentes, a partir de uma condição que deve ser testada.

Sendo assim, esta estrutura possui dois únicos elementos filhos imediatos, **true** e **false**, que habilitam o ramo de execução correspondente em caso de condição verdadeira ou falsa, respectivamente. Dentro de true ou false, somente pode aparecer um dos elementos da linguagem, excetuando-se process. A condição de teste é representada por um atributo, **condition**, que recebe uma expressão booleana como entrada. As linguagens XRL e WQM disponibilizam elementos similares, condition e choice, respectivamente.

A figura 3.5 mostra a representação de uma estrutura condicional, de maneira gráfica e no formato da linguagem.

O elemento conditional é comparável a um IF-THEN-ELSE das linguagens procedimentais. No entanto, na estrutura definida por Kiepuszewski, Hofstede e Bussler (2000), é permitida a habilitação de vários ramos condicionais, ou seja, cada ramo possui uma condição específica que é avaliada. Para o caso da linguagem aqui definida, foi feita uma restrição de funcionalidade, de maneira que existem somente dois ramos possíveis e uma condição que vale para ambos. A representação de múltipla escolha é realizada utilizando-se parallel e conditional em conjunto.

Loop

O elemento **loop** (elemento while_do em XRL e WQM) representa um laço estruturado, no qual o elemento em seu interior será executado repetidamente. Porém, diferentemente das outras linguagens, este elemento executa o conteúdo interno até que uma

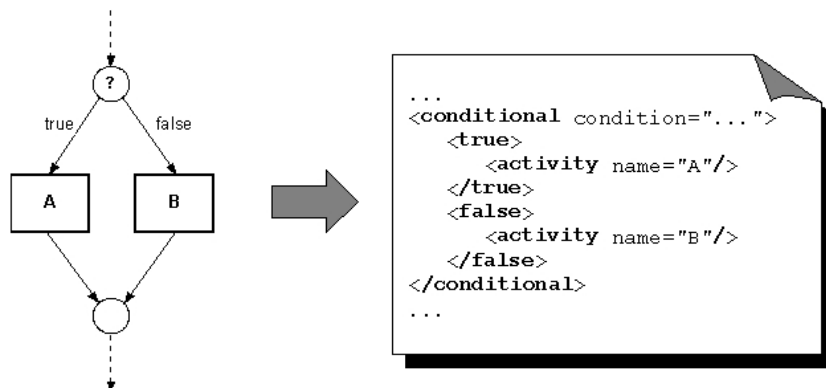


FIGURA 3.5 – Elemento `conditional`

condição seja alcançada, ou seja, testa a saída do laço ao final. Como conseqüência, o laço é executado, no mínimo, uma vez. A condição a ser verificada é representada por um atributo `until`.

A figura 3.6 ilustra a utilização de um elemento `loop`, no qual uma seqüência de atividades será executada repetidamente.

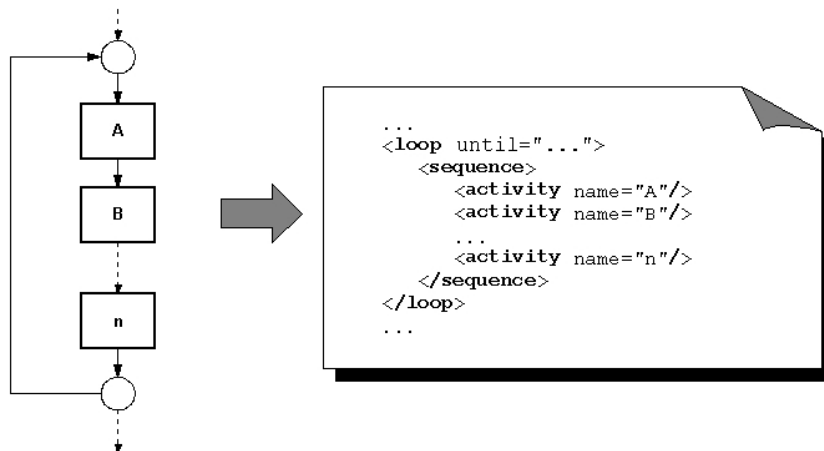


FIGURA 3.6 – Elemento `loop`

Cabe ressaltar que é possível a execução do elemento `loop` como se fosse um laço do tipo WHILE-DO. Para tanto, o elemento `conditional` deve ser utilizado em conjunto com `loop`, testando a condição antes do início do laço de repetição.

Representação de Outras Estruturas de Modelagem

A classe dos *workflows* arbitrários permite, como já mencionado, uma grande flexibilidade para a modelagem dos fluxos de controle. Isso faz com que inúmeras possibilidades surjam, além das formas tradicionais. Em Aalst et al. (2000; 2002), foram catalogados vários **padrões de *workflow***, ou seja, estruturas freqüentemente encontradas nas ferramentas comerciais, que vão além das apresentadas por Kiepuszewski, Hofstede e Bussler (2000), como por exemplo, múltipla escolha (*multi-choice*), junção parcial (*discriminator*), laço arbitrário (*arbitrary cycle*), entre outras. Os padrões são agrupados em categorias, dos mais básicos aos mais complexos.

A linguagem descrita nesta seção contempla apenas os padrões básicos (*sequence*,

parallel split, *synchronization*, *exclusive choice* e *simple merge*), enquanto que os restantes ou são construídos a partir dos básicos ou não possuem representação na linguagem. Como exemplo de transformação entre estruturas, a figura 3.7 apresenta uma estrutura de múltipla escolha condicional (*multi-choice*), que pode ser equivalentemente transformada, utilizando-se as estruturas paralela e condicional.

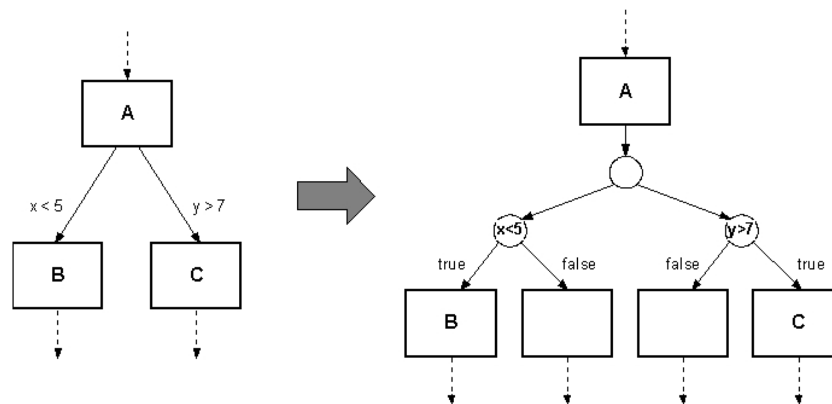


FIGURA 3.7 – Representação de múltipla escolha (AALST et al., 2002)

3.1.4 Exemplo de Modelagem

Nesta seção, é apresentado um exemplo de utilização da linguagem proposta, conforme a figura 3.8. Nesta figura tem-se uma representação gráfica (a) e o esquema XML correspondente (b) de um fluxo hipotético. As informações relativas ao elemento `life-cycle` foram omitidas no documento XML, pois serão tratadas no capítulo 5.

3.1.5 Considerações sobre a Linguagem de Representação de *Workflow*

A linguagem para representação de *workflow* em XML, apresentada ao longo da seção 3.1, respeita as propriedades de correção, simplicidade, clareza e facilidade de alteração, descritas na seção 3.1.1.

A correção de um esquema de *workflow*, baseado na linguagem proposta nesta seção, está intimamente ligada com as regras de boa formação (*well-formedness*) e de validação (*validation*) do documento XML correspondente. Um documento é considerado bem formado se segue a determinadas restrições que normatizam sua forma e/ou aparência (WORLD WIDE WEB CONSORTIUM, 2000). As restrições se adequam perfeitamente ao formato das estruturas da linguagem, apresentadas anteriormente. Restrições como a presença de um único elemento raiz e o encaixe de um elemento inteiramente dentro de outro são algumas dessas regras respeitadas. A validação, por sua vez, é feita comparando-se o documento com a DTD ou *Schema* correspondente, a fim de verificar se o mesmo segue regras de construção específicas. Com a utilização de estruturas, a tarefa de validação se torna muito simples, pois não se permite a ocorrência de estruturas anômalas. Um exemplo de regra de construção é a que define quais elementos podem ser filhos de `conditional`: somente `true` e `false`. Neste caso, a presença de qualquer outro elemento deve resultar num erro de validação, apesar de não causar erros de boa formação.

Existe um grande número de ferramentas que verificam se um documento XML é bem-formado e/ou válido. Assim, não é preciso construir uma arquitetura nem algoritmos

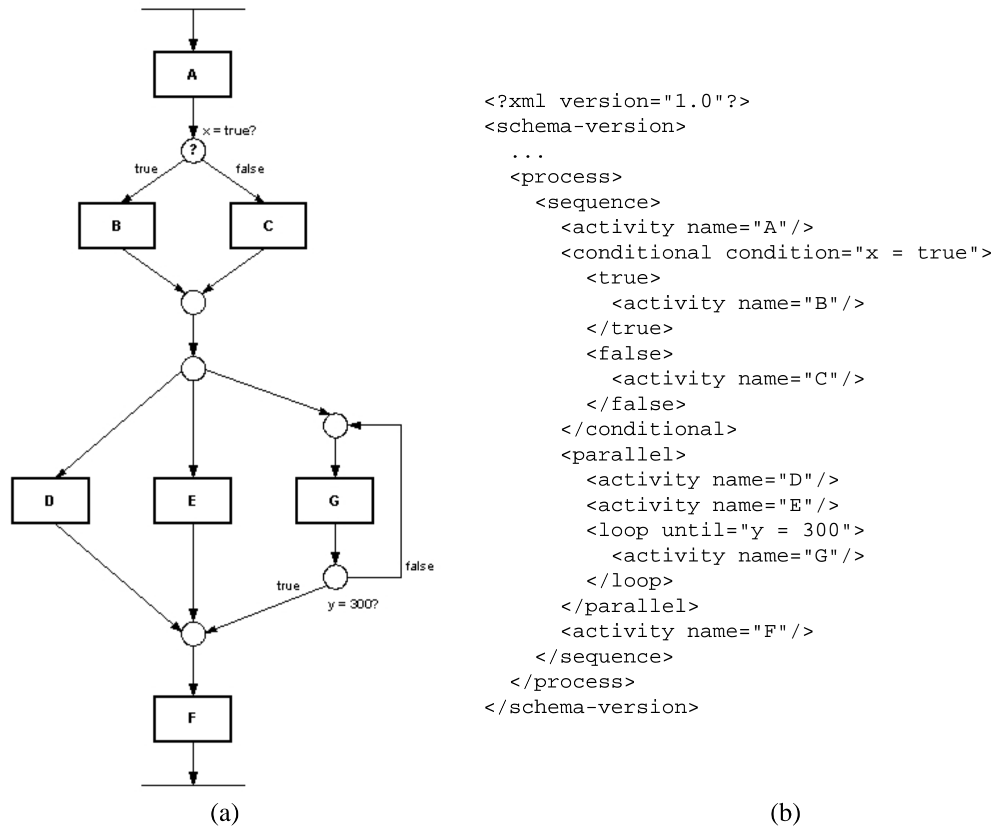


FIGURA 3.8 – Exemplo de um fluxo de controle representado na linguagem proposta

específicos para o processo de correção de esquemas.

No que tange à simplicidade, apesar da pequena quantidade de elementos, os mesmos são suficientes para a modelagem de grande parte dos fluxos reais. Os elementos usados são simples, mas ao mesmo tempo possuem um bom poder de expressão.

O aspecto clareza é alcançado, uma vez que o aninhamento dos elementos permite que as estruturas correspondentes sejam compreendidas, bem como o processo como um todo. É possível perceber qual o tipo de relacionamento existente entre as atividades do *workflow*. Contribui para esse entendimento o fato de que os elementos são bastante intuitivos, sendo quase todos utilizados na construção de fluxogramas tradicionais.

Quanto à facilidade de alteração, na seção 3.2 será mostrado que um *workflow* baseado na linguagem pode ser facilmente alterado mantendo-se ao mesmo tempo correto. Novamente, a chave para que também esta propriedade seja respeitada é o conjunto de estruturas utilizado, que permite alterações profundas no fluxo lidando somente com a posição das estruturas no documento.

Convém, por último, destacar que um *workflow* representado nessa linguagem tem condições de ser transformado em um *workflow* em XPDL, linguagem que representa transições explicitamente e é o padrão adotado pela WfMC. Porém, da mesma forma que *workflows* arbitrários nem sempre podem ser considerados estruturados, um fluxo em XPDL pode não possuir equivalência na linguagem descrita neste trabalho. A transformação para XPDL, entretanto, está fora do escopo desta dissertação.

3.2 Alteração de Esquemas de *Workflow* em XML

O grande objetivo da linguagem para representação de *workflow*, apresentada na seção 3.1, é permitir que alterações possam ser consistentemente efetuadas nos *workflows*, de modo a adequar-se a novas realidades. Sendo assim, é necessário que o WFMS ofereça um mecanismo para que essas alterações possam ser efetuadas de maneira adequada.

Como XML está se tornando um padrão para a representação de dados e troca de informações na *Web*, inúmeras linguagens estão disponíveis para consulta em documentos. No entanto, um aspecto ainda pouco explorado por essas linguagens é a possibilidade das mesmas realizarem modificações em um documento. Portanto, isso se torna um complicador para os *workflows* representados em XML e que necessitem ser modificados.

Algumas propostas de linguagens específicas para modificação de documentos XML estão surgindo para suprir essa lacuna deixada pelas linguagens de consulta. Elas permitem que qualquer parte constituinte de um documento (elementos, atributos, etc.) possa ser modificada, por meio de instruções apropriadas.

A seguir, são apresentados alguns tipos de modificações em documentos XML, além de propostas de linguagem de modificação de documentos, com a finalidade de identificar como as alterações em um esquema de *workflow* podem ser representadas no documento XML correspondente, tomando como base a linguagem apresentada na seção 3.1.

3.2.1 Propostas de Linguagens para Alteração de Documentos XML

Uma linguagem para modificação de documentos XML é um "grande desejo" dos desenvolvedores e profissionais que trabalham com esta tecnologia. No entanto, a W3C, entidade que define padrões para a *Web*, até o momento não incorporou aspectos que permitam alterar documentos na sua linguagem de consulta oficial, a XQuery (WORLD WIDE WEB CONSORTIUM, 2002a), nem apresentou algo específico para preencher esta lacuna.

Assim sendo, foram lançadas algumas propostas de linguagens de modificação de documentos XML, independentemente da W3C. Duas alternativas (extensão de XQuery e XUpdate) são apresentadas a seguir.

3.2.1.1 Extensão de XQuery

Uma das primeiras propostas que surgiu para o tratamento do problema da alteração de documentos XML foi uma extensão da linguagem XQuery, proposta por Tatarinov et al. (2001). A linguagem estendida apresenta, além das cláusulas tradicionais **FOR**, **LET**, **WHERE** e **RETURN** da linguagem XQuery, uma cláusula **UPDATE**, responsável pelo encapsulamento e descrição das modificações a serem realizadas no documento XML.

A cláusula UPDATE aceita as seguintes operações, que podem ser aplicadas sobre elementos ou atributos: **Delete**, **Rename**, **Insert**, **InsertBefore**, **InsertAfter**, **Replace** e **Sub-Update**.

A figura 3.9 apresenta um documento XML que contém uma listagem de amigos (a) e o mesmo documento modificado (b), com a inclusão de um telefone para Jose Silva. A operação necessária, de acordo com a extensão proposta para XQuery, é ilustrada na figura 3.10.

```
<?xml version="1.0"?>
<amigos>
  <amigo>
    <nome>Jose Silva</nome>
    <aniv dia="10" mes="11" ano="1975"/>
    <cidade>Porto Alegre</cidade>
    <pais>Brasil</pais>
  </amigo>
  ...
</amigos>
```

(a)

```
<?xml version="1.0"?>
<amigos>
  <amigo>
    <nome>Jose Silva</nome>
    <aniv dia="10" mes="11" ano="1975"/>
    <fone>3331-3131</fone>
    <cidade>Porto Alegre</cidade>
    <pais>Brasil</pais>
  </amigo>
  ...
</amigos>
```

(b)

FIGURA 3.9 – Documento de exemplo (a) e após uma modificação (b)

```
FOR $amg IN document("amigos.xml")/amigos/amigo[1],
    $brt IN $amg/aniv
UPDATE $amg {
  INSERT <fone>3331-3131</fone> AFTER $brt
}
```

FIGURA 3.10 – Operação necessária para modificação de acordo com a extensão de XQuery

3.2.1.2 Linguagem XUpdate

Outra proposta para permitir a alteração de documentos XML partiu de um grupo, o XML:DB, que definiu uma linguagem denominada **XUpdate** (XML:DB, 2000). Essa linguagem, diferentemente da proposta anterior, não necessita de uma terminologia específica para a declaração das modificações a serem realizadas. XUpdate utiliza a própria linguagem XML como meio de expressão das alterações.

A linguagem permite que várias alterações possam ser efetuadas em um documento, utilizando-se da linguagem XPath para representar expressões de caminho (WORLD WIDE WEB CONSORTIUM, 1999). Ela possibilita as seguintes construções, que também podem ser aplicadas sobre atributos e/ou elementos: `insert-before`, `insert-after`, `append`, `update`, `remove`, `rename`, `variable`, `value-of` e `if` (STAKEN, 2000).

Considerando novamente o exemplo da figura 3.9, para que o documento (a) seja transformado no documento (b), é aplicado o documento XUpdate da figura 3.11.

```
<?xml version="1.0"?>
<xupdate:modifications version="1.0"
  xmlns:xupdate="http://www.xmldb.org/xupdate">

  <xupdate:insert-after select="/amigos/amigo[1]/aniv">
    <xupdate:element name="fone">3331-3131</xupdate:element>
  </xupdate:insert-after>

</xupdate:modifications>
```

FIGURA 3.11 – Operação necessária para modificação de acordo com a linguagem XUpdate

Outra vantagem que favorece a linguagem XUpdate é o fato da mesma já possuir implementações, permitindo que ela possa ser utilizada para a modificação de documentos XML.

As modificações realizadas com XUpdate garantem que o documento continua bem formado; no entanto, podem torná-lo inválido. Por isso, o documento modificado deve ser novamente validado, para que eventuais problemas possam ser detectados e corrigidos.

A partir da análise das propostas, escolheu-se XUpdate como linguagem para a representação das operações de modificação em esquemas de *workflow* em XML pois, além de já possuir implementações, representa as operações no mesmo formato dos "dados" a serem modificados, ou seja, em XML.

3.2.2 Operações de Modificação

Um documento XML pode ser alterado de muitas maneiras, conforme apresentado anteriormente. Entre as principais modificações tem-se inserção, remoção, troca de posição, etc., seja de elementos, atributos ou de conteúdos destes.

Dentre as formas possíveis de modificação de documentos, algumas são importantes, ou seja, fazem sentido, quando o documento representa um esquema de *workflow*. No caso dos esquemas construídos a partir da linguagem apresentada na seção 3.1, o fluxo de controle é determinado pela disposição das estruturas, ou seja, pela posição dos elementos no documento. Portanto, qualquer modificação que venha a alterar a ordem dos elementos teoricamente representa uma alteração no esquema de *workflow*. Por outro lado, os atributos não mantém informações relacionadas ao fluxo de controle em si. Assim sendo, serão consideradas somente as mudanças que tenham, como alvo, elementos.

Dessa maneira, as seguintes operações genéricas sobre elementos (separadas ou em conjunto) são consideradas alterações em esquemas de *workflow*:

- **inserção** – colocação de um novo elemento folha ou nova estrutura em uma determinada localização no documento;
- **remoção** – retirada de um elemento ou estrutura existente no documento;
- **movimentação** – alteração de posição de um elemento ou estrutura existente dentro do documento;
- **troca de estrutura** – mudança do nome de um elemento, ocasionando uma transformação de uma estrutura para outra.

Estas operações genéricas de modificação de *workflows* podem ser implantadas, usando a linguagem XUpdate, utilizando-se as seguintes operações, restritas a elementos XML:

- **`insert-before`**/**`insert-after`** e **`append`**– responsáveis pela inserção de elemento antes/depois de um determinado elemento escolhido e pela inserção no interior de um elemento, como último filho;
- **`remove`** – responsável pela remoção de um determinado elemento escolhido;
- **`variable`** e **`value-of`** – a operação `variable` armazena um elemento em uma variável, enquanto que `value-of` recupera um elemento armazenado. Com essas duas operações, é possível implantar a cópia e a movimentação de elementos, em conjunto com uma das operações de inserção e a operação de remoção (no caso de movimentação);
- **`rename`** – responsável pela troca de nome em um determinado elemento escolhido.

Pelo fato de que XUpdate pode armazenar várias modificações a serem realizadas, o documento que está sendo alterado pode se tornar temporariamente inconsistente após a aplicação de algumas operações. No entanto, a validação do mesmo somente é feita ao final da aplicação de todas as operações de modificação. Se o documento for considerado inválido quanto às regras de sintaxe, a modificação deve ser desfeita integralmente, recuperando-se o esquema original.

Sendo assim, a seguir será feito um estudo da implementação das operações sobre as estruturas da linguagem de representação de *workflow* em XML definida neste trabalho. Considera-se, aqui, que cada operação (inserção, remoção, ...) respeita o princípio do isolamento, ou seja, cada operação é realizada como se fosse a única.

3.2.2.1 Implementação da Operação de Inserção

Qualquer atividade ou estrutura da linguagem pode ser inserida num esquema, à exceção de `process`. No entanto, essa operação causa impactos diferentes, dependendo da estrutura que irá receber o novo elemento:

- `sequence` e `parallel` – a inserção não causa problemas dentro destas duas estruturas, pois elas permitem vários elementos em seu interior;
- `conditional` – esta estrutura somente permite a inserção de um elemento `false` quando este ainda não existir, pois `false` é um elemento opcional;
- `true`, `false`, `loop` e `process` – estes elementos não permitem a inserção de outro elemento, por aceitarem somente um filho imediato.

A figura 3.12 mostra a inserção da atividade H logo após ao elemento `conditional`, tomando como base o esquema XML apresentado na figura 3.8. A inserção é realizada dentro de um elemento `sequence`, que aceita vários elementos filhos.

```
<?xml version="1.0"?>
<xupdate:modifications version="1.0"
  xmlns:xupdate="http://www.xmldb.org/xupdate">

  <xupdate:insert-after select="/schema-version/
    process/sequence/conditional">
    <xupdate:element name="activity">
      <xupdate:attribute name="name">H
      </xupdate:attribute>
    </xupdate:element>
  </xupdate:insert-after>

</xupdate:modifications>
```

(a)

```
<?xml version="1.0"?>
<schema-version>
  ...
  <sequence>
    <activity name="A"/>
    <conditional condition="x = true">
      ...
    </conditional>
    <activity name="H"/>
    <parallel>
      ...
    </parallel>
  </sequence>
  ...
</schema-version>
```

(b)

FIGURA 3.12 – Operação de inserção

3.2.2.2 Implementação da Operação de Remoção

A operação de remoção de elemento causa mais impactos do que a operação de inserção. Cuidados especiais devem ser tomados para que uma estrutura não fique vazia, ou seja, sem elementos internos, o que não é permitido pela sintaxe da linguagem. Assim sendo, seguem observações caso a caso:

- `sequence` – pode sofrer remoção de elementos internos, desde que não se torne um elemento vazio;
- `parallel` – exige um mínimo de dois elementos internos. Portanto, se `parallel` possuir mais do que dois filhos, a remoção pode ser efetuada;
- `conditional` – dos dois elementos internos (`true` e `false`), somente o elemento `false` pode ser removido, por ser opcional;
- `true`, `false`, `loop` e `process` – não permitem remoção, pois possuem somente um elemento filho imediato.

A figura 3.13 apresenta a operação de remoção da estrutura repetitiva presente dentro de `parallel`. Como essa estrutura possui elementos internos, todos são removidos. A retirada desse elemento não viola a estrutura paralela, pois a mesma continua com dois elementos em seu interior.

```
<?xml version="1.0"?>
<xupdate:modifications version="1.0"
  xmlns:xupdate="http://www.xmldb.org/xupdate">

  <xupdate:remove select="/schema-version/
    process/sequence/parallel/loop"/>

</xupdate:modifications>
```

(a)

```
<?xml version="1.0"?>
<schema-version>
  ...
  <parallel>
    <activity name="D"/>
    <activity name="E"/>
    <!-- loop foi removido -->
  </parallel>
  ...
</schema-version>
```

(b)

FIGURA 3.13 – Operação de remoção

3.2.2.3 Implementação da Operação de Movimentação

Esta operação nada mais é do que uma composição das operações de remoção e inserção, explanadas anteriormente. Assim sendo, deve-se levar em consideração as restrições das duas operações em conjunto para que uma movimentação seja realizada com sucesso, ou seja, verifica-se se o elemento não causa inconsistências na remoção e na inserção.

O elemento a ser trocado de lugar deve ser armazenado em uma variável, para depois ser recuperado e inserido. Isso é realizado, em XUpdate, por meio das operações `variable` e `value-of`, respectivamente.

A figura 3.14 ilustra a movimentação da estrutura paralela, que foi colocada imediatamente à frente da estrutura condicional. De acordo com a figura 3.14(a), são utilizadas as operações básicas de inserção e remoção, em conjunto com o armazenamento e recuperação de variáveis. A operação de movimentação não causa transtornos ao documento visto que se processa dentro de um elemento `sequence`.

3.2.2.4 Implementação da Operação de Troca de Estrutura

Essa operação permite que uma estrutura seja transformada em outra. No entanto, esse tipo de operação somente pode ser aplicado em estruturas sintaticamente semelhantes, ou seja, que possuem a mesma estrutura para os seus subelementos. Caso contrário, seriam necessárias adequações extras.

A troca de nome de um elemento somente é permitida nos casos de `sequence` para `parallel` e vice-versa, pois ambos os elementos não possuem atributos e podem conter

```
<?xml version="1.0"?>
<xupdate:modifications version="1.0"
  xmlns:xupdate="http://www.xmldb.org/xupdate">

  <xupdate:variable name="paralelo"
    select="/schema-version/process/sequence/
    parallel"/>
  <xupdate:insert-before select="/schema-version
    /process/sequence/conditional">
    <xupdate:value-of select="$paralelo"/>
  </xupdate:insert-before>
  <xupdate:remove select="/schema-version/process
    /sequence/parallel"/>

</xupdate:modifications>
```

(a)

```
<?xml version="1.0"?>
<schema-version>
  ...
  <parallel>
    <activity name="D"/>
    <activity name="E"/>
    <loop until="y = 300">
      <activity name="G"/>
    </loop>
  </parallel>
  <conditional condition="x = true">
    <true>
      <activity name="B"/>
    </true>
    <false>
      <activity name="C"/>
    </false>
  </conditional>
  ...
</schema-version>
```

(b)

FIGURA 3.14 – Operação de movimentação

vários filhos. Porém, com essa operação, a semântica de execução dos filhos se altera. No caso de um elemento `parallel` passar para `sequence`, os elementos internos não serão mais executados em paralelo, mas em seqüência. De maneira similar, se um elemento `sequence` for renomeado para `parallel`, os elementos devem ser executados em paralelo. Neste último caso, entretanto, devem existir pelo menos dois elementos internos para que essa operação seja considerada válida, uma vez que o elemento `parallel` exige, no mínimo, dois filhos.

A figura 3.15 apresenta a operação de troca da estrutura `parallel` para `sequence`, ou seja, as atividades D e E e o laço de repetição serão executados em seqüência.

```
<?xml version="1.0"?>
<xupdate:modifications version="1.0"
  xmlns:xupdate="http://www.xmldb.org/xupdate">

  <xupdate:rename select="/schema-version/process/
    sequence/parallel">sequence</xupdate:rename>

</xupdate:modifications>
```

(a)

```
<?xml version="1.0"?>
<schema-version>
  ...
  <sequence>
    <activity name="D"/>
    <activity name="E"/>
    <loop until="y = 300">
      <activity name="G"/>
    </loop>
  </sequence>
  ...
</schema-version>
```

(b)

FIGURA 3.15 – Operação de troca de estrutura

3.2.3 Considerações sobre Alteração de Esquemas

Ao longo da seção 3.2, foram apresentadas as modificações que podem ser realizadas sobre um esquema de *workflow* em XML. Além disso, duas linguagens de modificação de documentos foram estudadas, sendo que a linguagem XUpdate apresentou-se como a mais adequada para a implantação das operações genéricas de modificação de esquema.

As modificações nos esquemas são simples e fáceis de serem aplicadas, uma vez que envolvem estruturas. Um exemplo da facilidade em se realizar uma alteração pode

ser vista na figura 3.16. Esta figura ilustra um esquema construído numa linguagem baseada em RET (a) e o mesmo esquema construído de acordo com a linguagem apresentada na seção 3.1 (b). A modificação consiste em mover a atividade B para antes da atividade A. No esquema (a), são necessárias várias atualizações em atributos, o que torna a modificação complexa. Já no esquema (b), somente é necessário inverter as posições das atividades A e B no documento, o que é muito mais simples e natural.

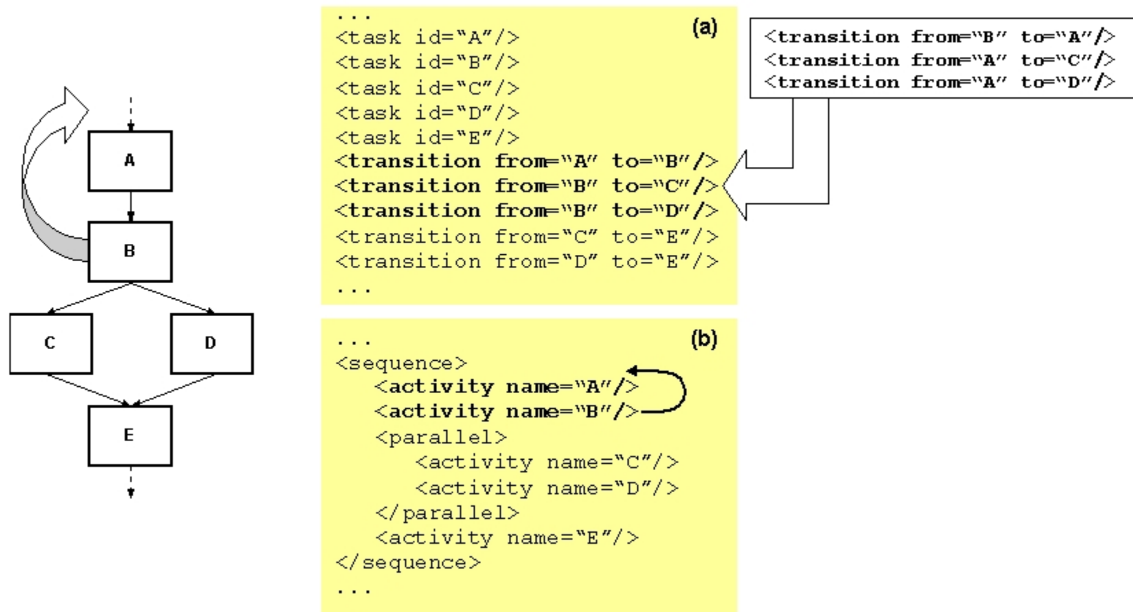


FIGURA 3.16 – Facilidade de alteração de esquema de *workflow* de acordo com a linguagem de representação

Conforme foi discutido, qualquer modificação no documento XML teoricamente equivale a uma alteração do esquema correspondente. Na prática, entretanto, podem acontecer casos em que o esquema não foi efetivamente modificado (caso 1) ou algumas operações aplicadas sobre certas estruturas não modificam a semântica das mesmas (caso 2).

Um exemplo que ilustra o primeiro caso é a aplicação de operações que anulam umas às outras, como por exemplo, uma inserção e uma posterior remoção do elemento previamente inserido.

O caso 2 acontece principalmente com a operação de movimentação. Em algumas estruturas, a ordem dos elementos internos não faz diferença. Exemplos são os elementos `parallel` e `conditional`. O elemento `parallel` possui vários subelementos, que podem estar em qualquer ordem, pois são executados em paralelo. Assim, uma alteração de posição entre eles não deve significar uma alteração no esquema. O elemento `conditional`, por sua vez, possui `true` e `false` como filhos. Da mesma forma, não importa a ordem desses dois elementos entre si e uma alteração de posição não implica em modificação.

A situação ilustrada no caso 2, entretanto, difere do caso 1, pois uma troca de posição somente se refere a um trecho do documento, enquanto que a anulação completa das operações relaciona-se com o documento inteiro.

Portanto, ao se aplicar as operações de modificação sobre um esquema, o WFMS

deve ser capaz de verificar se aconteceu realmente uma modificação e, caso contrário, avisar que nada foi realizado. Neste trabalho será considerado que o responsável pela alteração do esquema aplica as operações de modificação de maneira correta, sem gerar "falsas" atualizações.

3.3 Considerações Finais

Este capítulo apresentou uma linguagem para especificação de esquemas de *workflow* em XML. A linguagem foi desenvolvida como formato para representação de esquemas que pudessem evoluir no tempo, refletindo novos requisitos ou corrigindo erros existentes. Quatro aspectos principais foram levados em consideração para que esse objetivo fosse alcançado: correção, simplicidade, clareza e facilidade de alteração. Foi mostrado que as outras propostas de representação de *workflow* em XML não se encaixavam, de alguma forma, com o objetivo definido, fator que motivou a definição desta linguagem.

A linguagem para especificação de *workflow* se baseou nos *workflows* estruturados, que são constituídos de estruturas básicas e que apresentam muita semelhança com a sintaxe XML. Dessa maneira, a linguagem é composta das quatro estruturas básicas além da atividade propriamente dita. Outras estruturas podem ser obtidas através de transformações ou composições de estruturas básicas.

Além disso, foram especificadas algumas operações a serem aplicadas sobre os esquemas, possibilitando a evolução dos mesmos. A linguagem XUpdate, neste sentido, foi escolhida para a implantação das operações genéricas. Foi também efetuado um estudo da aplicação das operações de modificação sobre as estruturas da linguagem de representação. Basicamente, constatou-se que a maioria das estruturas não aceita modificações isoladas, ou seja, que não sejam complementadas com outras operações, em razão das restrições sintáticas da linguagem. No entanto, a aplicação de um conjunto atômico de modificações pode ser realizado, mesmo que em alguns momentos o esquema fique temporariamente inconsistente.

4 Representação e Migração de Instâncias

Um esquema de *workflow* é modelado para refletir determinado processo de negócio do mundo real. Atividades são arranjadas da melhor maneira, participantes são definidos para elas e informações fluem por toda essa estrutura. Entretanto, um esquema nada mais é do que um molde, que deve ser capaz de gerar instâncias. As instâncias, como já destacado na seção 2.1, são representações de casos particulares que seguem o formato do esquema, sendo responsáveis pela efetiva execução das atividades.

Algumas das linguagens apresentadas na seção 2.3 são orientadas a instâncias, ou seja, cada documento representa um caso particular. Isso acontece especialmente com a linguagem XRL, que também segue a abordagem RIT como a linguagem definida na seção 3.1. No entanto, esta última tem como diferencial a definição de esquemas de *workflow*, ao invés de modelar os casos individualmente.

É necessário que as instâncias que serão geradas a partir do esquema possuam uma representação adequada, de forma que seja possível recuperar tudo o que ocorreu com as mesmas durante o tempo em que estiveram ativas. Além disso, em virtude da evolução de esquemas, essa representação deve ser capaz de permitir a migração de instâncias para uma nova representação.

A seguir, esses aspectos são abordados em maior profundidade. A seção 4.1 enfatiza aspectos de representação de instâncias de *workflow* baseadas na linguagem definida anteriormente. Na seção 4.2, é proposto um método para a migração de instâncias para outro esquema, baseado em regras de compatibilidade. Finalmente, a seção 4.3 apresenta algumas considerações finais sobre o capítulo.

4.1 Representação das Instâncias

Uma instância de um esquema de *workflow* em XML é gerada a partir de uma operação *createInstance*, aplicada sobre um determinado esquema. Dessa maneira, tem-se como resultado uma representação específica para um caso particular, que deve obedecer o fluxo determinado pelo esquema.

Cada instância deve possuir um identificador único, de maneira a preservar a sua identidade frente às outras instâncias. Este identificador não pode ser alterado durante a existência da instância, nem mesmo na hipótese de migração ou adaptação para outro esquema, por meio de uma evolução.

Por se tratar de um objeto (no sentido mais amplo) que é dinâmico, ou seja, que possui atividades executando a cada momento, também se faz necessário armazenar o andamento da instância durante o tempo. Uma consulta instantânea a uma determinada instância deve revelar quais atividades estão executando, quais terminaram com sucesso, entre outras possibilidades. Além disso, estando em um ambiente evolutivo, a instância deve manter informações sobre os esquemas que lhe serviram de base durante o período em que a mesma esteve em execução. Dessa maneira, é possível recuperar o ciclo de vida de cada caso, desde o seu surgimento até o término de sua execução.

Sendo assim, deve-se levar em consideração as seguintes características, para a representação de instâncias:

- a instância deve ser compatível com um esquema de base, ou seja, deve seguir o mesmo fluxo determinado por ele;

- a instância deve armazenar o andamento das atividades, por meio de um conjunto de estados;
- em virtude da evolução de esquemas, a instância deve armazenar informações referentes aos esquemas nos quais ela se baseou.

A estratégia adotada neste trabalho para a representação de instâncias baseia-se no próprio esquema que as sustenta. Em outras palavras, cada instância será representada por um documento XML particular, similar ao esquema, contendo informações específicas a cada caso. A seguir, esta estratégia é detalhada, sendo destacadas cada uma das características apresentadas anteriormente.

4.1.1 Estrutura Básica

Uma instância, como destacado previamente, deve ser compatível com o esquema de base. O fluxo de controle determinado pelo esquema deve ser seguido pelas instâncias. Sendo assim, uma alternativa para a representação das mesmas é isolar cada instância em um documento XML próprio, contendo o fluxo de controle. Além disso, informações específicas de cada instância também devem ser consideradas, referentes ao andamento das mesmas no tempo.

Dessa maneira, tem-se os seguintes passos quando da criação de uma instância:

1. cria-se um novo documento XML e embute-se a seguinte estrutura preliminar:

   ```
   <instance id="...">
     <schema-ref>
       <version name="..." itime="..."/>
     </schema-ref>
   </instance>
   ```

   O elemento `instance` é o elemento raiz do documento de instância, contendo um atributo `id` que identifica unicamente a mesma. O valor desse atributo é fornecido pelo WFMS, para que um mesmo identificador não seja utilizado em dois documentos diferentes. O elemento `instance` possui, inicialmente, um filho, `schema-ref`, responsável pelo armazenamento das referências aos esquemas nos quais a instância se baseará durante sua execução (seção 4.1.3).

2. o elemento `process`, presente no esquema, é copiado para o documento de instância, como irmão de `schema-ref`;

3. para cada elemento `activity` presente no interior de `process`, é adicionado um atributo `status`, que mostra o que está ocorrendo com a atividade num certo momento (seção 4.1.2).

É importante que o documento possa armazenar todo o andamento da instância, ou seja, todas as execuções de atividades. Entretanto, surge um caso especial, que merece atenção. O elemento `loop` define que o seu conteúdo seja executado n vezes ($n \geq 1$). No caso desta estrutura ser executada mais de uma vez para uma mesma instância, não haveria como armazenar na instância todas as execuções de atividades, pois os valores dos atributos dinâmicos seriam sobrescritos a cada volta do laço. Isso inviabilizaria, sobretudo, o armazenamento da história da instância. Uma solução para este problema é a inserção de um elemento adicional `lap`, várias vezes dentro da estrutura de repetição,

cada uma delas representando uma volta do laço. Assim, o elemento a ser repetido seria replicado em todas as ocorrências, possibilitando a representação das voltas do laço de repetição. A figura 4.1 ilustra a representação de uma estrutura de repetição em um esquema (a) e numa instância (b), sendo executada n vezes.

```
...
<loop until="...">
  <activity name="X"/>
</loop>
...
```

(a)

```
...
<loop until="...">
  <lap> <!-- 1 -->
    <activity name="X" status="..."/>
  </lap>
  <lap> <!-- 2 -->
    <activity name="X" status="..."/>
  </lap>
  ...
  <lap> <!-- n -->
    <activity name="X" status="..."/>
  </lap>
</loop>
...
```

(b)

FIGURA 4.1 – Representação do elemento `loop` em um esquema (a) e em uma instância (b)

A inclusão de elementos `lap` dentro da estrutura de repetição violaria o critério de correção desta estrutura, se os mesmos estivessem presentes em um esquema. No entanto, como o elemento `lap` está presente somente em instâncias, não ocorre nenhum tipo de erro, uma vez que a instância não necessita ser verificada quanto à correção.

4.1.2 Estados das Atividades

As atividades, representadas na linguagem de representação de *workflow* pelo elemento `activity`, são as partes "ativas" do esquema, ou seja, são executadas pelas instâncias. Uma atividade pode passar por diversas fases, que podem ser representadas por estados, caracterizando momentos distintos durante a execução da instâncias. Neste trabalho, foram definidos os seguintes estados:

- **not-ready** – é um estado inicial, no qual a atividade não está pronta para executar;
- **ready** – neste estado, a atividade está pronta para ser executada;
- **discarded** – a atividade foi descartada, não tendo sido iniciada sua execução. É um estado final, ou seja, a atividade não pode alcançar outro estado quando tiver alcançado o estado discarded;
- **running** – a atividade está executando. Quando a mesma entra neste estado, o tempo é anotado, correspondendo ao início da execução da atividade. Da mesma maneira, quando a atividade sai do estado running, o tempo é finalizado;
- **completed** – neste estado, a atividade terminou sua execução. Da mesma forma que o estado discarded, completed também é um estado final.

A figura 4.2 ilustra um diagrama, contendo os estados das atividades anteriormente apresentados, e as ligações entre eles.

Os estados pelos quais uma atividade passa são representados nas instâncias por meio do atributo `status`, que é anexado nos elementos `activity`, conforme apresentou

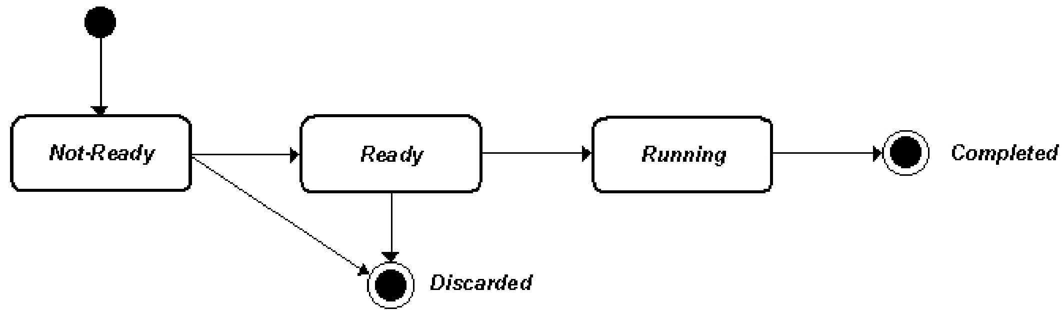


FIGURA 4.2 – Diagrama de estados das atividades

a figura 4.1(b). Esse atributo é não temporal, ou seja, não armazena todos os estados pelos quais a atividade passou, mas somente guarda o estado atual. Além desse atributo, podem estar presentes também os atributos `itime` e `ftime`. Eles armazenam os tempos inicial e final, respectivamente, em que a atividade esteve em execução, ou seja, durante o estado running.

No momento da criação de uma instância, todas as atividades são automaticamente definidas como not-ready. Quando a primeira atividade a ser executada estiver pronta, ela passa para o estado ready e, posteriormente, para running. Quando a atividade finalizar sua execução, é passada para o estado completed, que habilita as próximas atividades a mudarem seus estados para ready.

A passagem para o estado running é considerada a regra, mas em alguns casos nem todas as atividades que estão prontas são executadas. Isso acontece no caso do elemento `conditional`, que é uma estrutura de desvio e sempre possuirá um ramo que não será executado. Assim, é necessário que se marque adequadamente as atividades que não foram executadas. Para tanto, é utilizado o estado discarded, indicando que uma atividade (ou um conjunto de atividades) foi desconsiderada.

Pode-se deduzir, então, que os estados podem ser analisados sob dois ângulos, em relação à atividade: **interno** e **externo**. O aspecto interno diz respeito ao estado como uma informação única e exclusiva da atividade em questão, indicando o andamento da mesma no tempo. Já o aspecto externo refere-se ao efeito que uma mudança de estado em uma atividade produz em outra. Como exemplo, pode-se citar uma atividade que completou sua execução, o que possibilita que uma atividade posterior possa ser declarada pronta (permite uma mudança para o estado ready).

O término da execução de uma estrutura habilita a próxima estrutura na instância. A seguir, é considerada a passagem de estados de acordo com as estruturas da linguagem:

- `sequence` – nesta estrutura, as atividades são executadas na ordem em que estão dispostas. Assim, cada atividade deve passar pelos estados not-ready, ready, running e completed, de maneira que, quando uma atividade atinge o estado completed, a próxima pode passar de not-ready para ready, e assim sucessivamente até o final da estrutura;

- `parallel` – quando a estrutura paralela se torna apta, habilita todos os ramos, passando as atividades para o estado ready. O término da estrutura paralela somente se dará quando todas as atividades dos ramos paralelos estiverem concluídas;

- `conditional` – somente um ramo (`true` ou `false`) pode executar suas respectivas atividades. As atividades que pertencem ao ramo que não foi escolhido devem

ser marcadas com o estado discarded. A estrutura é considerada concluída quando as atividades do ramo escolhido tiverem finalizado;

- `loop` – a estrutura de repetição exige mais cuidados que as anteriores, pois cresce progressivamente de tamanho, de acordo com o número de repetições que são realizadas. No entanto, um elemento `lap` sempre será colocado, no mínimo, por ocasião da criação da instância. O elemento interno será executado e a expressão deve ser testada ao final do laço. Se a condição for avaliada como falsa, então outro elemento `lap` é inserido na instância e a primeira atividade é habilitada. Caso a condição seja avaliada como verdadeira, o laço é terminado e a próxima atividade externa ao laço é habilitada.

4.1.3 Informações sobre os Esquemas de Base

Outro importante aspecto a ser considerado na formação das instâncias é o armazenamento de informações decorrentes da evolução de esquemas. Geralmente, quando um esquema é modificado, gerando um novo esquema, suas instâncias devem "migrar" para a nova representação (o que será apresentado na seção 4.2). Para efeitos históricos, é importante saber quais esquemas serviram de suporte para a instância durante seu período de atividade. Portanto, essa informação é própria de cada instância, visto que são independentes umas das outras.

Na seção 4.1.1, foi apresentado o elemento `schema-ref`, que é responsável por este tipo de tarefa. Cada mudança de esquema de base é registrada num elemento `version`, que indica à qual versão de esquema a instância está "presa" no momento (o versionamento de esquemas será discutido no capítulo 5). Dessa maneira, se a instância migrar n vezes para outros esquemas, estarão presentes n ocorrências do elemento `version`.

O elemento `version` contém alguns atributos: `name`, `itime` e `ftime`. O atributo `name` é responsável por armazenar o nome do documento XML que representa o esquema de *workflow* no qual a instância se baseia (ou se baseou). Já os atributos `itime` e `ftime` denotam o tempo inicial e final, respectivamente, que a instância esteve "presa" ao esquema.

Pode-se constatar, então, que `schema-ref` é um elemento temporal da instância, pelo fato de armazenar todas as referências aos esquemas durante o tempo. Com isso, pode-se recuperar todo o ciclo de vida da instância, desde a sua criação até o término de sua execução.

4.1.4 Exemplo de Instância

A figura 4.3 ilustra um exemplo de uma instância de um esquema hipotético v1, em um instante do tempo. É importante notar que a instância ainda não foi migrada para outro esquema, visto que só existe uma referência `version`. Além disso, a estrutura condicional, de acordo com esta visão instantânea, não pode ser considerada pronta e começar a executar, pois antes dela existe uma estrutura paralela que ainda não está totalmente finalizada.

```
<?xml version="1.0"?>
<instance id="1">
  <schema-ref>
    <version name="v1.xml" itime="5"/>
  </schema-ref>
  <process>
    <sequence>
      <activity name="A" status="completed" itime="7" ftime="9"/>
      <conditional condition="...">
        <true>
          <activity name="B" status="completed" itime="10" ftime="13"/>
        </true>
        <false>
          <activity name="C" status="discarded"/>
        </false>
      </conditional>
      <parallel>
        <activity name="D" status="running" itime="14"/>
        <activity name="E" status="ready"/>
        <loop until="...">
          <lap>
            <activity name="G" status="completed" itime="14" ftime="15"/>
          </lap>
        </loop>
      </parallel>
      <conditional condition="...">
        <true>
          <activity name="K" status="not-ready"/>
        </true>
        <false>
          <activity name="W" status="not-ready"/>
        </false>
      </conditional>
    </sequence>
  </process>
</instance>
```

FIGURA 4.3 – Exemplo de uma instância

4.2 Migração de Instâncias para outro Esquema

Conforme destacado anteriormente, uma instância, durante todo o seu tempo de vida, deve basear sua execução em algum esquema de *workflow* existente. Quando ocorrem mudanças no esquema e as instâncias necessitam migrar para outro esquema, a mesma regra continua valendo, ou seja, a instância deve continuar sua execução, mas agora com base no novo esquema criado.

Sendo assim, é correto dizer que uma determinada instância de *workflow* deve ser sempre **compatível** com um esquema de *workflow*. Conforme Casati et al. (1998), uma instância é compatível com um esquema se a mesma executou um dos possíveis caminhos desse esquema.

Segundo Horn e Jablonski (1998), existem dois locais a serem considerados quando da evolução de esquemas e da migração de instâncias: o **ponto de modificação** (esquema) e o **ponto de execução** (instância). Em regras gerais, se o ponto de execução de uma instância de *workflow* é anterior ao ponto de modificação de um esquema, então a instância pode ser migrada para o esquema. Caso contrário, serão necessárias adequações, como *rollback* ou tarefas de compensação.

Para o caso da linguagem de representação de *workflow* em XML apresentada neste trabalho, é necessária a definição de mecanismos de verificação da compatibilidade entre instâncias e esquemas, uma vez que o formato de representação do fluxo baseia-se em

estruturas hierárquicas.

No contexto deste trabalho, são propostos dois tipos de compatibilidade: a **compatibilidade total** e **compatibilidade parcial**, que serão explanados nas seções 4.2.1 e 4.2.2, respectivamente. Após, é explicitada a forma como as instâncias podem migrar para outros esquemas.

4.2.1 Compatibilidade Total

A compatibilidade total é definida como sendo a igualdade entre os fluxos de controle de uma instância e de um esquema quaisquer. Uma instância é considerada totalmente compatível com um esquema quando possui a mesma disposição de estruturas presente no esquema. Em outras palavras, se num determinado momento é verificado que uma instância é totalmente compatível com um esquema, então garante-se que aquela executou e continuará executando suas atividades de acordo com o fluxo definido pelo esquema, considerando que não haja uma evolução no futuro.

Para que a compatibilidade total entre uma instância e um esquema seja verificada, é necessário realizar uma comparação entre os respectivos documentos. No entanto, facilmente se nota que os documentos de esquema e de instância são diferentes entre si, por possuirem, além do processo, informações específicas. Dessa maneira, para que a comparação seja possível, se faz necessário que alguns elementos e/ou atributos sejam retirados de ambos os documentos. Utiliza-se, então, uma representação auxiliar, em forma de árvore, para que as informações dos documentos não sejam perdidas. As árvores podem ser geradas utilizando-se o modelo de objetos DOM (*Document Object Model*), definido pela W3C, que permite a representação hierárquica de um documento XML em memória, possibilitando a manipulação dos nodos por meio de métodos próprios (WORLD WIDE WEB CONSORTIUM, 2002b).

A seguir, é apresentado um algoritmo para a verificação de compatibilidade total entre uma instância e um esquema de *workflow* em XML:

1. cria-se uma estrutura de árvore tanto para o esquema quanto para a instância. Nestas árvores serão realizadas adequações para verificação da compatibilidade total;
2. na árvore do esquema:
   (a) remove-se a subárvore cuja raiz é `life-cycle`;
   (b) retira-se o nodo `schema-version`, tomando o seu lugar a subárvore cuja raiz é `process`;
3. na árvore da instância:
   (a) remove-se a subárvore cuja raiz é `schema-ref`;
   (b) retira-se o nodo `instance`, sendo que a subárvore cuja raiz é `process` ocupa seu lugar;
   (c) para todos os nodos `activity`, são removidos os atributos `status`, `itime` e `ftime`, se existirem;
   (d) para cada nodo `loop`:
      i. sendo n o número de subárvores cuja raiz é `lap`, remove-se as primeiras $n - 1$ dessas subárvores;

ii. na subárvore `lap` restante do passo anterior, remove-se o nodo `lap`, ocupando o seu lugar a subárvore desse nodo;

4. compara-se as árvores finais. Se são idênticas, então a instância é totalmente compatível com o esquema, caso contrário, a compatibilidade não é verificada.

Basicamente, a compatibilidade total verifica se os elementos `process`, tanto da instância quanto do esquema, são idênticos, desconsiderando outros elementos, como ciclo de vida de esquema e instância, por exemplo.

4.2.2 Compatibilidade Parcial

A compatibilidade parcial, diferentemente da compatibilidade total, analisa se uma determinada instância é idêntica a um esquema até o ponto em que a mesma está executando no momento. Na compatibilidade total, verifica-se a igualdade passada, presente e futura dos fluxos de controle, enquanto que na compatibilidade parcial, somente os momentos passado e presente são considerados. Em outras palavras, neste tipo de compatibilidade, os fluxos devem ser os mesmos até o momento da verificação, nada garantindo-se sobre o futuro. Nota-se, portanto, que o momento em que a verificação é realizada é importante, pois o conjunto de atividades que estão executando ou já terminaram se modifica no tempo.

Como no caso da compatibilidade total, segue um algoritmo para a obtenção das árvores do esquema e da instância:

1. da mesma maneira que o algoritmo de compatibilidade total, duas estruturas hierárquicas são criadas (para instância e esquema);

2. na árvore do esquema:

   (a) remove-se a subárvore cuja raiz é `life-cycle`;

   (b) retira-se o nodo `schema-version`, tomando o seu lugar a subárvore cuja raiz é `process`;

3. na árvore da instância:

   (a) remove-se a subárvore cuja raiz é `schema-ref`;

   (b) retira-se o nodo `instance`, sendo que a subárvore cuja raiz é `process` ocupa seu lugar;

   (c) os nodos `activity` cujos atributos `status` são iguais a not-ready, ready ou discarded devem ser removidos (restam somente as atividades em execução ou concluídas);

   (d) nos nodos `activity` restantes, são removidos os atributos `status`, `itime` e `ftime`, se existirem;

   (e) para cada nodo `loop`:

      i. sendo n o número de subárvores cuja raiz é `lap`, remove-se as primeiras $n - 1$ dessas subárvores;

      ii. na subárvore `lap` restante do passo anterior, remove-se o nodo `lap`, ocupando o seu lugar a subárvore desse nodo;

   (f) as estruturas que porventura se tornarem vazias devem ser removidas.

Com as árvores definidas, aplica-se um algoritmo de percurso a ambas, para que as mesmas sejam comparadas nodo a nodo. Basicamente, compara-se um nodo da árvore da instância com o respectivo nodo na árvore do esquema, até o final da árvore da instância. O algoritmo de percurso **pré-ordem** foi o escolhido, pois avalia uma árvore da raiz para as folhas, da esquerda para a direita, sendo que o fluxo de controle também está organizado dessa forma no documento. Assim sendo, se os nodos da árvore da instância possuem correspondência na árvore do esquema, então a instância é considerada parcialmente compatível com o esquema. Caso contrário, ela é considerada incompatível.

Entretanto, o algoritmo de percurso em pré-ordem não pode ser aplicado diretamente em todos os nodos da árvore. Dois deles necessitam de avaliação especial: `conditional` e `parallel`. No caso da estrutura condicional, um dos ramos (`true` ou `false`) é removido, pois as suas atividades foram marcadas como discarded. Assim, é necessário que se verifique qual alternativa permaneceu e realizar o percurso adequado nas duas árvores. Por sua vez, a estrutura paralela necessita mais cuidados, pois quando inicia a executar, habilita vários filhos ao mesmo tempo (estado ready). Portanto, se a estrutura paralela estiver ativa no momento da avaliação, é provável que certas atividades ainda não iniciaram, deixando a estrutura incompleta. Por causa disso, o percurso em pré-ordem do nodo `parallel` não refletiria a ordem correta de execução das atividades.

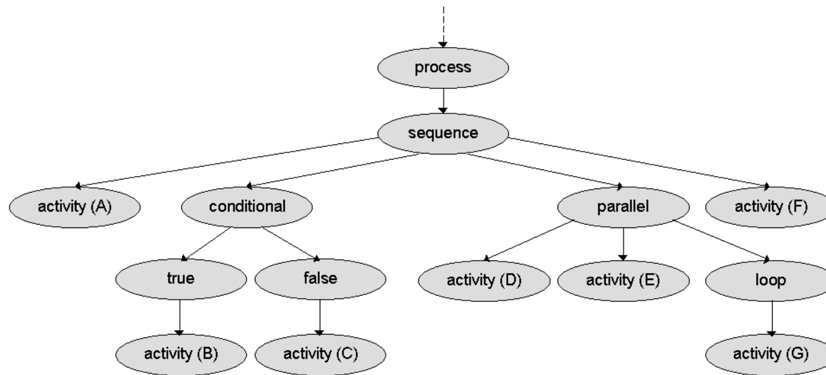
Uma solução para esse problema é a aplicação de n chamadas paralelas de pré-ordem, sendo n o número de subárvores da estrutura paralela da instância. Cada subárvore é percorrida separadamente e comparada à respectiva subárvore do esquema, da mesma maneira como para as outras estruturas. A avaliação da estrutura paralela da instância é concluída quando todas as suas subárvores forem percorridas e não existirem diferenças entre os nodos da instância comparados com os do esquema.

Somente a garantia de que a estrutura paralela incompleta da instância reflete a estrutura paralela do esquema não assegura que a instância é compatível com o esquema. Se o algoritmo de percurso em pré-ordem conseguir detectar a presença de outros nodos após a avaliação da estrutura paralela incompleta, pode-se dizer que a instância é incompatível com o esquema, visto que a estrutura paralela da instância não foi terminada adequadamente segundo o esquema.

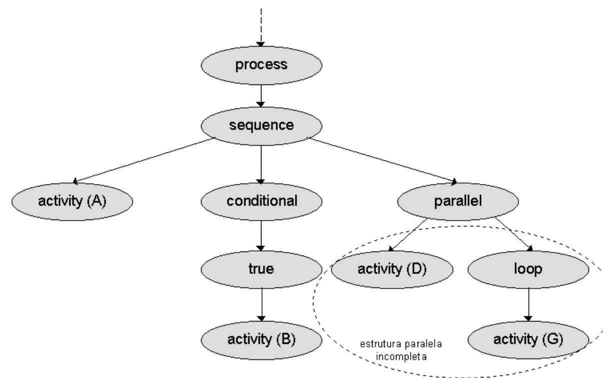
Para melhor ilustrar o processo de compatibilidade parcial, será realizada a verificação entre a instância da figura 4.3 e o esquema apresentado na figura 3.8(b) (página 38). Após a aplicação do algoritmo apresentado nesta seção, obtém-se as árvores do esquema e da instância, conforme a figura 4.4. A comparação das árvores é realizada, sendo constatada a presença de uma estrutura paralela incompleta na instância (a atividade E está ausente). Apesar disso, a instância é considerada parcialmente compatível com o esquema, visto que o fluxo foi o mesmo até aquela estrutura e não há nenhum nodo posterior a `parallel` na árvore da instância. Se, por outro lado, a estrutura condicional tivesse sido executada ou estivesse em execução no momento da verificação, a respectiva subárvore deveria aparecer na árvore da figura 4.4(b), tornando a instância incompatível com o esquema, pois a estrutura paralela (que é anterior à estrutura condicional) seria considerada concluída.

4.2.3 Migração de Instâncias Compatíveis

No início da seção 4.2, foram apresentadas as condições para que uma determinada instância pudesse ser migrada para outro esquema de *workflow*. Basicamente, verifica-se se as atividades já executadas pela instância correspondem a um possível caminho permitido pelo esquema. Por sua vez, a compatibilidade parcial, discutida na seção 4.2.2, tem



(a) Esquema



(b) Instância

FIGURA 4.4 – Árvores do esquema e da instância

como objetivo verificar se as atividades já concluídas ou em andamento da instância possuem correspondência no esquema, não importando as atividades que foram descartadas ou que ainda não iniciaram sua execução.

Dessa forma, pode-se concluir que se for verificada a compatibilidade parcial entre uma instância e um esquema quaisquer, então a migração pode ser estabelecida. Para que uma instância I seja migrada de um esquema E_1 para um esquema E_2, são necessários os seguintes passos, após verificada a compatibilidade parcial entre I e E_2:

1. o elemento `version`, correspondente a E_1, deve ter seu tempo final definido, no documento da instância;

2. um novo elemento `version` deve ser inserido, relativo a E_2, com o atributo `itime` recebendo o tempo em que a instância começou a se basear nesse esquema;

3. as atividades que ainda não iniciaram suas execuções (ready ou not-ready) devem ser removidas do documento de instância. Em seu lugar, são adicionadas as estruturas de E_2 que permitem a continuação da execução de I;

4. as atividades que estavam em execução no momento da migração podem terminar e novas atividades podem ser disparadas, já colocadas em I. Em especial, se

existir uma estrutura paralela incompleta, as atividades ausentes podem iniciar suas execuções.

Assim sendo, a figura 4.5 ilustra a instância representada originalmente pela figura 4.3, devidamente modificada de acordo com os passos anteriormente descritos, de forma que seja adequada ao esquema da figura 3.8 (para fins de ilustração, tal esquema foi nomeado como `v2.xml`).

```
<?xml version="1.0"?>
<instance id="1">
  <schema-ref>
    <version name="v1.xml" itime="5" ftime="16"/>
    <version name="v2.xml" itime="16"/>
  </schema-ref>
  <process>
    <sequence>
      <activity name="A" status="completed" itime="7" ftime="9"/>
      <conditional condition="...">
        <true>
          <activity name="B" status="completed" itime="10" ftime="13"/>
        </true>
        <false>
          <activity name="C" status="discarded"/>
        </false>
      </conditional>
      <parallel>
        <activity name="D" status="running" itime="14"/>
        <activity name="E" status="ready"/>
        <loop until="...">
          <lap>
            <activity name="G" status="completed" itime="14" ftime="15"/>
          </lap>
        </loop>
      </parallel>
      <activity name="F" status="not-ready"/>
    </sequence>
  </process>
</instance>
```

FIGURA 4.5 – Instância migrada para outro esquema

4.2.4 Adaptação de Instâncias Incompatíveis

Nas seções anteriores, se verificou como as instâncias compatíveis com o novo esquema podem migrar para o mesmo. Entretanto, uma parcela das instâncias está excluída desse processo, ou seja, aquelas que não são parcialmente compatíveis. Dependendo da situação, é necessário que todas as instâncias migrem para uma nova representação, pois o esquema antigo não será mais válido.

No caso de incompatibilidade, a migração natural não é possível, pois os fluxos da instância e do esquema não são os mesmos. Assim, é necessário adaptar as instâncias incompatíveis, para torná-las “migráveis”. Alternativas para essa adaptação podem ser *rollback* (desfazer atividades), reinício da instância ou a aplicação de tarefas de compensação. O *rollback* não pode ser aplicado em atividades que não podem ser desfeitas, como no caso de processos materiais. O reinício de toda a instância, por sua vez, poderia desperdiçar muito trabalho já realizado. Já a aplicação de tarefas de compensação se caracteriza por ser essencialmente manual, o que demanda intensa participação do administrador do *workflow*. Apesar disso, essas alternativas são as mais usadas em ambientes reais de adaptação.

Neste trabalho são consideradas somente as instâncias que podem migrar diretamente para o novo esquema, ou seja, que são parcialmente compatíveis com o mesmo.

4.3 Considerações Finais

Este capítulo apresentou uma estrutura de representação de instâncias, baseadas na linguagem de modelagem de *workflow* descrita no capítulo 3. As instâncias são cópias dos esquemas a partir dos quais foram criadas, com a inclusão de atributos especiais que indicam o estado de cada atividade e de informações que registram o comportamento das mesmas frente à evolução de esquemas.

Buscou-se, com esse tipo de representação, o armazenamento de todo o andamento das instâncias, principalmente para fins históricos, uma vez que todas as atividades executadas são registradas na instância específica. Este aspecto, em conjunto com as informações dos esquemas de base, permite que o período de existência de cada instância seja recuperado.

Além disso, face a possibilidade de ocorrência de evolução dos esquemas, foi também desenvolvido um método para a determinação de compatibilidade entre uma instância e um determinado esquema, denominado de compatibilidade parcial. Dessa forma, se uma instância é parcialmente compatível com um esquema, pode ser migrada para ele e continuar sua execução como se nada houvesse acontecido. No entanto, o método somente verifica a migração de instâncias compatíveis, não permitindo que instâncias incompatíveis sejam adaptadas. Esta tarefa é, na maioria das vezes, manual, necessitando interferência direta do administrador de *workflow* para que as instâncias possam seguir suas execuções.

O método proposto, entretanto, apresenta limitações. De acordo com a seção 4.2.2, uma instância é compatível com um esquema se as suas respectivas árvores possuirem os mesmos elementos do ponto de vista da instância. Porém, é possível que as duas representações sejam compatíveis, mesmo havendo diferença de nodos. Isso ocorre pois muitos dos nodos são roteadores e não interferem no caminho de execução da instância. Entretanto, essa “compatibilidade velada” não é detectada pelo método, reduzindo dessa maneira o número de instâncias que poderiam ser migradas naturalmente para a nova representação.

5 Versionamento de Esquemas de *Workflow*

Um esquema de *workflow*, conforme já destacado, é constantemente modificado, pelas mais diversas razões. Algumas modificações têm como objetivo realizar correções no esquema, fazendo com que um novo esquema substitua o atual. Outras alterações são de caráter alternativo, abrindo a possibilidade de existência de mais de uma forma de se realizar determinado processo. No entanto, tais operações não devem ser consideradas específicas para a geração de substituições ou alternativas. O conjunto de operações de modificação de *workflow* é responsável somente por alterar um esquema, sendo que a escolha pela geração de um esquema de substituição ou alternativo deve ficar a cargo do aplicador das mudanças, ou seja, do administrador de *workflow*.

Além disso, pode-se desejar armazenar todo o conjunto de esquemas que foram gerados (substitutos ou alternativos), mantendo-se assim um histórico de tudo o que foi modificado, desde o esquema originalmente definido. Dessa forma, pode-se ter vários esquemas válidos num determinado instante.

Por outro lado, quando um esquema é modificado, as instâncias devem ser migradas para o novo esquema gerado. Isso é típico em um esquema de substituição, o qual anula o esquema antigo e passa a valer no lugar deste. Porém, quando se tem esquemas alternativos, tal migração não é necessária para todas as instâncias, que podem continuar executando segundo o esquema original, uma vez que todas podem ser válidas naquele instante.

Os argumentos apresentados motivam o uso de versões como suporte para evolução de esquemas de *workflow*, permitindo que todas as modificações efetuadas sobre os mesmos, desde o esquema original, possam ser armazenadas, bem como possibilitando a representação de versões de esquemas alternativas.

A seção 5.1 introduz alguns conceitos sobre versões, importantes para este trabalho. A seção 5.2 apresenta a estrutura definida para o armazenamento de versões de esquema de *workflow*. A seção 5.3 mostra os estados possíveis que uma versão de esquema de *workflow* pode assumir durante seu ciclo de vida. Os aspectos de tempo, aplicados aos estados que uma versão pode assumir, são discutidos na seção 5.4. A seção 5.5 apresenta as principais operações que podem ser aplicadas sobre as versões. Por fim, a seção 5.6 traz algumas conclusões.

5.1 Conceitos

Nesta seção, são apresentados alguns termos sobre versões, de acordo com Galante, Edelweiss e Santos (2002), Golendziner (1995), Kradolfer (2000), Moro (2001):

- **versão** – representa um estado de uma entidade que evolui, sendo que esta entidade pode ser um arquivo num sistema de arquivos, um objeto em um banco de dados orientado a objetos, etc. No caso deste trabalho, a entidade em questão é um esquema de *workflow*, e as versões são as diferentes descrições deste esquema ao longo do tempo;

- **grafo de versões** – responsável por agrupar as versões de uma entidade e representar os relacionamentos existentes entre elas. Esses relacionamentos são do tipo antecessor-sucessor. Normalmente, existe uma versão que inicia o grafo, sendo que

esta não possui antecessor por ser a primeira versão. As formas mais comuns de grafos de versões são seqüência linear, árvore e grafo acíclico dirigido, sendo o diferencial em cada uma das formas o número de antecessores e/ou sucessores permitido. O grafo de versões, no caso de objetos, é denominado de objeto versionado, ou seja, uma coleção de versões de um mesmo objeto. De maneira similar, um **esquema versionado** pode ser considerado como uma coleção de versões de esquema de *workflow*. Esta estrutura pode ser criada de duas maneiras: explícita, quando existe um comando específico de criação de esquema versionado e posteriormente uma versão inicial é anexada a ele, e implícita, quando o esquema versionado é definido automaticamente quando da criação de um novo esquema;

- **estados de versão** – a cada momento, uma versão pode estar em um dos estados definidos para ela. Os estados representam o ciclo de vida de uma versão, indicando quais operações podem ser realizadas sobre a mesma. Exemplos de estados podem ser: em trabalho, estável, liberada, etc.

Além destes conceitos, existem outros, como configuração, referência, etc., porém os apresentados nesta seção serão os necessários para a definição do modelo que baseará a evolução de esquemas proposto neste trabalho.

5.2 Estrutura do Esquema Versionado

Os modelos que se baseiam em características de versionamento tipicamente armazenam as versões das entidades, de maneira que seja possível saber quais versões pertencem a determinada entidade, quais são os relacionamentos entre essas versões, entre outros aspectos. Além disso, quando uma determinada versão não interessa mais ao contexto, ela é descartada, sendo que suas informações não podem mais ser recuperadas.

No caso particular do versionamento de esquemas de *workflow* deste trabalho, além da preocupação com os relacionamentos entre versões, também são enfatizados aspectos históricos. Para tanto, a abordagem de versionamento aqui proposta leva em consideração dois princípios básicos:

- qualquer modificação de uma versão sempre deve gerar uma nova versão;
- não existe remoção física de versões, ou seja, todas são armazenadas para fins históricos e para consultas.

A estrutura mais adequada para representar um esquema versionado de *workflow* é a **árvore** (figura 5.1), por apresentar alguns aspectos desejáveis a este caso:

1. exige que uma versão seja filha de somente uma outra versão, ou seja, que possua somente uma versão antecessora (com exceção da primeira);
2. permite que uma versão possua várias versões filhas, ou seja, que possua várias sucessoras.

É essencial que, no caso de versionamento de esquemas de *workflow*, os dois requisitos sejam respeitados. O requisito 1 é necessário porque tanto as operações de modificação de versão de esquema (capítulo 3) quanto a operação de derivação de versão

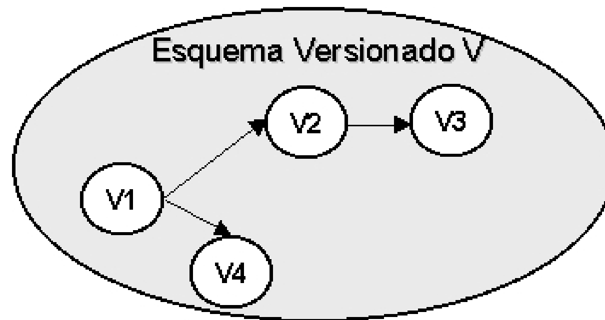


FIGURA 5.1 – Esquema versionado de *workflow* em árvore

(seção 5.5) são sempre aplicadas sobre uma versão em particular. Já o requisito 2 garante que possam existir várias versões alternativas de uma mesma versão antecessora.

A estrutura seqüencial não é a melhor estrutura pois, apesar de uma versão ser filha de somente uma antecessora (requisito 1), o requisito 2 não é satisfeito, uma vez que uma seqüência não permite versões alternativas de uma versão antecessora. Da mesma forma, o grafo acíclico dirigido também não é a estrutura mais adequada, pois viola o requisito 1, apesar de respeitar o requisito 2.

Uma das grandes vantagens de XML é a representação eficiente de estruturas hierárquicas, ou seja, baseadas em árvore. Seus nodos podem ser aninhados pelas *tags* de XML, de forma que é possível estabelecer e identificar as relações de antecessor-sucessor presentes nas árvores.

Portanto, uma vez que o esquema versionado de *workflow* pode ser representado como uma árvore de versões e XML, por sua vez, modela estruturas hierárquicas, adotou-se esta linguagem para a representação da árvore de versões de esquemas de *workflow*, ou seja, para a representação de seu esquema versionado.

A figura 5.2 ilustra um exemplo de representação das versões de um esquema de *workflow* por meio de um documento XML que corresponde ao esquema versionado. A partir dessa representação, pode-se observar a relação pai-filho(s) existente entre os elementos XML e, conseqüentemente, entre os nodos da árvore. Versões alternativas são representadas como elementos filhos de um mesmo elemento pai. Outra observação importante é que cada esquema deve possuir seu próprio esquema versionado, ou seja, o número de documentos XML que representam esquemas versionados deve ser igual ao número de diferentes esquemas.

A estrutura do documento XML que representa o esquema versionado é composta pelos elementos **`versioned-schema`** e **`version`**. O elemento `versioned-schema` é a raiz do documento, responsável por dar o nome do esquema versionado por meio de um atributo, **`id`**, e também por manter as versões em seu interior. Já cada versão é representada por um elemento `version`, contendo uma referência à versão por meio do atributo **`name`**. Como apresentado no capítulo 3, cada versão de esquema corresponde a um documento XML que especifica o fluxo de controle daquela versão.

5.3 Estados das Versões de Esquema de *Workflow*

Durante o ciclo de vida de uma versão de esquema de *workflow*, a mesma pode passar por diversos estados, que representam estágios do seu desenvolvimento. Estes

```
<versioned-schema id="v">
  <version name="1">
    <version name="2">
      <version name="5"/>
      <version name="7"/>
    </version>
    <version name="3"/>
    <version name="4">
      <version name="6"/>
    </version>
  </version>
</versioned-schema>
```

FIGURA 5.2 – Esquema versionado de *workflow* em XML

estados, apresentados na figura 5.3, são os seguintes:

- **working** – neste estado, a versão está sendo criada, podendo sofrer modificações até ser considerada correta. Então, por meio de uma promoção, ela passa ao estado stable. Por outro lado, a mesma pode ser excluída fisicamente, uma vez que está numa fase preliminar do seu ciclo de vida, não possuindo ainda uma “existência real”. A exclusão física se caracteriza pela remoção de todas as informações da versão de esquema. Além disso, uma versão no estado working ainda não pode ser instanciada;

- **stable** – uma versão alcança este estado a partir de uma promoção do estado working, sendo considerada, a partir de então, correta, porém ainda incapaz de gerar instâncias. Neste estado, a versão de esquema pode servir de base para a derivação de novas versões, as quais iniciam no estado working. Neste caso, a versão original permanece no estado stable. Para a criação de instâncias, a versão deve passar para o estado active, a partir de uma operação de ativação. Outra operação permitida é a exclusão lógica da versão de esquema que, ao contrário da exclusão física, não remove suas informações (ver estado logically removed);

- **active** – uma versão alcança este estado por meio de uma operação de ativação a partir do estado stable. Este é o único estado do ciclo de vida que permite a geração de instâncias. A versão pode ser suspensa temporariamente, passando para o estado suspended. Oportunamente, pode-se retornar ao estado active, por uma reativação. Assim como o estado stable, o estado active pode também derivar outras versões de esquemas. Ao gerar uma nova versão, a versão atual continua no estado active. Se a versão não possuir mais utilidade, ela pode ser direcionada para being substituted

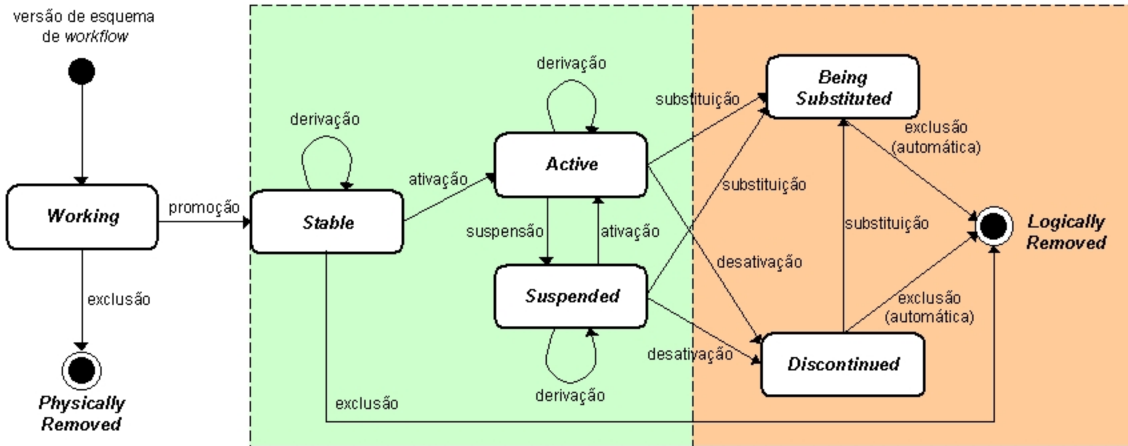


FIGURA 5.3 – Diagrama dos estados possíveis para uma versão de esquema de *workflow*

ou discontinued, responsáveis pela adequada manipulação das instâncias ativas e pela exclusão lógica da versão;

- **suspended** – este é um estado temporário, no qual uma versão de esquema pode permanecer por tempo indeterminado. Não se permite a geração de novas instâncias. Se isso for necessário, deve-se retornar ao estado active, por meio de uma operação de ativação. Uma versão no estado suspended não invalida as instâncias em execução, que podem ser finalizadas segundo o esquema. Da mesma forma que no estado active, a versão pode ser desativada, passando para um dos estados específicos para este fim;

- **discontinued** – uma versão de esquema alcança este estado a partir dos estados active ou suspended. Este estado, bem como o estado being substituted, é um estado terminal, ou seja, não permite o retorno a algum estado anterior. Nesses dois estados, novas instâncias não podem ser geradas, porém, no estado discontinued, as antigas podem terminar segundo a versão atual de esquema. Após o término da execução das instâncias, a versão passa automaticamente para o estado logically removed. No entanto, se ao final de um certo período for necessária sua exclusão, mesmo ainda possuindo instâncias, a versão pode passar para o estado being substituted;

- **being substituted** – uma versão de esquema alcança este estado a partir dos estados active ou suspended. Neste caso, diferentemente do estado discontinued, a versão será excluída, mesmo que possua instâncias em execução. Essas instâncias devem ser migradas para outra versão antes da exclusão lógica, que é processada automaticamente;

- **logically removed** – uma versão de esquema chega até este estado a partir dos estados stable, discontinued ou being substituted, seja por uma operação de exclusão explícita ou automática. Neste estado, a versão é mantida para fins históricos, diferenciando-se, dessa forma, da exclusão física.

A tabela 5.1 apresenta as ações que são permitidas a uma versão de esquema de acordo com o estado no qual a mesma se encontra.

TABELA 5.1 – Ações possíveis sobre os estados de uma versão de esquema de *workflow*

Estados	Derivar versão	Gerar instâncias	Possuir instâncias	Ser excluída
working	não	não	não	sim (física)
stable	sim	não	não	sim (lógica)
active	sim	sim	sim	não
suspended	sim	não	sim	não
discontinued	não	não	sim	sim (lógica)
being substituted	não	não	sim (*)	sim (lógica)

(*) - a versão pode possuir instâncias na entrada do estado, mas ao final não deverá ter nenhuma

5.4 Análise Temporal das Versões de Esquema

O tempo de duração de cada estado é uma característica importante que deve ser armazenada. Com essa informação, consultas sobre o passado, presente e o (provável) futuro de cada versão podem ser realizadas. No modelo de evolução de esquemas de *workflow* aqui proposto, toda mudança de estado é registrada para cada versão, com a informação temporal de início e fim da validade daquele estado.

De acordo com Edelweiss (1998), existem dois tipos principais de tempo: o **tempo de transação**, que define quando um fato foi inserido num banco de dados, e o **tempo de validade**, que indica quando o fato é considerado válido. O tempo de transação é atribuído pelo sistema, enquanto que o tempo de validade é fornecido pelo usuário. Bancos de dados que armazenam tanto o tempo de transação quanto o tempo de validade são chamados de **bitemporais**.

No modelo de versões que está sendo apresentado neste capítulo, será utilizado somente o tempo de validade para a anotação dos períodos de cada estado, uma vez que este tipo de tempo é o que realmente informa quando uma determinada versão de esquema esteve em um determinado estado.

Uma mudança de estado pode ser feita instantaneamente ou armazenada para ser efetivamente realizada num ponto futuro. No entanto, o instante em que essa mudança é armazenada (tempo de transação) não é levado em consideração. Dessa maneira, tem-se que o início do tempo de validade pode ser o momento presente (instantâneo) ou um momento futuro, mas nunca passado, pois isso influiria em ações que já foram realizadas.

A figura 5.4 mostra a evolução de três versões de esquemas de *workflow*. A versão de esquema A foi declarada stable no instante 1. No instante 3, a mesma versão foi declarada active. Após isso, no instante 5, A derivou a versão B, que foi declarada stable em 6 e active no instante 8. No instante 9, A derivou outra versão, C, declarada stable em 10 e active em 12. Após isso, A foi desativada, passando ao estado discontinued em 13. No instante 14, foi armazenada uma solicitação de suspensão de C para o instante 15. Em 17, a versão A é passada para o estado being substituted, sendo que suas instâncias ainda ativas são migradas para B. A migração finaliza no instante 19, quando A é excluída automaticamente.

Pode-se observar que, numa mudança de estado, inicia-se a validade do novo estado, devendo a validade do estado anterior ser terminada. Dessa forma, para a versão A do exemplo anterior, os intervalos de validade dos seus estados ficaram, após o instante 19, os seguintes: estado stable, $[1, 3)$; estado active, $[3, 13)$; estado discontinued,

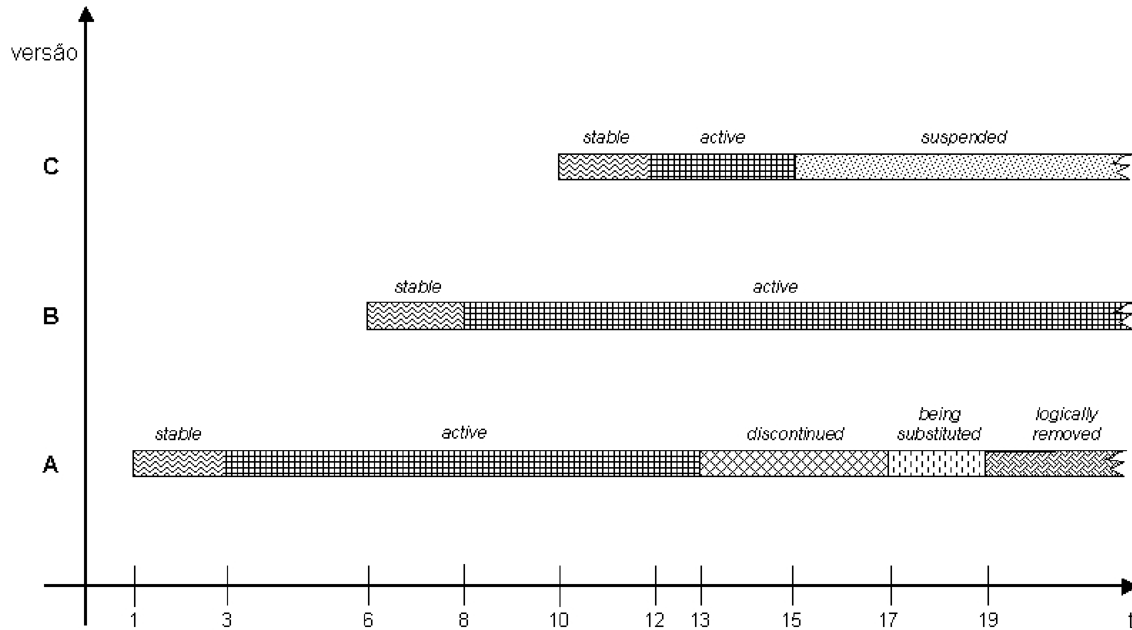


FIGURA 5.4 – Gráfico do ciclo de vida de versões em função do tempo

$[13, 17)$; estado being substituted, $[17, 19)$; e estado logically removed, $[19, now)$. Deve ser ressaltada a obrigatoriedade do intervalo aberto à direita em cada estado, para que não seja permitida a uma versão que esteja em dois estados ao mesmo tempo. O estado logically removed sempre terá *now* como limite superior, ou seja, este estado é infinito para o futuro.

Como os estados (e seus respectivos tempos de validade) são relacionados às versões, decidiu-se anexar a estas os períodos de tempo em que seus estados valeram. Sendo assim, as informações temporais são colocadas em cada arquivo XML que representa uma versão particular. A figura 5.5 ilustra a inclusão dos valores temporais dos estados na versão A do exemplo desta seção.

```
<?xml version="1.0"?>
<schema-version name="A">
  <life-cycle>
    <status name="stable" itime="1" ftime="3"/>
    <status name="active" itime="3" ftime="13"/>
    <status name="discontinued" itime="13" ftime="17"/>
    <status name="being substituted" itime="17" ftime="19"/>
    <status name="logically removed" itime="19"/>
  </life-cycle>
  <process>
    ...
  </process>
</schema-version>
```

FIGURA 5.5 – Representação dos estados na versão de esquema

Cada estado é armazenado em um elemento `status`, com as informações de nome, início e final da validade. Todos os elementos `status` de uma versão são filhos de um elemento `life-cycle`. Esse trecho de documento XML é anexado ao documento que corresponde a uma versão de esquema.

5.5 Operações sobre Esquemas e Versões

Como enfatizado anteriormente, o versionamento permite a criação de alternativas para um determinado esquema de *workflow*, de forma que toda sua evolução possa ser registrada. Tanto a criação de um esquema versionado quanto o processo de mudança de uma determinada versão de esquema são realizados por alguém responsável pela manutenção de toda a estrutura. Por isso, é importante que algumas operações possam ser disponibilizadas para auxílio nessas tarefas. Além disso, uma grande parte das mudanças de estados deve ser feita pelo administrador, o que também exige a existência de mecanismos adequados para sua realização.

Esta seção apresentará algumas operações aplicáveis sobre esquemas de *workflow* e versões. Essas operações realizam, indiretamente, alterações nos esquemas versionados correspondentes.

5.5.1 Operação de Criação de Esquema

A operação *createSchema* tem por finalidade criar um esquema de *workflow*, refletindo um novo processo de negócio. Para tanto, é necessário que a operação receba como parâmetro a especificação do esquema, de acordo com a linguagem apresentada no capítulo 3. Além disso, convém destacar que, quando da criação de um novo esquema de *workflow*, ainda não existe um esquema versionado correspondente. Dessa maneira, ocorre a criação implícita de esquema versionado.

A seguir, é feita uma descrição dos passos desta operação:

1. o arquivo XML que contém a especificação do esquema deve ser verificado quanto à correção, utilizando-se um *parser* apropriado;

2. como se trata de um esquema novo, o mesmo também representa a primeira versão do novo esquema versionado. Dessa forma, cria-se uma versão de esquema, que é posteriormente armazenada no repositório apropriado;

3. a versão inicia no estado working e é automaticamente promovida a stable, por não necessitar de correções;

4. adicionalmente, deve-se também criar um esquema versionado para este esquema, cuja primeira versão será a própria especificação passada como parâmetro e já armazenada no repositório de versões de esquemas. O novo esquema versionado é armazenado em um repositório apropriado para tal categoria.

5.5.2 Operação de Derivação de Versões

A operação *deriveVersion* é responsável pela aplicação de mudanças a uma versão de esquema de *workflow*. As mudanças são sempre efetuadas a partir de um documento XUpdate que contém uma ou várias operações descritas na seção 3.2.2. Como conseqüência, pode-se dizer que *deriveVersion* cria uma nova versão de esquema por meio de uma derivação, uma vez que qualquer modificação aplicada sobre uma versão resulta numa nova versão. Os parâmetros necessários são a versão de esquema a ser modificada e o arquivo XML contendo as instruções XUpdate.

Pelo fato de ser uma implementação de derivação, esta operação somente pode ser aplicada a versões cujo estado atual permite que uma derivação possa ser realizada. Sendo

assim, pela figura 5.3 e pela tabela 5.1, tem-se que os estados que admitem derivação são stable, active e suspended, não sendo permitida em outros estados.

Como no caso da operação *createSchema*, segue uma descrição da operação *deriveVersion*:

1. é feita uma verificação se a derivação é aplicável ao estado atual da versão de esquema. Um erro deve ser reportado caso a versão esteja em um estado diferente de stable, active ou suspended;

2. da mesma forma que na criação de esquema, o arquivo XML recebido como parâmetro deve ser verificado quanto à correção;

3. é criada uma cópia da versão a ser modificada. A versão original permanece no mesmo estado e a cópia passa para o estado working. Essa nova versão não é colocada ainda no esquema versionado, pois ainda não foi modificada com as operações e, além disso, pode ser excluída fisicamente;

4. as operações de modificação, descritas no arquivo XML, são aplicadas sobre a versão de esquema que está em trabalho. Considera-se que as operações XUpdate não causam falsas atualizações, como discutido na seção 3.2.3;

5. em caso de sucesso nas alterações, a nova versão é promovida para o estado stable, constituindo-se em uma versão correta;

5.5.3 Operações de Mudança de Estados

As operações apresentadas nesta seção são utilizadas para mudança de estados das versões, ou seja, sempre envolvem dois estados interligados conforme o diagrama apresentado na figura 5.3. Desta forma, essas operações são responsáveis pelo andamento da versão pelo seu ciclo de vida.

A operação *activateVersion* tem por finalidade habilitar uma versão a gerar instâncias, ou seja, passar a mesma para o estado active. A operação de ativação obriga que a versão de esquema esteja ou no estado stable ou no estado suspended, permitindo que a mesma comece a possuir instâncias ou volte a produzi-las, respectivamente. Se a versão estiver em algum estado diferente de stable ou suspended, um erro deve ser reportado.

A operação *suspendVersion* desabilita a geração de instâncias por uma versão, levando a mesma do estado active ao estado suspended. Se a versão não estiver no estado active, a operação deve ser desconsiderada.

A operação *deactivateVersion* é responsável por levar uma versão do estado active ou suspended para o estado discontinued, o qual permite que as instâncias antigas continuem em execução. Após o término da última instância, automaticamente a versão é excluída logicamente.

A operação *replaceVersion* tem como objetivo migrar todas as instâncias de uma versão para outra versão descendente. A versão original, então, não pode mais ser utilizada. Esta operação passa uma versão do estado active, suspended ou discontinued para o estado being substituted. Neste estado, as instâncias são migradas e, após isso, a versão é excluída logicamente.

A operação *deleteVersion*, por sua vez, caracteriza-se por levar uma versão diretamente para o estado logically removed ou physically removed. Esta operação pode ser aplicada em versões que estejam no estado working ou stable. A operação *deleteVersion*

não pode ser aplicada sobre versões no estado discontinued e being substituted, pois nestes casos a passagem para o estado logically removed se processa de maneira automática.

Além das operações descritas anteriormente, outras mais podem ser definidas. Entre elas, podem ser destacadas operações de consulta a versões (por exemplo, *getState*, que devolve o estado em que uma versão se encontra) e de migração individual de instâncias para alguma versão (*migrateInstance*), sem a necessidade de exclusão da versão. No entanto, não é objetivo deste trabalho explanar todas as possíveis operações, porém somente aquelas que tratam diretamente com a mudança dos estados das versões.

A tabela 5.2 apresenta as operações sobre versões, discutidas nas seções 5.5.2 e 5.5.3, relacionando-as com os estados que permitem a realização das mesmas. Por sua vez, a tabela 5.3 apresenta a utilização das operações sobre esquemas e versões, aplicadas ao exemplo da seção 5.4. Por esta tabela, é possível verificar que algumas operações são feitas automaticamente pelo WFMS, como a promoção (passagem ao estado stable) e a exclusão lógica (quando a versão estiver no estado discontinued ou being substituted). As outras operações, entretanto, são realizadas explicitamente pelo responsável pelo esquema de *workflow*, podendo ser aplicadas instantaneamente ou armazenadas para execução futura.

TABELA 5.2 – Estados que aceitam operações sobre versões

Operações	Estados Permitidos
deriveVersion	stable, active, suspended
activateVersion	stable, suspended
suspendVersion	active
deactivateVersion	active, suspended
replaceVersion	active, suspended, discontinued
deleteVersion	working, stable

TABELA 5.3 – Aplicação das operações sobre esquemas e versões de esquema

Tempo	Operação Realizada
0	EV = *createSchema*(A.xml)
1	A é declarada stable (operação interna)
3	*activateVersion*(A)
5	B = *deriveVersion*(A, op. XUpdate)
6	B é declarada stable (operação interna)
8	*activateVersion*(B)
9	C = *deriveVersion*(A, op. XUpdate)
10	C é declarada stable (operação interna)
12	*activateVersion*(C)
13	*deactivateVersion*(A)
14	pedido para suspensão de C em 15
15	aplicação de *suspendVersion*(C) armazenada em 14
17	*replaceVersion*(A,B)
19	A é excluída logicamente (operação interna)

5.6 Considerações Finais

Este capítulo apresentou um modelo para evolução de esquemas de *workflow*, enfatizando o versionamento dos mesmos, permitindo o surgimento de versões alternativas válidas para um mesmo esquema e também o armazenamento de todas as versões para fins históricos. Em contrapartida, essa abordagem demanda muito espaço de armazenamento, uma vez que nenhuma versão que no mínimo foi promovida é desperdiçada.

O esquema versionado, estrutura que agrupa todas as versões de um mesmo esquema, foi definido em XML, uma vez que esta linguagem é eficiente para a representação de hierarquias, encaixando-se perfeitamente na proposta de esquema baseado em árvore. As relações antecessor-sucessor(es), bem como versões alternativas, são corretamente representadas, a partir de elementos `version`.

Além disso, foi definido um conjunto de estados que podem ser assumidos pelas versões de esquemas, constituindo-se no ciclo de vida das mesmas. Esses estados indicam quais operações podem ser realizadas sobre uma determinada versão, como por exemplo, derivar novas versões, permitir a geração de instâncias, determinar a suspensão temporária de uma versão, permitem que uma versão possa ser armazenada quando a mesma estiver obsoleta ou desatualizada, entre outras possibilidades. A evolução dos estados de uma versão é armazenada na própria versão, sendo que todos os períodos de validade podem ser consultados.

Por último, foram criadas algumas operações para a manipulação de esquemas e versões, visando facilitar as tarefas do responsável pelos mesmas. Com essas operações, pode-se criar um novo esquema (e conseqüentemente um esquema versionado), derivar nova versão, bem como mudar a versão de estado. É importante ressaltar, no entanto, que as operações de criação de esquema e de derivação de versões recebem como parâmetro, entre outras informações, documentos XML responsáveis pela definição de esquema e pela modificação do mesmo, cujas estruturas foram apresentadas no capítulo 3. Além desse conjunto de operações, outras podem ser definidas, de acordo com as finalidades específicas.

Convém destacar também que as estruturas de esquema versionado e de estados apresentadas neste capítulo são genéricas e independentes do formato de especificação das versões dos esquemas. A figura 5.2 apresentou documentos XML como sendo o formato dos nodos do esquema versionado. Entretanto, tais versões poderiam ser representadas, por exemplo, em Redes de Petri ou outro formalismo que modele esquemas de *workflow*.

6 Estudo de Caso

Neste capítulo, as estratégias e conceitos propostos nos capítulos 3, 4 e 5 são abordados em conjunto, com a utilização de um estudo de caso que permite o versionamento de esquemas de *workflow* e a manipulação de instâncias. Na seção 6.1, o contexto do estudo de caso é descrito. Na seção 6.2, o mesmo é modelado e modificado de acordo com os capítulos anteriores. Por fim, na seção 6.3 são apresentadas algumas considerações finais sobre o estudo de caso.

6.1 Descrição do Contexto

O estudo de caso que será descrito neste capítulo refere-se a uma variação do exemplo apresentado por Christophides, Hull e Kumar (2001), referente ao processo de montagem de computadores. No caso deste estudo em particular, se trata de uma empresa montadora de computadores *desktop* por demanda, que recebe pedidos de compra dos clientes e realiza a montagem dos componentes fornecidos por outras empresas.

O processo é composto por duas grandes partes: a parte da **análise do pedido** e a parte da **montagem** propriamente dita. A parte da análise do pedido inicia a partir da recepção de uma solicitação de compra. A seguir, algumas atividades são realizadas sobre o pedido. De acordo com o valor total, o mesmo é direcionado para aprovação do gerente (valor $\leq$ R$ 5.000,00) ou do vice-presidente da empresa (valor $>$ R$ 5.000,00). Enquanto isso, são realizadas verificações técnica e orçamentária sobre o pedido. Após esses passos, se o pedido for aprovado, então passa para a montagem, senão, deve ser refeito. Como a empresa não possui as peças para a montagem dos computadores, a mesma deve solicitá-las a terceiros sob demanda, através de pedidos específicos. Quando da chegada desses componentes, acontece a montagem dos computadores e o posterior envio dos mesmos ao solicitante. Ao final, notas de cobrança são emitidas para que o cliente pague pelo pedido realizado. O fluxo de controle deste processo é apresentado na figura 6.1.

Dessa maneira, a diretoria solicitou ao administrador de *workflow* que construísse uma representação computacional do processo de montagem de computadores. No entanto, o mesmo foi alertado que o processo poderia ser modificado no futuro, em virtude de inovações que a empresa desejasse implantar em sua linha de produtos. Então, o administrador decidiu por utilizar uma estratégia de versionamento, permitindo que variações do mesmo processo estivessem ativas ao mesmo tempo.

A seguir, é apresentada a modelagem do estudo de caso e as evoluções ocorridas no mesmo em um certo período de tempo.

6.2 Modelagem do Estudo de Caso

Após conversar com os responsáveis pelo processo de montagem de computadores, o administrador de *workflow* construiu um esquema para representar tal processo. Ao final dessa etapa, ele apresentou o esquema gerado, denominado PC1, ilustrado na figura 6.2.

Após a verificação e a concordância da diretoria, o administrador implantou esse esquema no WFMS, no dia 01 de janeiro de 2001, às 7h, a partir da operação de criação de esquemas:

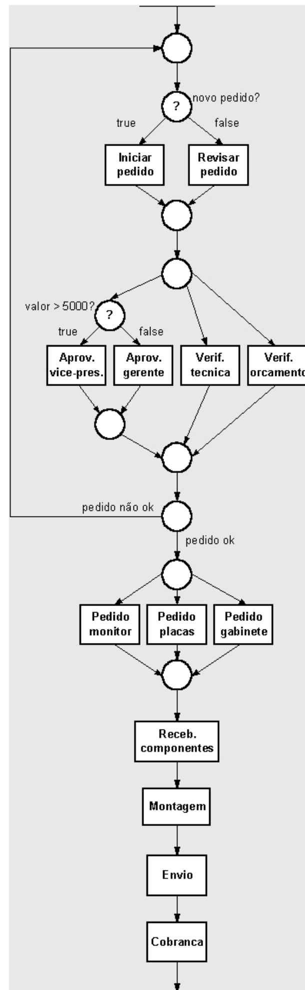


FIGURA 6.1 – Fluxo de controle do processo analisado

PC = createSchema(PC1.xml)

Internamente, essa operação foi responsável por:

1. verificar PC1 quanto aos aspectos de correção;

2. criar um esquema versionado (`PC.xml`) com a referência à PC1, que se tornou a primeira versão:

```
<?xml version="1.0" encoding="UTF-8"?>
<versioned-schema id="PC">
  <version name="PC1.xml"/>
</versioned-schema>
```

3. passar a versão PC1 para o estado stable, uma vez correta. Além disso, atualizou-se PC1 com informações do estado:

```
...
<life-cycle> <!-- versão PC1 -->
  <status name="stable" itime="2001-01-01T07:00:00"/>
</life-cycle>
...
```

```
<?xml version="1.0" encoding="UTF-8"?>
<schema-version xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.inf.ufrgs.br/~fabiozs/workflow.xsd">
  <process>
    <sequence>
      <loop until="pedido = ok">
        <sequence>
          <conditional condition="novopedido">
            <true>
              <activity name="Iniciar pedido"/>
            </true>
            <false>
              <activity name="Revisar pedido"/>
            </false>
          </conditional>
          <parallel>
            <conditional condition="valor &gt; 5000">
              <true>
                <activity name="Aprov. vice-pres."/>
              </true>
              <false>
                <activity name="Aprov. gerente"/>
              </false>
            </conditional>
            <activity name="Verif. tecnica"/>
            <activity name="Verif. orcamento"/>
          </parallel>
        </sequence>
      </loop>
      <parallel>
        <activity name="Pedido monitor"/>
        <activity name="Pedido placas"/>
        <activity name="Pedido gabinete"/>
      </parallel>
      <activity name="Receb. componentes"/>
      <activity name="Montagem"/>
      <activity name="Envio"/>
      <activity name="Cobranca"/>
    </sequence>
  </process>
</schema-version>
```

FIGURA 6.2 – Esquema de *workflow* de montagem de computadores (`PC1.xml`)

Após isso, decidiu-se que este processo deveria começar a funcionar, para que pudesse receber os primeiros pedidos. O administrador então modificou o estado de PC1 às 8h do dia 02 de janeiro, permitindo que esta versão possuísse instâncias em execução:

activateVersion(PC1)

Com essa operação, a versão PC1 passou para o estado active, concluindo o tempo de permanência em stable:

```
...
<life-cycle> <!-- versão PC1 -->
  <status name="stable" itime="2001-01-01T07:00:00" ftime="2001-01-02T08:00:00"/>
  <status name="active" itime="2001-01-02T08:00:00"/>
</life-cycle>
...
```

Ainda no mesmo dia, começaram a surgir os pedidos. Para cada um deles, uma operação *createInstance(PC1)* foi necessária. A partir de uma operação desse tipo, um documento de instância foi criado, com uma estrutura básica e referência a PC1, que era a versão de base. A seguir, um exemplo de uma instância criada:

```
<instance id="1">
  <schema-ref>
    <version name="PC1.xml" itime="2001-01-02T10:30:00"/>
  </schema-ref>
  <process>
    ...
  </process>
</instance>
```

Após algum tempo, a diretoria resolveu modificar o processo, retirando o vice-presidente da tarefa de aprovação de pedidos, por não dispôr de tempo suficiente para todas as obrigações. Assim, decidiu-se que toda e qualquer aprovação deveria ser efetuada somente pelo gerente responsável. Além disso, constatou-se que as atividades de envio e cobrança poderiam ser feitas simultaneamente, o que reduziria o tempo médio de execução das instâncias. Dessa forma, essas modificações foram encaminhadas ao administrador de *workflow*, que decidiu aplicar o documento XUpdate `mod1.xml` (figura 6.3) a PC1 em 25 de abril de 2001, às 14h, derivando dessa forma uma nova versão, armazenada em `PC2.xml` (figura 6.4):

PC2 = deriveVersion(PC1, mod1.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<xupdate:modifications version="1.0" xmlns:xupdate="http://www.xmldb.org/xupdate">

  <!-- Mudanca 1: retirar a aprovacao pelo vice-presidente -->
  <!-- selecao da atividade Aprov. gerente -->
  <xupdate:variable name="gerente" select="/schema-version/process/sequence/loop/sequence/
    parallel/conditional/false/activity"/>
  <!-- insercao da atividade Aprov. gerente antes do condicional -->
  <xupdate:insert-before select="/schema-version/process/sequence/loop/sequence/parallel/
    conditional">
    <xupdate:value-of select="$gerente"/>
  </xupdate:insert-before>
  <!-- remocao do condicional -->
  <xupdate:remove select="/schema-version/process/sequence/loop/sequence/parallel/
    conditional"/>

  <!-- Mudanca 2: colocar cobranca em paralelo a envio -->
  <!-- selecao das atividades Envio e Cobranca -->
  <xupdate:variable name="envio" select="/schema-version/process/sequence/
    activity[@name='Envio']"/>
  <xupdate:variable name="cobranca" select="/schema-version/process/sequence/
    activity[@name='Cobranca']"/>
  <!-- insercao da estrutura paralela apos a atividade Montagem -->
  <xupdate:insert-after select="/schema-version/process/sequence/
    activity[@name='Montagem']">
    <xupdate:element name="parallel"/>
  </xupdate:insert-after>
  <!-- adicao das atividades Envio e Cobranca a estrutura paralela -->
  <xupdate:append select="/schema-version/process/sequence/parallel[2]">
    <xupdate:value-of select="$envio"/>
  </xupdate:append>
  <xupdate:append select="/schema-version/process/sequence/parallel[2]">
    <xupdate:value-of select="$cobranca"/>
  </xupdate:append>
  <!-- remocao das atividades Envio e Cobranca antigas -->
  <xupdate:remove select="/schema-version/process/sequence/activity[@name='Envio']"/>
  <xupdate:remove select="/schema-version/process/sequence/activity[@name='Cobranca']"/>

</xupdate:modifications>
```

FIGURA 6.3 – Modificações aplicadas sobre a versão PC1 (`mod1.xml`)

```
<?xml version="1.0" encoding="UTF-8"?>
<schema-version xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.inf.ufrgs.br/~fabiozs/workflow.xsd">
  <process>
    <sequence>
      <loop until="pedido = ok">
        <sequence>
          <conditional condition="novopedido">
            <true>
              <activity name="Iniciar pedido"/>
            </true>
            <false>
              <activity name="Revisar pedido"/>
            </false>
          </conditional>
          <parallel>
            <activity name="Aprov. gerente"/>
            <activity name="Verif. tecnica"/>
            <activity name="Verif. orcamento"/>
          </parallel>
        </sequence>
      </loop>
      <parallel>
        <activity name="Pedido monitor"/>
        <activity name="Pedido placas"/>
        <activity name="Pedido gabinete"/>
      </parallel>
      <activity name="Receb. componentes"/>
      <activity name="Montagem"/>
      <parallel>
        <activity name="Envio"/>
        <activity name="Cobranca"/>
      </parallel>
    </sequence>
  </process>
</schema-version>
```

FIGURA 6.4 – Versão derivada de PC1 (`PC2.xml`)

Essa operação produziu as seguintes modificações nos documentos:

1. foi criada uma cópia de `PC1.xml`, denominada `PC2.xml`, que iniciou no estado working;

2. aplicou-se as modificações presentes em `mod1.xml` e a versão PC2 foi verificada quanto à correção;

3. a versão PC2 foi adicionada ao esquema versionado PC, como elemento filho de PC1:

```
<?xml version="1.0" encoding="UTF-8"?>
<versioned-schema id="PC">
  <version name="PC1.xml">
    <version name="PC2.xml"/>
  </version>
</versioned-schema>
```

4. passou-se a versão PC2 para o estado stable:

```
...
<life-cycle> <!-- versão PC2 -->
  <status name="stable" itime="2001-04-25T14:00:00"/>
</life-cycle>
...
```

Em seguida, a versão PC1 foi desativada. Com isso, o WFMS não permitiu que novos pedidos pudessem ser feitos segundo a mesma, mas os pedidos em andamento puderam ser concluídos. Isso foi realizado pelo administrador no mesmo dia da derivação, às 14h30min, a partir da operação:

deactivateVersion(PC1)

Esta operação foi responsável por passar a versão PC1 para o estado discontinued. Dessa forma, atualizou-se o estado no documento:

```
...
<life-cycle> <!-- versão PC1 -->
  <status name="stable" itime="2001-01-01T07:00:00" ftime="2001-01-02T08:00:00"/>
  <status name="active" itime="2001-01-02T08:00:00" ftime="2001-04-25T14:30:00"/>
  <status name="discontinued" itime="2001-04-25T14:30:00"/>
</life-cycle>
...
```

Pelo fato de PC1 não poder mais gerar instâncias, o administrador ativou a versão PC2, passando-a para o estado active às 14h45min:

activateVersion(PC2)

Atualizou-se, dessa forma, o documento `PC2.xml` com o novo estado:

```
...
<life-cycle> <!-- versão PC2 -->
  <status name="stable" itime="2001-04-25T14:00:00" ftime="2001-04-25T14:45:00"/>
  <status name="active" itime="2001-04-25T14:45:00"/>
</life-cycle>
...
```

À medida que novos pedidos de computadores chegaram, estes foram tratados como instâncias de PC2. Dessa forma, durante os dias seguintes, várias instâncias foram geradas, uma para cada pedido.

O último pedido que estava seguindo a versão PC1 terminou sua execução no dia 30 de abril de 2001, às 11h. Esta versão teve seu estado atualizado, passando diretamente para logically removed sem nenhuma interferência do administrador. Apesar disso, PC1 permaneceu no esquema versionado PC, para fins históricos. O ciclo de vida de PC1 então foi concluído:

```
...
<life-cycle> <!-- versão PC1 -->
  <status name="stable" itime="2001-01-01T07:00:00" ftime="2001-01-02T08:00:00"/>
  <status name="active" itime="2001-01-02T08:00:00" ftime="2001-04-25T14:30:00"/>
  <status name="discontinued" itime="2001-04-25T14:30:00"
    ftime="2001-04-30T11:00:00"/>
  <status name="logically removed" itime="2001-04-30T11:00:00"/>
</life-cycle>
...
```

A empresa cresceu e, após algum tempo, a diretoria detectou que necessitava de maior agilidade na entrega das encomendas, tendo em vista a grande concorrência estabelecida naquele nicho de mercado. Dessa forma, ao invés de solicitar os componentes sob demanda, a mesma resolveu criar e manter um estoque, com nível médio baseado na quantidade de pedidos já realizados. Ao mesmo tempo, decidiu investir também na montagem de *laptops*, a partir das sugestões dos clientes. No entanto, os diretores ainda não tinham uma idéia da procura por tal serviço, razão pela qual os componentes, neste caso específico, continuariam sendo solicitados por demanda.

Diante desses fatos, o processo de montagem de computadores teve de ser modificado, para melhor representar a nova realidade da empresa. O administrador de *workflow* propôs a criação de duas versões separadas, uma para tratar da montagem de *desktops* a partir de componentes em estoque e outra para a montagem de *laptops*, o que foi aceito pela diretoria.

Assim, no dia 10 de setembro de 2001, o administrador decidiu derivar duas versões, a partir de PC2. Primeiramente, às 9h, ele derivou a versão PC3, responsável pelo novo processo de montagem de *desktops*, sendo atualizada pelo documento `mod2.xml` (figura 6.5):

PC3 = deriveVersion(PC2, mod2.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<xupdate:modifications version="1.0" xmlns:xupdate="http://www.xmldb.org/xupdate">

  <!-- Mudanca 1: retirar os pedidos a terceiros e o recebimento -->
  <!-- retirada do elemento paralelo, contendo os pedidos a terceiros -->
  <xupdate:remove select="/schema-version/process/sequence/parallel[1]"/>
  <!-- remocao da atividade Receb. componentes -->
  <xupdate:remove select="/schema-version/process/sequence/
    activity[@name='Receb. componentes']"/>

  <!-- Mudanca 2: inserir a atividade Solic. componentes antes da atividade Montagem -->
  <xupdate:insert-before select="/schema-version/process/sequence/
    activity[@name='Montagem']">
    <xupdate:element name="activity">
      <xupdate:attribute name="name">Solic. componentes</xupdate:attribute>
    </xupdate:element>
  </xupdate:insert-before>

</xupdate:modifications>
```

FIGURA 6.5 – Modificações aplicadas sobre a versão PC2 (`mod2.xml`)

A derivação causou as seguintes modificações:

1. a versão PC3 foi criada como uma cópia de `PC2.xml`, iniciando no estado working;
2. as modificações constantes em `mod2.xml` foram aplicadas sobre PC3;
3. sendo considerada correta, a versão PC3 (figura 6.6) foi adicionada ao esquema versionado PC, como elemento filho de PC2:

```
<?xml version="1.0" encoding="UTF-8"?>
<versioned-schema id="PC">
  <version name="PC1.xml">
    <version name="PC2.xml">
      <version name="PC3.xml"/>
    </version>
  </version>
</versioned-schema>
```

4. passou-se a versão PC3 para o estado stable:

```
...
<life-cycle> <!-- versão PC3 -->
  <status name="stable" itime="2001-09-10T09:00:00"/>
</life-cycle>
...
```

```
<?xml version="1.0" encoding="UTF-8"?>
<schema-version xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.inf.ufrgs.br/~fabiozs/workflow.xsd">
  <process>
    <sequence>
      <loop until="pedido = ok">
        <sequence>
          <conditional condition="novopedido">
            <true>
              <activity name="Iniciar pedido"/>
            </true>
            <false>
              <activity name="Revisar pedido"/>
            </false>
          </conditional>
          <parallel>
            <activity name="Aprov. gerente"/>
            <activity name="Verif. tecnica"/>
            <activity name="Verif. orcamento"/>
          </parallel>
        </sequence>
      </loop>
      <activity name="Solic. componentes"/>
      <activity name="Montagem"/>
      <parallel>
        <activity name="Envio"/>
        <activity name="Cobranca"/>
      </parallel>
    </sequence>
  </process>
</schema-version>
```

FIGURA 6.6 – Versão derivada de PC2 (`PC3.xml`)

Mais tarde, às 9h30min, o administrador derivou, também a partir da versão PC2, a versão PC4, responsável pela montagem de *laptops*. Essa cópia foi atualizada pelo documento XUpdate `mod3.xml` (figura 6.7):

PC4 = deriveVersion(PC2, mod3.xml)

A operação de derivação foi responsável por:

1. criar uma cópia de PC2, `PC4.xml`, que iniciou no estado working;
2. aplicar as operações do documento `mod3.xml` sobre `PC4.xml`, gerando a versão modificada (figura 6.8);
3. adicionar PC4 como elemento filho de PC2 no esquema versionado:

```
<?xml version="1.0" encoding="UTF-8"?>
<versioned-schema id="PC">
  <version name="PC1.xml">
    <version name="PC2.xml">
      <version name="PC3.xml"/>
      <version name="PC4.xml"/>
    </version>
  </version>
</versioned-schema>
```

4. passar a versão PC4 para o estado stable:

```
...
<life-cycle> <!-- versão PC4 -->
  <status name="stable" itime="2001-09-10T09:30:00"/>
</life-cycle>
...
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xupdate:modifications version="1.0" xmlns:xupdate="http://www.xmldb.org/xupdate">
  <!-- Mudanca 1: adicionar pedidos dos componentes para laptop e retirar pedidos dos
    componentes para desktop -->
  <!-- inclusao pedidos de componentes de laptop -->
  <xupdate:append select="/schema-version/process/sequence/parallel[1]">
    <xupdate:element name="activity">
      <xupdate:attribute name="name">Pedido tela laptop</xupdate:attribute>
    </xupdate:element>
  </xupdate:append>
  <xupdate:append select="/schema-version/process/sequence/parallel[1]">
    <xupdate:element name="activity">
      <xupdate:attribute name="name">Pedido placas laptop</xupdate:attribute>
    </xupdate:element>
  </xupdate:append>
  <xupdate:append select="/schema-version/process/sequence/parallel[1]">
    <xupdate:element name="activity">
      <xupdate:attribute name="name">Pedido gabinete laptop</xupdate:attribute>
    </xupdate:element>
  </xupdate:append>
  <!-- remocao dos pedidos de componentes de desktop -->
  <xupdate:remove select="/schema-version/process/sequence/parallel/
    activity[@name='Pedido monitor']"/>
  <xupdate:remove select="/schema-version/process/sequence/parallel/
    activity[@name='Pedido placas']"/>
  <xupdate:remove select="/schema-version/process/sequence/parallel/
    activity[@name='Pedido gabinete']"/>
  <!-- Mudanca 2: trocar atividade Montagem por atividade Montagem laptop -->
  <!-- remocao montagem de desktop -->
  <xupdate:remove select="/schema-version/process/sequence/activity[@name='Montagem']"/>
  <!-- insercao ativ. Montagem laptop -->
  <xupdate:insert-after select="/schema-version/process/sequence/
    activity[@name='Receb. componentes']">
    <xupdate:element name="activity">
      <xupdate:attribute name="name">Montagem laptop</xupdate:attribute>
    </xupdate:element>
  </xupdate:insert-after>
</xupdate:modifications>
```

FIGURA 6.7 – Modificações aplicadas sobre a versão PC2 (`mod3.xml`)

A empresa decidiu também que a versão PC2 seria substituída no dia 15 de setembro, às 7h. No entanto, poderiam haver instâncias em execução no momento do término do funcionamento de PC2. Como as instâncias estavam relacionadas à montagem de *desktops*, a saída natural foi migrar ou adaptar as mesmas para PC3. Essa operação de substituição foi armazenada para que na data estabelecida fosse realizada.

Enquanto isso, o administrador ativou a versão PC3 às 14h, para permitir a geração de instâncias:

activateVersion(PC3)

Essa operação modificou o estado da versão PC3, que passou para active:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema-version xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.inf.ufrgs.br/~fabiozs/workflow.xsd">
  <process>
    <sequence>
      <loop until="pedido = ok">
        <sequence>
          <conditional condition="novopedido">
            <true>
              <activity name="Iniciar pedido"/>
            </true>
            <false>
              <activity name="Revisar pedido"/>
            </false>
          </conditional>
          <parallel>
            <activity name="Aprov. gerente"/>
            <activity name="Verif. tecnica"/>
            <activity name="Verif. orcamento"/>
          </parallel>
        </sequence>
      </loop>
      <parallel>
        <activity name="Pedido tela laptop"/>
        <activity name="Pedido placas laptop"/>
        <activity name="Pedido gabinete laptop"/>
      </parallel>
      <activity name="Receb. componentes"/>
      <activity name="Montagem laptop"/>
      <parallel>
        <activity name="Envio"/>
        <activity name="Cobranca"/>
      </parallel>
    </sequence>
  </process>
</schema-version>
```

FIGURA 6.8 – Versão derivada de PC2 (`PC4.xml`)

```
...
<life-cycle> <!-- versão PC3 -->
  <status name="stable" itime="2001-09-10T09:00:00" ftime="2001-09-10T14:00:00"/>
  <status name="active" itime="2001-09-10T14:00:00"/>
</life-cycle>
...
```

Também a versão PC4, responsável pela montagem de *laptops*, deveria gerar instâncias, o que implicou numa operação de ativação, realizada às 14h15:

activateVersion(PC4)

Assim como em PC3, a ativação mudou o estado da versão PC4:

```
...
<life-cycle> <!-- versão PC4 -->
  <status name="stable" itime="2001-09-10T09:30:00" ftime="2001-09-10T14:15:00"/>
  <status name="active" itime="2001-09-10T14:15:00"/>
</life-cycle>
...
```

Quando o dia 15 de setembro chegou, a operação de substituição de versões que tinha sido armazenada foi executada. Assim, às 7h, o WFMS disparou a operação:

replaceVersion(PC2, PC3)

Essa operação realizou as seguintes tarefas:

1. verificou se PC2 poderia ser substituída (a mesma deveria estar no estado active, suspended ou discontinued);
2. verificou se PC3 poderia manter instâncias (permitido nos estados active, suspended e discontinued, embora neste último caso fosse desaconselhado);
3. mudou o estado em PC2 para being substituted:

```
...
<life-cycle> <!-- versão PC2 -->
  <status name="stable" itime="2001-04-25T14:00:00"
    ftime="2001-04-25T14:45:00"/>
  <status name="active" itime="2001-04-25T14:45:00"
    ftime="2001-09-15T07:00:00"/>
  <status name="being substituted" itime="2001-09-15T07:00:00"/>
</life-cycle>
...
```

4. as instâncias foram migradas ou adaptadas para a versão PC3. A migração foi realizada se a instância fosse parcialmente compatível com a nova versão. Já a adaptação se processou, na maioria dos casos, de forma manual;

Tomando como exemplo a instância do pedido cujo *id* era 189 (figura 6.9), aplicou-se a verificação da compatibilidade parcial entre ela e a versão PC3. Após a geração das árvores (figura 6.10), foi verificado que a instância seguia a mesma disposição de nodos da nova versão, até o ponto de execução da instância. Assim, a mesma poderia ser migrada para PC3. Intuitivamente, pode-se perceber que qualquer instância de PC2 que estivesse executando em algum ponto anterior às atividades de pedidos de componentes poderia migrar para PC3. No entanto, as instâncias que já tiveram passado por aquele ponto tiveram que ser adaptadas pelo administrador.

Cada instância que passou para PC3 teve de atualizar o esquema de base. Dessa forma, a instância cujo *id* era 189 foi modificada como segue:

```
<instance id="189">
  <schema-ref>
    <version name="PC2.xml" itime="2001-09-05T15:20:00"
      ftime="2001-09-15T07:05:00"/>
    <version name="PC3.xml" itime="2001-09-15T07:05:00"/>
  </schema-ref>
  <process>
    ...
  </process>
</instance>
```

Após 40 minutos do início da substituição, terminou a migração ou adaptação de todas as instâncias. Então, a versão PC2 foi excluída logicamente pelo WFMS, o que ocorreu de maneira automática:

```
...
<life-cycle> <!-- versão PC2 -->
  <status name="stable" itime="2001-04-25T14:00:00" ftime="2001-04-25T14:45:00"/>
  <status name="active" itime="2001-04-25T14:45:00" ftime="2001-09-15T07:00:00"/>
  <status name="being substituted" itime="2001-09-15T07:00:00"
    ftime="2001-09-15T07:40:00"/>
  <status name="logically removed" itime="2001-09-15T07:40:00"/>
</life-cycle>
...
```

Após esta data, algumas outras modificações ainda foram realizadas no processo. No entanto, para fins de estudo de caso, as situações descritas são suficientes para ilustrar

```
<?xml version="1.0" encoding="UTF-8"?>
<instance id="189">
  <schema-ref>
    <version name="PC2.xml" itime="2001-09-05T15:20:00"/>
  </schema-ref>
  <process>
    <sequence>
      <loop until="pedido = ok">
        <lap>
          <sequence>
            <conditional condition="novopedido">
              <true>
                <activity name="Iniciar pedido" status="completed"
                  itime="2001-09-05T15:30:00" ftime="2001-09-05T17:00:00"/>
              </true>
              <false>
                <activity name="Revisar pedido" status="discarded"/>
              </false>
            </conditional>
            <parallel>
              <activity name="Aprov. gerente" status="running"
                itime="2001-09-07T09:30:00"/>
              <activity name="Verif. tecnica" status="running"
                itime="2001-09-06T10:00:00"/>
              <activity name="Verif. orcamento" status="running"
                itime="2001-09-06T14:00:00"/>
            </parallel>
          </sequence>
        </lap>
      </loop>
      <activity name="Solic. componentes" status="not-ready"/>
      <activity name="Montagem" status="not-ready"/>
      <parallel>
        <activity name="Envio" status="not-ready"/>
        <activity name="Cobranca" status="not-ready"/>
      </parallel>
    </sequence>
  </process>
</instance>
```

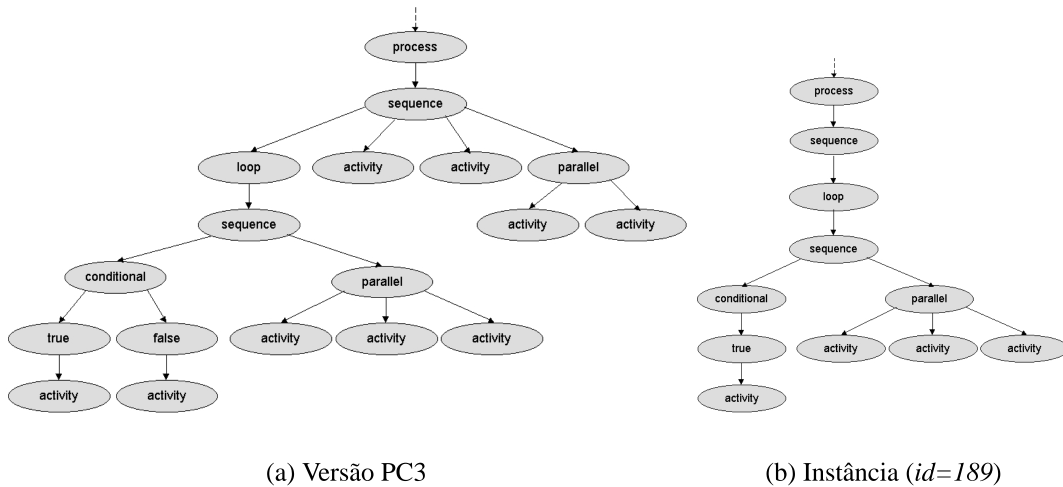
FIGURA 6.9 – Instância baseada em PC2

a importância da evolução de esquemas de *workflow*, bem como as implicações que tal evolução acarreta em outros componentes do sistema de gerência de *workflow*, especificamente nas versões e nas instâncias, representadas em XML. A figura 6.11 apresenta o esquema versionado PC ao final da análise do estudo de caso.

6.3 Considerações Finais

O estudo de caso apresentado neste capítulo ilustrou a utilização das estratégias descritas nos capítulos 3, 4 e 5 para a representação e evolução de *workflows* em XML. Foi utilizado um processo fictício de montagem de computadores, o qual foi modelado e modificado, de acordo com as especificações da empresa. O enfoque principal do estudo de caso foi dado ao versionamento, ou seja, partindo dessa característica, foi possível apresentar as versões e também as instâncias, bem como seus ciclos de vida, de forma integrada. Além disso, o versionamento permitiu que representações alternativas pudessem estar ativas ao mesmo tempo, possibilitando uma maior flexibilidade do processo.

O processo foi modelado de acordo com a linguagem proposta no capítulo 3, não apresentando problemas para sua representação. As características de correção, faci-



(a) Versão PC3 (b) Instância (*id=189*)

FIGURA 6.10 – Árvores da versão PC3 e da instância 189

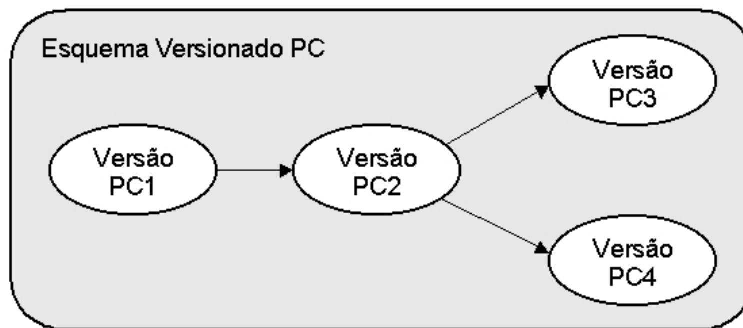


FIGURA 6.11 – Esquema versionado PC

lidade de alteração, simplicidade e clareza puderam ser constatadas, uma vez que o fluxo pôde ser facilmente construído e modificado. Além disso, as estruturas denotaram perfeitamente o roteamento das atividades, sem causar ambigüidade ou problemas de interpretação.

Tomando como base o esquema criado, várias instâncias puderam ser disparadas, representando pedidos dos clientes. As instâncias indicaram o andamento das atividades no tempo, de acordo com o fluxo pré-determinado.

Além disso, ocorreram modificações no processo, que foram devidamente refletidas no esquema de *workflow*. Cada modificação gerou uma nova versão do esquema, o que pôde ser feito por meio da operação de derivação, que utiliza documentos XUpdate para a representação das mudanças no esquema. Também foram utilizadas operações para mudança de estados das versões, necessárias para a representação do ciclo de vida das mesmas. Em alguns casos, as instâncias tiveram que migrar ou se adaptar a uma nova versão, em virtude da substituição de versões não mais válidas. Utilizou-se a verificação da compatibilidade parcial para a determinação da migração das instâncias, sendo que as instâncias compatíveis puderam passar naturalmente para a nova versão, enquanto que as incompatíveis tiveram de ser manualmente adaptadas.

Deve-se destacar também que as informações de controle das versões e das instâncias foram armazenadas adequadamente, permitindo a realização de consultas

ao histórico. Da mesma forma, o documento de esquema versionado possibilitou a representação da hierarquia de versões.

Desta maneira, o estudo de caso realizado atingiu o objetivo de demonstrar a utilização e a aplicabilidade das estratégias propostas neste trabalho em um ambiente real, embora tanto a empresa quanto o processo sejam fictícios. Além disso, pelo fato da linguagem possuir um alto grau de estruturação e organização das atividades, é fortemente indicada para a modelagem de *workflows* administrativos ou de produção, sendo pouco recomendada para *workflows ad-hoc*, que possuem coordenação de atividades praticamente manual e, por conseqüência, baixa estruturação.

Cabe ainda ressaltar que o administrador de *workflow* é o responsável por definir e manter o escopo do processo, ou seja, deve ser capaz de identificar quando as modificações em um processo o tornam desfigurado, de forma que uma versão não possua mais relação com o objetivo do processo. Neste caso, é conveniente que se crie um novo processo, mantendo em um mesmo esquema as versões relacionadas entre si.

7 Conclusão

Evolução de esquemas de *workflow* é um importante tema de pesquisa atualmente, pois visa oferecer alternativas para a eficiente representação das mudanças ocorridas nos processos de negócio. Ao mesmo tempo, a linguagem XML vem sendo muito utilizada para a modelagem de *workflows*, por suas características de interoperação e pela flexibilidade. No entanto, pelo fato dos modelos e linguagens para a representação de *workflow* em XML serem recentes, ainda não dispõem de mecanismos adequados para tratar a evolução de seus esquemas. Dessa maneira, a contribuição central deste trabalho é a definição de uma estratégia para o tratamento da evolução de esquemas de *workflow* representados em XML. Esta estratégia está baseada no conceito de versionamento, permitindo que sejam armazenadas várias versões de um mesmo esquema, além de possibilitar análises históricas.

É proposta uma linguagem para representação de esquemas de *workflow*, uma vez que as linguagens disponíveis não são adequadas para o tratamento da evolução, de acordo com os critérios de simplicidade, correção, clareza e facilidade de alteração. Além disso, para o propósito da evolução, são identificadas operações genéricas de modificação de fluxo (inserção, remoção, movimentação, troca de estrutura), as quais são implantadas de acordo com a linguagem XUpdate, própria para a aplicação de mudanças sobre documentos XML. Para cada operação definida, é realizada uma verificação da possibilidade de sua implantação sobre as estruturas da linguagem, em termos de correção. Com a utilização da linguagem de representação proposta e das operações definidas, constata-se que a evolução de esquemas se torna mais simples do que quando aplicada aos outros modelos.

É proposta ainda uma estrutura para a representação de instâncias, as quais são geradas a partir dos esquemas construídos de acordo com a linguagem. De acordo com a proposta, cada instância armazena sua própria execução. Além disso, é definido um conjunto de estados aplicáveis às atividades, para registro do andamento das mesmas. É proposto também um método que indica se a migração de uma instância para uma nova representação é possível, no caso da ocorrência de uma evolução. Esse método baseia-se no conceito de compatibilidade, que é a verificação do caminho já percorrido pela instância. Metadados também são armazenados em cada instância, a fim de possibilitar o registro dos esquemas nos quais a mesma se baseou.

Para permitir o armazenamento das versões de esquema geradas, é definido um modelo de versionamento de esquemas de *workflow*. É apresentado um conjunto de estados nos quais as versões podem permanecer, constituindo-se no seu ciclo de vida. O modelo é composto ainda de uma estrutura de esquema versionado, responsável por registrar a relação existente entre as versões, e de operações para a criação de novo esquema, derivação de versão e mudança de estados. Dessa maneira, várias versões podem estar ativas ao mesmo tempo, o que possibilita a existência de alternativas para um mesmo esquema.

De maneira a avaliar a aplicabilidade dessa estratégia de evolução proposta, é realizado um estudo de caso envolvendo um processo de negócio fictício. Constata-se que o mesmo é adequadamente modelado e que as modificações ocorridas no mesmo são representadas corretamente. A cada operação aplicada sobre os componentes do esquema versionado, são mostradas todas as implicações nas outras estruturas, demonstrando a conexão existente entre elas.

Uma das grandes características das propostas apresentadas neste trabalho é que todos os componentes (versões de esquema, operações de modificação, instâncias e esquema versionado) são documentos XML, padronizando a forma de representação dos mesmos em um WFMS.

As principais contribuições deste trabalho são, portanto:

- a definição de uma linguagem para a representação de esquemas de *workflow* em XML que evoluam no tempo;
- a criação de uma estrutura para representação de instâncias e de mecanismos para o tratamento da migração para um novo esquema;
- a definição de um modelo de versionamento de esquemas de *workflow*, para o armazenamento de versões de esquema alternativas e do histórico das mesmas.

No entanto, algumas questões não estão completamente definidas ou não são abordadas neste trabalho: o problema da adaptação das instâncias incompatíveis, a verificação de falsas atualizações (seção 3.2.3), o problema das instâncias passíveis de migração mesmo sendo parcialmente incompatíveis (seção 4.3), a migração de uma instância particular para outra versão qualquer, consultas aos históricos armazenados, entre outras.

Entre algumas sugestões para a complementação deste trabalho, pode-se destacar:

- definição de um ambiente que permita a utilização adequada de todas as operações e componentes de *workflow* propostos, além da integração com um sistema de gerência;
- definição de uma linguagem mais apropriada para a representação das operações de modificação, uma vez que XUpdate é uma linguagem muito genérica;
- utilização de critérios de compatibilidade mais amplos, permitindo um maior número de instâncias migradas;
- criação de mecanismos de consulta aos históricos;
- tratamento de outros aspectos relacionados aos modelos de *workflow*, como fluxo de dados e modelo organizacional;
- formalização das propostas e validação em um caso existente, para conclusões mais precisas;
- interoperabilidade com outras linguagens de representação.

Anexo XML *Schema* da Linguagem de Representação de *Workflow*

A seguir, segue a descrição completa da linguagem de representação de *workflow* em XML *Schema*.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:element name="schema-version" type="schemaType"/>
  <!--tipos-->
  <xsd:group name="components">
    <xsd:choice>
      <xsd:element name="activity" type="activityType"/>
      <xsd:element name="sequence" type="sequenceType"/>
      <xsd:element name="parallel" type="parallelType"/>
      <xsd:element name="conditional" type="conditionalType"/>
      <xsd:element name="loop" type="loopType"/>
    </xsd:choice>
  </xsd:group>
  <xsd:complexType name="schemaType">
    <xsd:sequence>
      <xsd:element name="life-cycle" type="lifecycleType" minOccurs="0"/>
      <xsd:element name="process" type="processType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="lifecycleType">
    <xsd:sequence>
      <xsd:element name="status" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name="name" use="required">
            <xsd:simpleType>
              <xsd:restriction base="xsd:string">
                <xsd:enumeration value="stable"/>
                <xsd:enumeration value="active"/>
                <xsd:enumeration value="suspended"/>
                <xsd:enumeration value="discontinued"/>
                <xsd:enumeration value="being substituted"/>
                <xsd:enumeration value="logically removed"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:attribute>
          <xsd:attribute name="itime" type="xsd:dateTime" use="required"/>
          <xsd:attribute name="ftime" type="xsd:dateTime" use="optional"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="processType">
    <xsd:group ref="components"/>
  </xsd:complexType>
  <xsd:complexType name="activityType">
    <xsd:attribute name="name" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:complexType name="sequenceType">
    <xsd:sequence maxOccurs="unbounded">
      <xsd:group ref="components"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="parallelType">
    <xsd:sequence minOccurs="2" maxOccurs="unbounded">
      <xsd:group ref="components"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="conditionalType">
    <xsd:choice>
      <xsd:sequence>
        <xsd:element name="true">
          <xsd:complexType>
```

```
          <xsd:group ref="components"/>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="false" minOccurs="0">
        <xsd:complexType>
          <xsd:group ref="components"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:sequence>
      <xsd:element name="false">
        <xsd:complexType>
          <xsd:group ref="components"/>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="true">
        <xsd:complexType>
          <xsd:group ref="components"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:choice>
  <xsd:attribute name="condition" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="loopType">
  <xsd:group ref="components"/>
  <xsd:attribute name="until" type="xsd:string" use="required"/>
</xsd:complexType>
</xsd:schema>
```

Referências

AALST, W. M. P. van der. The Application of Petri Nets to Workflow Management. **The Journal of Circuits, Systems and Computers**, Singapore, v.8, n.1, p.21–66, 1998.

AALST, W. M. P. van der; BARROS, A. P.; HOFSTEDE, A. H. M. ter; KIEPUSZEWSKI, B. Advanced Workflow Patterns. In: INTERNATIONAL CONFERENCE ON COOPERATIVE INFORMATION SYSTEMS, CoopIS, 7., 2000, Eilat, Israel. **Proceedings...** Berlin: Springer-Verlag, 2000. p.18–29.

AALST, W. M. P. van der; HOFSTEDE, A. H. M. ter; KIEPUSZEWSKI, B.; BARROS, A. P. **Workflow Patterns**. Eindhoven: Eindhoven University of Technology, 2002. Relatório Técnico. Disponível em: <http://tmitwww.tm.tue.nl/research/patterns>. Acesso em: abr. 2002.

AALST, W. M. P. van der; KUMAR, A. **XML Based Schema Definition for Support of Inter-organizational Workflow**. Boulder: University of Colorado, 2000. Relatório Técnico. Disponível em: <http://spot.colorado.edu/∼akhil/pubs.html>. Acesso em: jan. 2002.

BARTHELMESS, P.; WAINER, J. Workflow Systems: a Few Definitions and a Few Suggestions. In: CONFERENCE ON ORGANIZATIONAL COMPUTING SYSTEMS, COOCS, 1995, Milpitas, USA. **Proceedings...** New York: ACM Press, 1995. p.138–147.

CASATI, F.; CERI, S.; PERNICI, B.; POZZI, G. Conceptual Modeling of Workflows. In: INTERNATIONAL CONFERENCE ON OBJECT-ORIENTED AND ENTITY-RELATIONSHIP MODELING, OO-ER, 14., 1995, Gold Coast, Australia. **Proceedings...** Berlin: Springer-Verlag, 1995. p.341–354.

CASATI, F.; CERI, S.; PERNICI, B.; POZZI, G. Workflow Evolution. **Data & Knowledge Engineering**, Amsterdam, v.24, n.3, p.211–238, Jan. 1998.

CHRISTOPHIDES, V.; HULL, R.; KUMAR, A. Querying and Splicing of XML Workflows. In: INTERNATIONAL CONFERENCE ON COOPERATIVE INFORMATION SYSTEMS, CoopIS, 9., 2001, Trento, Italy. **Proceedings...** Berlin: Springer-Verlag, 2001. p.386–402.

EDELWEISS, N. Bancos de Dados Temporais: Teoria e Prática. In: JORNADA DE ATUALIZACAO EM INFORMATICA, JAI, 17., 1998, Belo Horizonte, Brasil. **Anais...** Belo Horizonte: SBC, 1998. p.225–282.

ELLIS, C.; KEDDARA, K.; ROZENBERG, G. Dynamic Change Within Workflow Systems. In: CONFERENCE ON ORGANIZATIONAL COMPUTING SYSTEMS, COOCS, 1995, Milpitas, USA. **Proceedings...** New York: ACM Press, 1995. p.10–21.

GALANTE, R.; EDELWEISS, N.; SANTOS, C. S. dos. Change Management for a Temporal Versioned Object-Oriented Database. In: INTERNATIONAL WORKSHOP ON EVOLUTION AND CHANGE IN DATA MANAGEMENT, ECDM, 2., 2002, Tampere, Finland. **Proceedings...** Berlin: Springer-Verlag, 2002. p.1–12.

GEORGAKOPOULOS, D.; HORNICK, M.; SHETH, A. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. **Distributed and Parallel Databases**, Dordrecht, v.3, n.2, p.119–153, Apr. 1995.

GOLENDZINER, L. G. **Um Modelo de Versões para Banco de Dados Orientados a Objetos**. 1995. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre, Brasil.

HORN, S.; JABLONSKI, S. An Approach to Dynamic Instance Adaption in Workflow Management Applications. In: TOWARDS ADAPTIVE WORKFLOW SYSTEMS, 1998, Seattle, USA. **Proceedings...** [S.l.: s.n.], 1998. Disponível em: <http://ccs.mit.edu/klein/cscw98/>. Acesso em: dez. 2001.

IBM CORPORATION. **Web Services Flow Language (WSFL 1.0)**. 2001. Disponível em: <http://www-3.ibm.com/software/solutions/webservices/documentation.html>. Acesso em: abr. 2002.

JOERIS, G.; HERZOG, O. **Managing Evolving Workflow Specifications with Schema Versioning and Migration Rules**. Bremen, Germany: University of Bremen, 1999. Relatório Técnico. Disponível em: <http://www.informatik.uni-bremen.de/~joeris/publist.html>. Acesso em: abr. 2002.

KIEPUSZEWSKI, B.; HOFSTEDE, A. H. M. ter; BUSSLER, C. On Structured Workflow Modelling. In: CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, CAiSE, 12., 2000, Stockholm, Sweden. **Proceedings...** Berlin: Springer-Verlag, 2000. p.431–445.

KRADOLFER, M. **A Workflow Metamodel Supporting Dynamic, Reuse-based Model Evolution**. 2000. Thesis (Ph.D. Program in Informatics) — University of Zurich, Zurich, Switzerland.

KRADOLFER, M.; GEPPERT, A. Dynamic Workflow Schema Evolution Based on Workflow Type Versioning and Workflow Migration. In: INTERNATIONAL CONFERENCE ON COOPERATIVE INFORMATION SYSTEMS, CoopIS, 4., 1999, Edinburgh, Scotland. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 1999. p.104–114.

KUMAR, A.; ZHAO, J. L. Workflow Support for Electronic Commerce Applications. **Decision Support Systems**, Amsterdam, v.32, n.3, p.265–278, Jan. 2002.

MORO, M. **Modelo Temporal de Versões**. 2001. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre, Brasil.

REICHERT, M.; DADAM, P. ADEPT$_{flex}$ – Supporting Dynamic Changes of Workflows Without Loosing Control. **Journal of Intelligent Information Systems**, Dordrecht, v.10, n.2, p.93–129, Mar./Apr. 1998.

SADIQ, S.; SADIQ, W.; ORLOWSKA, M. Pockets of Flexibility in Workflow Specification. In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, ER, 20., 2001, Yokohama, Japan. **Proceedings...** Berlin: Springer-Verlag, 2001. p.513–526.

SADIQ, S. W.; MARJANOVIC, O.; ORLOWSKA, M. E. Managing Change and Time in Dynamic Workflow Processes. **International Journal of Cooperative Information Systems**, Singapore, v.9, n.1-2, p.93–116, Mar./June 2000.

SADIQ, W.; ORLOWSKA, M. E. **Modeling and Verification of Workflow Graphs**. Brisbane, Australia: The University of Queensland, 1996. Relatório Técnico. Disponível em: <http://www.dstc.edu.au/praxis/publications/>. Acesso em: abr. 2002.

SADIQ, W.; ORLOWSKA, M. E. Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models. In: CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, CAiSE, 11., 1999, Heidelberg, Germany. **Proceedings...** Berlin: Springer-Verlag, 1999. p.195–209.

STAKEN, K. **XML:DB XUpdate Use Cases**. 2000. Disponível em: <http://www.xmldatabases.org/projects/XUpdate-UseCases/>. Acesso em: jun. 2002.

TATARINOV, I.; IVES, Z. G.; HALEVY, A. Y.; WELD, D. S. Updating XML. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 2001, Santa Barbara, USA. **Proceedings...** New York: ACM Press, 2001. p.413–424.

WESKE, M. Formal Foundation and Conceptual Design of Dynamic Adaptions in a Workflow Management System. In: ANNUAL HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, HICSS, 34., 2001, Maui, Hawaii. **Proceedings...** [S.l.: s.n.], 2001. Disponível em: <http://computer.org/proceedings/hicss/0981/volume%207/0981toc.htm>. Acesso em: maio 2002.

WORKFLOW MANAGEMENT COALITION. **The Workflow Reference Model**. 1995. Disponível em: <http://www.wfmc.org/standards/docs.htm>. Acesso em: nov. 2001.

WORKFLOW MANAGEMENT COALITION. **Interface 1:** Process Definition Interchange, Process Model. 1999. Disponível em: <http://www.wfmc.org/standards/docs.htm>. Acesso em: nov. 2001.

WORKFLOW MANAGEMENT COALITION. **Terminology & Glossary**. 1999. Disponível em: <http://www.wfmc.org/standards/docs.htm>. Acesso em: dez. 2001.

WORKFLOW MANAGEMENT COALITION. **Workflow Standard - Interoperability Wf-XML Binding, Version 1.1**. 2001. Disponível em: <http://www.wfmc.org/standards/docs.htm>. Acesso em: mar. 2002.

WORKFLOW MANAGEMENT COALITION. **XML Process Definition Language, Version 1.0**. 2002. Disponível em: <http://www.wfmc.org/standards/docs.htm>. Acesso em: nov. 2002.

WORLD WIDE WEB CONSORTIUM. **XML Path Language (XPath)**. 1999. Disponível em: <http://www.w3.org/TR/xpath>. Acesso em: maio 2002.

WORLD WIDE WEB CONSORTIUM. **Extensible Markup Language (XML) 1.0 (Second Edition)**. 2000. Disponível em: <http://www.w3.org/TR/REC-xml>. Acesso em: mar. 2002.

WORLD WIDE WEB CONSORTIUM. **XQuery 1.0:** An XML Query Language. 2002. Disponível em: <http://www.w3.org/TR/xquery>. Acesso em: maio 2002.

WORLD WIDE WEB CONSORTIUM. **Document Object Model (DOM)**. 2002. Disponível em: <http://www.w3.org/DOM>. Acesso em: jan. 2003.

XML:DB. **XUpdate - XML Update Language**. 2000. Disponível em: <http://www.xmldb.org/xupdate>. Acesso em: maio 2002.

ZSCHORNACK, F.; EDELWEISS, N. Uma Linguagem para Representação de Workflow em XML com Suporte a Evolução e Versionamento de Esquemas. In: CONFERENCIA LATINOAMERICANA DE INFORMATICA, CLEI, 28., 2002, Montevideo, Uruguay. **Articulos...** Montevideo: Universidad de la Republica, 2002. 1 CD-ROM.