Aumentando a Migração de Instâncias Não Complacentes em Sistemas de Informação Cientes de Processos

Rafael Liberato Roberto

Tese apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Doutor em Ciências

Programa: Pós-Graduação em Ciência da Computação
Orientador: Prof. Dr. João Eduardo Ferreira

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da Fundação Araucária - Projeto DINTER - Doutorado Interinstitucional UTFPR/USP

São Paulo, Agosto de 2016

Aumentando a Migração de Instâncias Não Complacentes em Sistemas de Informação Cientes de Processos

Esta é a versão original da tese elaborada pelo candidato (Rafael Liberato Roberto), tal como submetida à Comissão Julgadora.

Resumo

LIBERATO, R.. **Aumentando a Migração de Instâncias Não Complacentes em Sistemas de Informação Cientes de Processos**. 2016. 101 f. Tese (Doutorado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2016.

O dinamismo apresentado no meio corporativo atual, tem exigido cada vez mais que as organizações mantenham seus processos em constante transformação para garantir competitividade no mercado. Diante desse cenário, as organizações têm recorrido a sistemas de informação que sejam capazes de prover apoio computacional para essas frequentes transformações no processo. Embora os Sistemas de Informação Cientes de Processos proporcionem um apoio flexível para o gerenciamento de processos, a evolução de um processo ainda apresenta desafios a serem superados. Dentre esses desafios, nesta tese, objetivou-se tratar as instâncias em execução que não são complacentes com as novas especificações do esquema modificado, mas que necessitam serem migradas. Para isso, desenvolveu-se um mecanismo estendendo a abordagem WED-*flow* (*Workflow, Event Processing and Data-flow*) que é capaz de migrar todas as instâncias em execução. Para habilitar a migração das instâncias não complacentes, o mecanismo promove ajustes automáticos e personalizados ao definir rotinas de recuperação com base na estratégia de *rollback* parcial. Além disso, o mecanismo reduz os efeitos colaterais inerentes ao *rollback* parcial com a reutilização de estados de dados durante a migração da instância para o esquema modificado. A solução proposta apresentou resultados satisfatórios nos experimentos realizados sob três perspectivas. Na primeira perspectiva, o mecanismo se mostrou capaz de migrar todas as instâncias em execução, independentemente do seu estado. Na segunda perspectiva, a análise sobre o tempo de execução indicou que a solução é factível para aplicação prática em cenários reais. Por fim, na terceira perspectiva, o reuso de estados de dados mostrou-se promissor ao reduzir significativamente o tempo de execução das instâncias após sua migração.

Palavras-chave: Adaptação de instâncias, Workflow dinâmico, Processos de Negócio, Sistema de Informação Ciente do Processo, Evolução de esquemas, Migração de instâncias, Instâncias não complacentes.

Abstract

LIBERATO, R.. **Increasing the Migration of Non-compliant Instances in Process-Aware Information Systems.** 2016. 101 f. Tese(Doutorado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2016.

The dynamic business environment has required organizations to maintain their business processes in constant transformation to ensure market competitiveness. In this scenario, organizations have adopted information systems that are able to provide computational support for these frequent changes in the process. Although Process-Aware Information Systems provide flexible support for process management, the process evolution still poses challenges to overcome. Among these challenges, this thesis aimed to address the running instances that are not compliant with the new specifications of the modified scheme, but that need to be migrated. For this, we developed a mechanism that extend the WED-flow approach (Workflow, Event Processing and Data-flow) for enabling the migration of all running instances. To enable migration of non-compliant instances, the mechanism realizes automatic and customized adjustments by defining recovery routines based on partial rollback strategy. Moreover, the mechanism reduces the side effects attached to partial rollback with the reuse of data states during instance migration to the modified schema. The proposed solution achieved satisfactory results in the experiments conducted from three perspectives. In the first perspective, the mechanism has been able to migrate all running instances, regardless of their compliance state. In the second perspective, the analysis of the execution time indicates that the solution is feasible for practical application in real scenarios. Finally, the third perspective, the reuse of data states has shown promise by reducing significantly the execution time of the instances after their migration.

Keywords: Dynamic change, Dynamic workflow, Business process, Process-aware information system, Schema Evolution, Instance migration, Non-compliant instances.

Sumário

Lista de Abreviaturas

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
BPA	*Basic Process Algebra*
DATA	*Database Modeling, Transactions, and Data Analysis*
ECA	Evento, Condição e Ação
ETM	*Extended Transaction Models*
FIFO	*First In, First Out*
LLT	*Long-lived Transactions*
LTL	Lógica Temporal Linear
PAIS	*Process-aware Infomation Systems*
XML	*Extensible Markup Language*
WED-*flow*	*Workflow, Event processing and Data-flow*

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O sucesso de grandes empresas e organizações no meio corporativo está cada vez mais relacionado a sua habilidade de reagir a mudanças em seus ambientes (Reichert e Weber, 2012). Tais mudanças tornam-se necessárias, por exemplo, quando novas leis entram em vigor, otimizações ou ajustes nos processos de negócio são necessárias, reações a novas tendências de mercado são requeridas, entre outros motivos. A transformação contínua do negócio — na qual a mudança é a regra, não a exceção — é fundamental para manter a competitividade e proporcionar produtos e serviços personalizados (Carlsen *et al.*, 1997). Portanto, é crescente a exigência no meio corporativo de que as organizações sejam capazes de refinar seus processos de negócio de forma rápida e flexível. Diante desse cenário, essas organizações têm buscado apoio computacional para modelagem e execução de seus processos de negócio. No entanto, os sistemas de informação tradicionais não provêm tal flexibilidade pois, de modo geral, o controle do processo é codificado diretamente na aplicação. Essa característica faz com que qualquer alteração no processo desencadeie esforço adicional de programação, tornando o desenvolvimento e manutenção do processo algo complexo e propenso a erros (Rinderle, 2004).

A carência de soluções flexíveis para manutenção dos processos de negócio tem estimulado a evolução dos sistemas de informação tradicionais de modo que eles tenham acesso a informações do processo por meio de modelos abstratos. Essa evolução resultou no termo "ciente do processo" ou "conhecedor do processo" (*process-aware*) e definiu uma nova geração de sistemas de informação. Os Sistemas de Informação Cientes de Processos (PAIS, do inglês *Process-Aware Information Systems*) surgiram para prover apoio mais flexível e dinâmico para o gerenciamento de processos de negócio (Reichert e Weber, 2012). Dois dos principais exemplos desta nova geração de PAIS são os Sistemas de Gerenciamento de *Workflow* (WfMS, do inglês *Workflow Management System*) e os Sistemas de Gerenciamento de Processo de Negócio (BPMS, do inglês *Business Process Management System*).

A principal característica de um PAIS é a separação entre a lógica do processo e o código da aplicação, isto é, a lógica do processo é explicitamente especificada por um modelo e não é embutida no código da aplicação (Rinderle, 2004). Esse modelo fornece uma descrição lógica do processo em um nível abstrato que é utilizado para guiar sua execução. Diferentes abordagens têm sido adotadas para especificação de processos de negócio. Destacam-se entre elas os modelos formais (Rede de Petri, Álgebra de Processo, Teoria dos Grafos, etc.) e os modelos declarativos (orientados a regras ou restrições). De modo geral, esses modelos descrevem o processo como uma sequência de atividades e um conjunto de restrições que determinam explicitamente o controle do fluxo de execução. Essa descrição é usualmente chamada de *esquema* do processo. Assim, dado um esquema W, diferentes instâncias do processo de negócio $I_1^W, \ldots, I_n^W$ podem ser criadas e executadas pelo

PAIS.

Embora a separação da lógica do processo do código da aplicação qualifica um PAIS mais flexível em comparação aos sistemas de informação tradicionais, ainda existem desafios a serem superados nesse quesito. Um dos principais desafios refere-se à evolução do processo de negócio. Mais especificamente, em como tratar as instâncias que estão execução quando uma modificação no esquema do processo é concretizada. Idealmente, todas as instâncias em execução deveriam ser migradas para o esquema modificado para que elas possam usufruir das alterações propostas.

1.1 Caracterização do problema

Atualmente, os PAIS são somente adequados aos processos de negócio cuja estrutura não possui variações de comportamento. Os PAIS provêm apoio rudimentar às mudanças dinâmicas estruturais e comportamentais do processo (Aalst, 2001; Ellis *et al.*, 1995). No entanto, como discutido anteriormente, a transformação contínua do negócio é um requisito chave para as organizações. Dessa forma, a aplicabilidade do PAIS torna-se limitada se eles forem rígidos ou intolerantes às alterações nos esquemas do processo previamente definidos pelo projetista (Reichert e Dadam, 1997).

Uma alteração pode ter como objetivo alcançar um grupo específico de instâncias (alteração *ad-hoc*) ou alcançar todas as instâncias do processo (alterações evolucionárias). Nas alterações evolucionárias, as instâncias que são inicializadas após a efetivação da alteração do esquema não causam problemas. As instâncias são criadas e executadas segundo o novo esquema. No entanto, as instâncias que estão em execução produzem um dos cenários mais desafiadores no contexto da evolução de processos de negócio: a migração das instâncias em execução para o esquema modificado. A comunidade tem adotado diferentes termos para referenciar esse problema, tais como evolução de esquemas dinâmicos (Casati *et al.*, 1998) e mudanças dinâmicas[1] (Ellis *et al.*, 1995). Por uma questão de clareza, neste trabalho, optou-se por adotar o termo **adaptação de instâncias**. Em síntese, o problema refere-se a como migrar as instâncias que estão em execução no esquema de origem para o esquema modificado.

Para assegurar uma migração segura, isto é, que a instância não cause erros ou inconsistências no esquema modificado, a maioria dos PAIS define um critério para validar a migração das instâncias. O critério de migração estabelece condições que avaliam se uma instância é complacente a um determinado esquema. Caso seja complacente, a instância está apta para ser migrada. As instâncias avaliadas como não complacentes são normalmente excluídas da migração e finalizam suas execuções no esquema de origem. Dessa forma, o problema de adaptação de instâncias pode ser descrito como:

> Considere uma instância i originalmente iniciada no esquema W, isto é, $i \in I^W$. Assuma também que uma alteração Δ transforma o esquema W em um esquema estruturalmente consistente W'. Então i pode ser migrada para W' tal que $i \in I^{W'}$ se, e somente se, i é complacente com W'.

No contexto das alterações evolucionárias, o problema da adaptação de instâncias introduz questões desafiadoras aos PAIS sob duas perspectivas: (i) identificar e migrar as instâncias complacentes com o esquema modificado; e (ii) tratar das instâncias não complacentes com a nova versão do processo de negócio. Na primeira perspectiva, o PAIS deve ser capaz de avaliar o estado

[1] do inglês *Dynamic Changes*

de complacência de uma instância em execução diante das alterações aplicadas na nova versão do esquema do processo. Para isso, o PAIS deve definir um critério de migração capaz de determinar se a instância pode ou não ser transferida com segurança para o esquema modificado. Na segunda perspectiva, o PAIS deve prover alternativas para apoiar as instâncias que não são complacentes com o esquema modificado. Em um cenário ideal, todas as instâncias em execução deveriam ser migradas para novo esquema.

Diferentes soluções têm sido propostas na literatura para tratar o problema da adaptação de instâncias. Na primeira perspectiva, eficientes estratégias têm sido apresentadas para aplicar o critério de migração na avaliação do estado de complacência das instâncias. Um amplo estudo contendo a análise dos principais critérios de migração adotados pelos PAIS pode ser encontrado em (Rinderle *et al.*, 2003). Considerando os desafios introduzidos na segunda perspectiva, as contribuições têm sido na direção de ampliar a capacidade de migração dos PAIS com a tentativa de reduzir o número de instâncias avaliadas como não complacentes.

Em resumo, a literatura dispõe de soluções consolidadas capazes de verificar o estado de complacência de uma instância em execução e migrá-la caso ela seja complacente com o esquema modificado. No entanto, ainda existe uma lacuna a ser preenchida no tratamento das instâncias não complacentes. Mais especificamente, não há uma solução satisfatória para cenários que exigem a migração imediata de todas as instâncias diante da evolução do processo de negócio. As soluções atuais são ou muito restritivas ou não aplicáveis na prática para tratar as instâncias não complacentes. Idealmente, o PAIS deveria ser capaz de promover ajustes de forma automática nas instâncias não complacentes de modo que elas tornem-se aptas para serem migradas.

1.2 Questão de Pesquisa

Diante da caracterização do problema, a questão geral abordada por esta tese é:

QP Como ampliar de forma automática a capacidade adaptativa de um PAIS na migração de instâncias que não são complacentes com uma nova versão do esquema do processo?

1.3 Hipótese e sua Justificativa

Considerando a questão de pesquisa definida e as limitações dos PAIS em tratar as instâncias não complacentes diante de uma alteração do esquema do processo de negócio, a hipótese que norteou o desenvolvimento desta tese foi:

> *Um PAIS cujo modelo de processo seja orientado a dado, combinando conceitos de eventos e modelos transacionais avançados tem sua capacidade de migração ampliada em relação às abordagens tradicionais.*

Embora a nova geração dos sistemas de informação, com o surgimento dos PAIS, tenha trazido flexibilidade e dinamismo para o gerenciamento de processos, essa mesma flexibilidade não é constatada quando o processo precisa ser alterado. Nota-se que PAIS cujos modelos são exclusivamente orientados a processo (*process-oriented*) têm dificuldades quando se deparam com situações que destoam do que foi inicialmente projetado. Assim como na migração de instâncias não complacentes,

outro exemplo dessa dificuldade pode ser observado no tratamento de exceções, tais como cancelamentos e falhas. Em (Ferreira *et al.*, 2010b) e (Ferreira *et al.*, 2012) os autores discutem sobre essa dificuldade enfrentada pelas abordagens tradicionais para tratar situações que desviam a execução do caminho projetado. Além disso, os autores propõem o WED-*flow* (*Workflow, Event processing and Data-flow*) um arcabouço alternativo para modelagem de processos de negócio que amplia a flexibilidade no tratamento de exceções. Esse arcabouço tem como característica a integração de quatro importantes elementos de processos (eventos, dados, condições e transições) além de fornecer uma perspectiva transacional adaptada ao contexto de processos de negócio. Mais detalhes sobre a abordagem WED-*flow* serão descritas na Seção 2.4.2.

Diante da flexibilidade provida pela abordagem WED-*flow* no tratamento de exceções, considerou-se que a integração dos conceitos proposta pelo arcabouço para modelar processos de negócio, poderiam ampliar a capacidade de migração de instâncias no contexto do problema de evolução de esquemas. Em especial, a hipótese tem sua justificativa apoiada em duas características específicas:

(1) **Estados de dados bem definidos.** Cada execução de atividade no processo resulta na produção de um novo estado de dado externalizando as modificações realizadas no dado. Dessa forma, considerou-se que, pelo fato da instância possuir estados de dados bem definidos a cada passo de sua execução, poderia ser possível reutilizar alguns desses resultados durante sua migração para o esquema modificado.

(2) **Propriedades transacionais.** Com a perspectiva transacional adaptada ao contexto de processos de negócio, considerou-se que seria possível definir rotinas de recuperação automática para promover ajustes nas instâncias não complacentes.

1.4 Objetivos

O principal objetivo deste trabalho é prover uma solução que preencha uma lacuna no contexto do problema de adaptação de instâncias. Essa lacuna é caraterizada pela ausência de uma solução satisfatória que apresente alternativas para migração automática das instâncias avaliadas como não complacentes com o esquema modificado. Para alcançar esse objetivo, a abordagem WED-*flow* (Ferreira *et al.*, 2010b, 2012) foi estendida com o desenvolvimento de um mecanismo de migração que é capaz de promover automaticamente ajustes nas instâncias não complacentes de modo a habilitar suas respectivas migrações para o esquema modificado. Os ajustes automáticos promovidos pelo mecanismo são realizados com base na estratégia de *rollback* parcial. Além disso, propôs-se a reutilização de estados de dados para reduzir os efeitos colaterais causados pelo *rollback* parcial.

1.5 Publicações

No decorrer do curso de doutoramento foram publicados alguns trabalhos relacionados ao desenvolvimento desta tese. São eles:

- Rafael Liberato, André Luis Schwerz, Osvaldo Kotaro Takai e João Eduardo Ferreira. Gerenciador de dados compartilhados entre processos de negócio: Data-control process para wed-tool. *Em VIII Workshop Brasileiro em Gestão de Processos de Negócios*, volume 02 do WBPM 2014, páginas 26-33, Londrina, PR, Brasil (Liberato *et al.*, 2014).

- Rafael Liberato, André Luis Schwerz, Calton Pu e João Eduardo Ferreira. Abordagem transacional para padrões de controle de fluxo de processos científicos e de negócios. *Em XXX Simpósio Brasileiro de Banco de Dados*, SBBD 2015, páginas 15-20, Petrópolis, RJ, Brasil (Liberato *et al.*, 2015).

- Rafael Liberato, André Luis Schwerz, Calton Pu e João Eduardo Ferreira. Increasing the Migration of Non-compliant Instances in Adaptive PAIS. *IEEE Transactions on Services Computing*. Manuscrito submetido para publicação, 2016 (Liberato *et al.*, 2016).

1.6 Organização da Tese

O restante deste trabalho está organizado da seguinte forma. No Capítulo 2 são apresentados os conceitos e abordagens referentes a área de gerenciamento de processos de negócio. O Capítulo 3 aborda os trabalhos que estão relacionados com esta tese. O Capítulo 4 descreve os detalhes da solução proposta pelo mecanismo de migração. O Capítulo 5 apresenta os resultados de experimentos para avaliar a solução proposta neste trabalho. Por fim, o Capítulo 6 apresenta as considerações finais e os possíveis trabalhos futuros.

Capítulo 2

Fundamentos

Como discutido no capítulo anterior, as empresas estão cada vez mais interessadas na utilização apoio computacional em seus processos de negócio. Nesse contexto, Curtis *et al.* (1992) apontam quatro perspectivas a serem consideradas na representação computacional de um processo de negócio:

- **Perspectiva funcional/estrutural**: quais atividades devem ser realizadas e como elas são estruturadas?
- **Perspectiva comportamental**: qual a ordem de execução das atividades (fluxo de controle)?
- **Perspectiva informacional**: quais dados serão produzidos e consumidos pelas atividades e como é definido o fluxo de dados entre as atividades?
- **Perspectiva organizacional**: quem são os responsáveis pela execução das atividades?

Dentre essas perspectivas, as três primeiras apresentam os principais desafios. Ao longo dos anos, diversas estratégias têm sido propostas para apoiar computacionalmente as características concernentes às três primeiras perspectivas. Mais especificamente, esforços têm sido concentrados na modelagem e execução dos processos.

Este capítulo apresenta características das principais estratégias utilizadas nos PAISs para apoiar a modelagem e a execução de processos. Na Seção 2.1, serão apresentadas características gerais dos PAISs. Em seguida, na Seção 2.2, serão apresentadas as abordagens baseadas em modelos formais, tais como: Rede de Petri, Álgebra de Processos e Grafos. Posteriormente, serão discutidos os modelos declarativos na Seção 2.3. Por fim, uma visão geral da evolução dos modelos transacionais avançados é apresentada na Seção 2.4.

2.1 PAIS

Historicamente, os sistemas de informação empresariais têm buscado prover mais apoio sob a perspectiva do processo de negócio. Esses sistemas tendem cada vez mais a deixar de serem orientados a dados (*data-centric*) para serem cientes ou conhecedores do processo (*process-aware*) (Dumas *et al.*, 2005). Nesse sentido, as informações inerentes ao processo de negócio e seu contexto organizacional devem ser consideradas pelos sistemas. O termo genérico Sistemas de Informação Cientes do Processo (PAISs, do inglês *Process-Aware Information Systems*) tem sido adotado para

representar sistemas que gerencia e executa processos operacionais envolvendo pessoas, aplicações e/ou fontes de informação, com base nos modelos dos processos (Dumas *et al.*, 2005). Exemplos desses sistemas podem ser observados com a evolução dos Sistemas de Gerenciamento de *Workflow* e dos Sistemas de Gerenciamento de Processos de Negócio. Embora neste trabalho o termo PAISs seja utilizado para referenciar tais sistemas, a seguir são apresentadas características individuais dos WfMSs e BPMSs.

2.1.1 Sistemas de Gerenciamento de *Workflows*

A necessidade de representar, simular e controlar processos de negócio, juntamente com o avanço tecnológico da década de 90, culminaram no desenvolvimento dos WfMSs. Desde então, muita tecnologia tem sido desenvolvida e aplicada em diversas áreas. Mais detalhes sobre a visão histórica dos WfMS podem ser vistas em (Georgakopoulos *et al.*, 1995; Muehlen, 2004).

Curtis *et al.* (1992) e Jablonski e Bussler (1996) ressaltam a importância de um *workflow* contemplar as perspectivas funcionais, comportamentais, informacionais e organizacionais de um processo. Jablonski e Bussler (1996) ainda destacam que essas cinco perspectivas são obrigatórias para um *workflow* completo. No entanto, como mencionado anteriormente, somente será enfatizado características concernentes ao fluxo de controle e o fluxo de dados do *workflow*. Uma descrição mais profunda e detalhada sobre os WfMSs pode ser encontrada em (Aalst e van Hee, 2004; Muehlen, 2004).

Na literatura há uma grande variedade de terminologias e tecnologias de diferentes áreas da Ciência da Computação que definem os WfMSs e seus conceitos subjacentes. A compreensão dos principais fundamentos dos WfMSs foram unificados em grande parte por meio do modelo de referência proposto pela *Workflow Management Coalition* (WFMC). Um WfMS é caracterizado pela WFMC como um sistema que define, cria e gerencia a execução de *workflows* por meio de software que é capaz de interpretar a definição do processo, interagir com os participantes e, quando necessário, invocar aplicações externas (Coalition, 1995). Genericamente, essa definição indica que um WfMS possui três principais características: (i) a capacidade de criar esquemas ou modelos do *workflow*; (ii) a capacidade de criar instâncias dos esquemas; e (iii) a capacidade de executar as instâncias.

Um *esquema* ou *modelo de workflow* é definido pela WFMC como a representação de um processo de negócio de tal forma que permita a manipulação automatizada. Um processo pode ser descrito por uma rede de atividades e suas relações, critérios para indicar o início e o término do processo, e informações sobre atividades individuais, tais como os participantes envolvidos, aplicações externas, dados, etc. A partir de um esquema, uma instância é criada e inicia a execução do *workflow*. As instâncias são independentes e podem tomar distintos caminhos modelados no esquema. As instâncias também podem ser caracterizadas como *instâncias de curta duração* ou *instâncias de longa duração*. As instâncias de curta duração levam um pequeno tempo para serem executadas. Por outro lado, as instâncias de longa duração podem levar semanas, meses ou até anos para serem finalizadas.

Em outra iniciativa para melhorar a compreensão dos conceitos fundamentais que sustentam os processos de negócios, Aalst *et al.* (2003a) propõem padrões de *workflow* (*Workflow Pattern Initiative*) com objetivo de identificar os principais construtores inerentes à tecnologia de *workflow*. O objetivo original foi delinear os requisitos fundamentais que surgem durante a modelagem de

um processo de negócio, mantendo-os em uma base recorrente que pudessem ser reutilizados. No entanto, esses padrões são genéricos, ou seja, expressam características dos processos independente de linguagem ou tecnologia. A ausência de técnicas avançadas que possibilitem a verificação e análise do processo de negócio modelado traz sérios riscos a sua manutenção, principalmente, para processos complexos. Nesse sentido, as Seções 2.2 e 2.3 apresentam estratégias tradicionais utilizadas na especificação do processo de negócio a fim de prover formas de analisá-lo e verificá-lo.

2.1.2 Sistemas de Gerenciamento de Processos de Negócio

Um Sistema de Gerenciamento de Processo de Negócio (BPMS, do inglês *Business Process Management System*) pode ser considerado como uma extensão do WfMS clássico (Aalst *et al.* , 2003b). Os WfMSs têm como foco principal o apoio computacional na execução do processo. Nos últimos anos, diversos pesquisadores e profissionais argumentaram que esse foco é muito restrito. Como consequência, o termo Gerenciamento de Processo de Negócio (BPM, *Business Process Management*) tem sido definido. Aalst *et al.* (2003b) definem o BPM como o apoio a processos de negócio usando métodos, técnicas e software a fim de projetar, promulgar, controlar e analisar processos operacionais, envolvendo humanos, organizações, aplicações, documentos e outras fontes de informação.

Com enfoque somente nas características concernentes à modelagem e execução do processo, alguns padrões têm se destacado no cenário do BPM, tais como o *Business Process Model and Notation* (BPMN) (OMG, 2011) e o *Web Service Business Process Execution Language* (WS-BPEL) (OASIS, 2007). O padrão BPMN é uma notação gráfica para expressar processos de negócio. O padrão foi proposto pelo *Object Management Group* (OMG) com objetivo de fornecer uma notação facilmente compreensível por todos os envolvidos no negócio, incluindo os analistas que esboçam os primeiros rascunhos, os técnicos que são responsáveis pela implantação por meio da tecnologia e os profissionais da área de negócios que são responsáveis por gerenciar e monitorar o processo. Assim, a BPMN cria uma forma padronizada de expressar os processos de negócios utilizada em todas as etapas.

Uma das iniciativa mais difundidas nesta área nos últimos anos foi a WS-BPEL (OASIS, 2007), uma linguagem para composição de serviços *web* com uma ampla gama de recursos em comparação a outras iniciativas da área (Russell, 2007). Uma comparação dos seus recursos em relação às outras linguagens de composição de serviços *Web* pode ser encontrada em (Aalst *et al.*, 2005; Wohed *et al.*, 2003).

Embora a WS-BPEL seja centrada no controle do fluxo de execução das atividades, ela também incorpora conceitos de outras perspectivas, tais como os participantes envolvidos, a troca de mensagens e o tratamento de falhas e exceções. A especificação de um processo em WS-BPEL é composta de quatro seções principais, como descritas a seguir (OASIS, 2007).

- **Participantes:** Definição dos serviços participantes que interagem com o processo;
- **Variáveis:** Definição das variáveis de dados utilizados pelo processo;
- **Tratadores de falhas:** Definição das tarefas a serem executadas em respostas às falhas;
- **Lógica da composição:** Especificação do corpo principal do processo definindo a ordem na qual os participantes são invocados;

Os padrões BPMN e WS-BPEL evidenciam uma proximidade entre os PAISs e a Arquitetura Orientada a Serviços (SOA, do inglês *Service Oriented Architeture*) em que um processo pode agrupar os serviços *Web* disponíveis e atuar como um orquestrador. Por um lado, como característica favorável, esses padrões apresentam uma poderosa expressividade. Por outro lado, trazem sérios desafios quanto a análise e verificação formal dos processos modelados (Ferreira *et al.*, 2010b). Embora existam pesquisas sobre o mapeamento da WS-BPEL em arcabouços formais, a verificação dos modelos ainda é uma desvantagem diante do seu grande poder de expressividade.

Com a intenção de ilustrar o conceito de processo e definir um exemplo a ser utilizado ao decorrer desta proposta, a Figura 2.1 apresenta uma versão simplificada de um processo para o controle de reservas dos veículos oficiais da Universidade Tecnológica Federal do Paraná (UTFPR). Esses veículos são utilizados por servidores públicos em viagens de interesse da instituição. O processo da Figura 2.1 está de acordo com o padrão BPMN (*Business Process Model and Notation*).

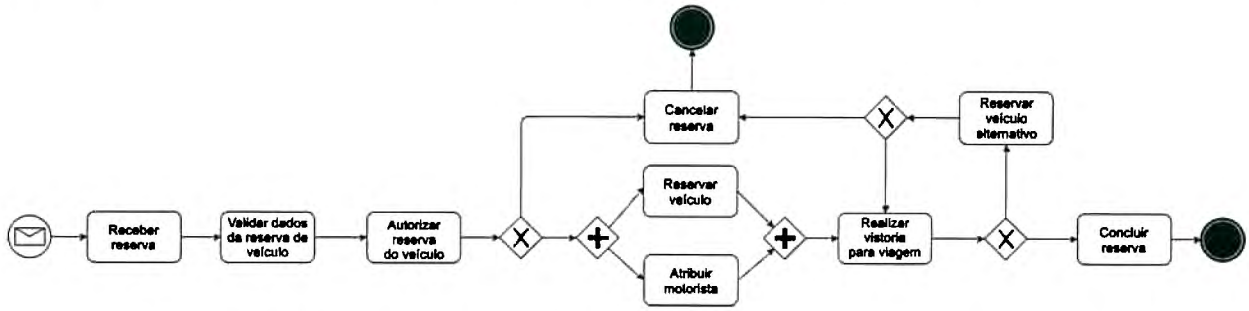


Figura 2.1: *Exemplo do processo de reserva de veículos representado em BPMN.*

Esse processo inicia quando um servidor solicita um veículo informando os dados para a reserva. Essa solicitação é validada pelo responsável da garagem e, então, é encaminhada ao diretor de área que deve autorizar a reserva. Logo após essa autorização, de forma paralela, um veículo é reservado e um motorista é atribuído para a viagem. No dia anterior à viagem, o motorista deve verificar se o veículo reservado está apto para o percurso. No caso do veículo apresentar algum problema ou avaria, ele deverá ser encaminhado ao serviço de manutenção. Nesse caso, o responsável pela garagem indica um veículo alternativo que também necessita ser vistoriado pelo motorista. Na ausência de veículos disponíveis, a reserva é cancelada. Por outro lado, se o carro está apto e vistoriado para a viagem, a reserva é concluída com sucesso.

A seguir são apresentados os principais arcabouços teóricos utilizados na especificação de um processo de negócio.

2.2 Modelos Formais

2.2.1 Rede de Petri

A Rede de Petri é uma técnica de modelagem e análise de processos com forte embasamento matemático. O conceito foi inicialmente proposto por Carl Adam Petri em sua tese de doutorado e, posteriormente, outras contribuições surgiram para complementar o formalismo inicial. Uma visão histórica sobre redes de Petri e uma vasta revisão bibliográfica podem ser encontradas em (Murata, 1989).

O forte embasamento matemático das redes de Petri permite a análise do sistema modelado.

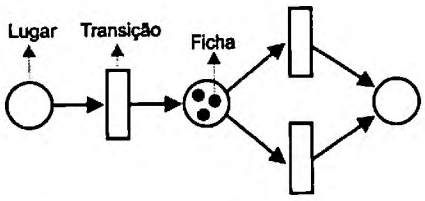


Figura 2.2: *Elementos básicos das redes de Petri.*

Dentre as análises possíveis, inclui-se verificações de propriedades inerentes aos sistemas concorrentes, tais como a relação de precedência entre eventos, sincronização e identificação de *deadlocks*.

Uma Rede de Petri clássica é um tipo particular de um grafo dirigido bipartido que possui dois tipos de vértices: *lugares* e *transições* (Aalst, 1998). A Figura 2.2 ilustra os componentes básicos das redes de Petri, na qual um lugar é representado por um círculo e uma transição por um retângulo. Além disso, um lugar pode ser interpretado como o componente passivo da rede, representando as variáveis de estado do sistema ou alguma pré ou pós-condição. A transição representa o componente ativo da rede que corresponde a alguma ação, tarefa ou evento do sistema. Há também dois tipos de arestas: as que ligam um lugar a uma transição e as que ligam uma transição a um lugar. Arestas de lugar para lugar ou de transição para transição são proibidas.

Baseado nesses tipos de arestas, define-se o conceito de *lugares de entrada* e *lugares de saída*. Um lugar p é um lugar de entrada da transição t se, e somente se, há uma aresta dirigida de p para t. De forma análoga, um lugar p é um lugar de saída da transição t se, e somente se, há uma aresta dirigida de t para p (Aalst e van Hee, 2004).

Os lugares podem conter fichas (ou marcas), representadas por círculos sólidos, que correspondem aos recursos disponíveis. Mais concretamente, a distribuição das fichas em uma rede é definida por uma marcação m. A marcação da rede de Petri representa seu estado em um determinado momento e a movimentação das fichas representa o comportamento dinâmico da rede. A movimentação das fichas é descrita pelas alterações dos estados da rede. A alteração de estado de uma rede de Petri ocorre por meio do disparo das transições (execução de ações), que segue as seguintes regras (Murata, 1989):

- Uma transição t é ativa (habilitada para execução) se cada lugar de entrada p de t contém pelo menos $w(p,t)$ fichas, em que $w(p,t)$ é o peso da aresta de p para t;
- Uma transição ativa pode ou não ser disparada;
- O disparo de uma transição ativa remove as $w(p,t)$ fichas de cada lugar de entrada p de t, e adiciona $w(t,p)$ fichas em cada lugar de saída p' de t, sendo $w(t,p)$ o peso da aresta de t para p.

Uma Rede de Petri PN é descrita formalmente pela quíntupla (Murata, 1989):

$$PN = (P, T, F, W, M_0) \tag{2.1}$$

em que:

- $P = \{p_1, p_2, \ldots, p_m\}$ é um conjunto finito de lugares;
- $T = \{t_1, t_2, \ldots, t_n\}$ é um conjunto finito de transições, tal que $P \cap T = \emptyset$ e $P \cup T \neq \emptyset$;

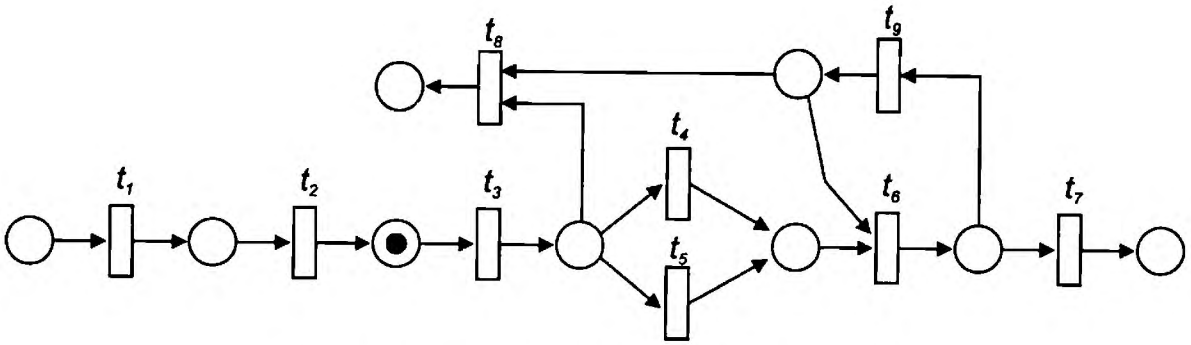


Figura 2.3: *Exemplo de uma Rede de Petri marcada.*

- $F \subseteq (P \times T) \cup (T \times P)$ é um conjunto de arestas direcionados também chamado de relações de fluxo;
- $W : F \to \mathbb{N}$ é a função que define o conjunto de pesos de arestas que relaciona a cada $f \in F$ um $n \in \mathbb{N}$, representando quantas fichas são consumidas de um lugar $s \in S$ por uma transição $t \in T$, ou, alternativamente, quantas fichas são produzidas por uma transição $t \in T$ e colocadas em um lugar $s \in S$;
- $M_0 : P \to \mathbb{N}$ é a marcação inicial no qual existem $n \in \mathbb{N}$ fichas para cada $s \in S$.

Uma rede marcada é definida pela tupla $\langle PN, m \rangle$, sendo PN a estrutura da rede e m sua marcação. A Figura 2.3 ilustra uma versão da Rede de Petri do processo de reserva descrito na Figura 2.1. Nessa rede, cada transição corresponde a uma atividade do processo, como identificado a seguir:

- t_1: Receber Reserva;
- t_2: Validar Dados da Reserva de Veículo;
- t_3: Autorizar Reserva do Veículo;
- t_4: Reservar Veículo;
- t_5: Atribuir Motorista;
- t_6: Realizar Vistoria para Viagem;
- t_7: Concluir Reserva;
- t_8: Cancelar Reserva;
- t_9: Reservar Veículo Alternativo.

A distribuição das fichas retrata o estado da rede. Nesse exemplo, a ficha no lugar de saída da transição t_2 indica que os dados da reserva já foram validados. Além disso, a transição responsável por autorizar a reserva do veículo está apta para ser executada. A seguir, são apresentadas as características do modelo formal baseado em grafos.

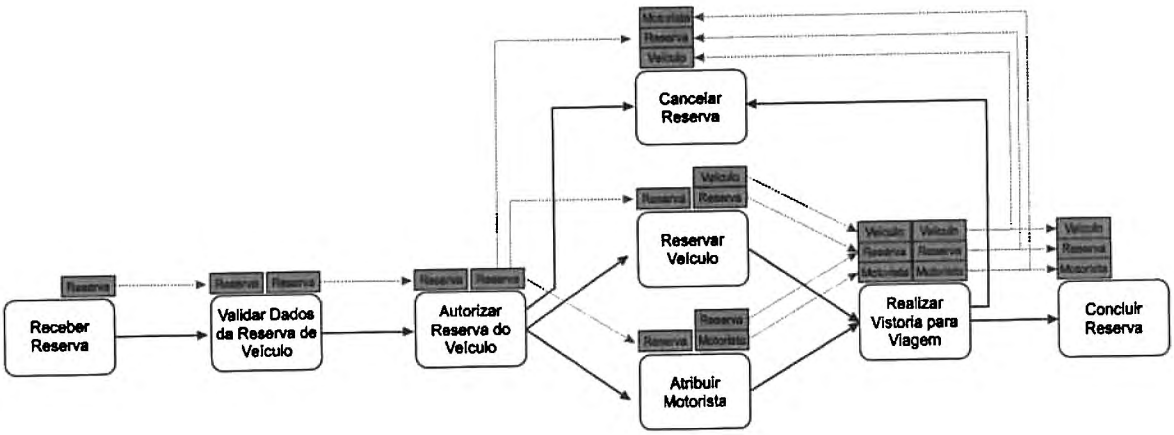


Figura 2.4: *Processo de reserva ilustrado por uma abordagem baseado em grafo.*

2.2.2 Modelos baseados em Grafos

Esta seção introduz, resumidamente, conceitos genéricos das abordagens que modelam *workflows* usando grafos. Uma descrição mais detalhada sobre as abordagens baseadas em grafos pode ser encontrada em (Weske, 2007). Essas abordagens normalmente incluem a representação explícita das dependências de dados entre as atividades e técnicas para descrever a semântica da execução dos processos de negócio. Para descrever os conceitos desses modelos, o exemplo de um processo de reserva de veículos, ilustrado na Figura 2.4, será considerado.

As atividades são representadas pelos vértices do grafo, o fluxo de controle é representado pelas arestas sólidas e o fluxo de dados pelas arestas pontilhadas. Por exemplo, as atividades *Receber Reserva* e *Validar Dados da Reserva de Veículo* são conectadas por uma aresta sólida que representa o fluxo de controle.

Cada atividade possui paramêtros de entrada e saída de dados. Sempre que um paramêtro de saída de uma atividade A_1 é usado como um paramêtro de entrada de outra atividade A_2, uma aresta $\langle A_1, A_2 \rangle$ representando o fluxo de dados é adicionada ao grafo. Por exemplo, entre essas atividades *Receber Reserva* e *Validar Dados da Reserva do Veículo*, há uma dependência de dados, representada pela aresta pontilhada, na qual o parâmetro de saída da atividade *Receber Reserva* é usado como paramêtro de entrada da atividade *Validar Dados da Reserva do Veículo*.

A semântica da execução é baseada em marcações nas arestas do fluxo de controle. Há dois tipos de marcações: verdadeiro e falso. Cada aresta tem uma condição associada. Essa condição é avaliada após o término da atividade origem da aresta. Se a condição for satisfeita, então a aresta é marcada como verdadeiro. Caso contrário, a aresta recebe a marcação falso. Quando todas as arestas incidentes em uma atividade são marcadas, a condição de início da atividade é avaliada.

A condição de início é determinante para tratar as diferentes estratégias de divisão, tais como paralelas (AND), escolha exclusiva (XOR), ou escolha múltipla (OR). Abordagens que não permitem a dupla marcação têm problemas ao tratar a junção quando a divisão OR é considerada. Com a dupla marcação, todos os caminhos são sinalizados, inclusive aqueles caminhos que não foram tomados durante a execução. Entretanto, essa técnica ainda não é suficiente para tratar ciclos no grafo, isto é, estruturas de repetição. Devido a esse motivo, a Figura 2.4 não ilustra a atividade *Reservar Veículo Alternativo* representada em BPMN na Figura 2.1. Em (Reichert e Dadam, 1997), há uma alternativa para considerar estruturas de repetição, descrita brevemente no Capítulo 3.

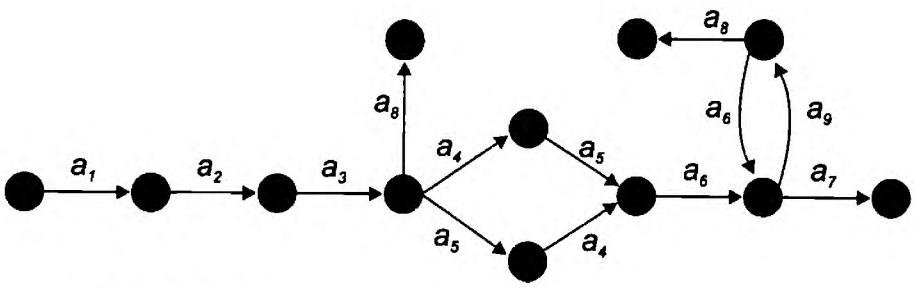


Figura 2.5: *Grafo do processo de reserva de acordo com o LTS.*

Além das marcações nas arestas de controle, a condição de início de uma atividade avalia os paramêtros de entrada de dados que devem ser recebidos das atividades anteriores. Se a condição for satisfeita, a atividade é inicializada. Caso contrário, a atividade é ignorada e as arestas que partem dela são marcadas com falso.

2.2.3 Álgebra de Processo

Esta seção aborda resumidamente os conceitos introdutórios da Álgebra de Processo e dos Sistemas de Transições Rotuladas. Uma descrição mais detalhada sobre tais conceitos pode ser encontrada em (Fokkink, 2000).

Uma maneira de representar comportamento dos processos de negócio é a utilização de um Sistema de Transições Rotuladas (LTS, do inglês *Labelled Transition System*). O LTS propõe uma representação do comportamento do processo de negócio por meio de um grafo no qual os vértices representam os estados do processo e as arestas rotuladas representam as transições de estado. Nesse grafo, um estado é selecionado para ser o estado raiz, isto é, o estado inicial do processo. Se o LTS contém uma aresta $s \xrightarrow{a} s'$, então o grafo pode evoluir do estado s para o estado s' ao executar a ação a. A Figura 2.5 apresenta o processo de reserva de veículos, inicialmente apresentado na Seção 2.1, na forma de um LTS. As ações são definidas pelo conjunto $A = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9\}$. Cada ação representa uma atividade do processo, como descrito a seguir:

- a_1. Receber Reserva;
- a_2. Validar Dados da Reserva de Veículo;
- a_3. Autorizar Reserva do Veículo;
- a_4. Reservar Veículo;
- a_5. Atribuir Motorista;
- a_6. Realizar Vistoria para Viagem;
- a_7. Concluir Reserva;
- a_8. Cancelar Reserva;
- a_9. Reservar Veículo Alternativo.

A Álgebra de Processos é uma representação de grafos de processos com propósito matemático. Ela apresenta enfoque na especificação e manipulação dos termos do processo por meio de uma coleção de operadores. Com os operadores básicos dessa coleção é possível construir processos finitos. Há também operadores de comunicação para expressar a concorrência e a recursividade que captura o comportamento infinito. A Álgebra de Processo constitui de uma representação formal dos processos e dados, com ênfase em processos que são executados concorrentemente. Essa representação pode ser utilizada para detectar propriedades indesejáveis e derivar, formalmente, propriedades desejáveis de uma especificação do sistema.

Uma estrutura básica para a álgebra de processo é constituída pelos seguintes operadores:

- Um conjunto finito e não vazio A de ações atômicas que representam um comportamento indivisível. O predicado $\xrightarrow{a} \surd$ representa a terminação com sucesso após a execução da ação a;
- Um operador binário $+$, referenciado como composição alternativa. Se os termos t_1 e t_2 representam os processos p_1 e p_2, respectivamente, então o termo $t_1 + t_2$ representa a execução de p_1 ou p_2.
- Um operador binário $\cdot$, chamado de composição sequencial. Se os termos t_1 e t_2 representam os processos p_1 e p_2, respectivamente, então o termo $t_1 \cdot t_2$ representa a execução de p_1 seguida por p_2.

Cada processo finito pode ser representado por um termo que é construído por ações atômicas do conjunto A e pelos operadores binários $+$ e $\cdot$. Esse termo é chamado de termo básico do processo e a coleção de todos os termos é chamado de Álgebra de Processo Básica (BPA, do inglês *Basic Process Algebra*.

A semântica operacional estrutural define uma coleção de regras de transição. Uma transição $t \xrightarrow{a} t'$ expressa que o termo t pode evoluir para o termo t' pela execução da ação a e o predicado $t \xrightarrow{a} \surd$ expressa que o termo t pode terminar com sucesso ao executar a ação a. A Tabela 2.1 apresenta a especificação do sistema de transição que constitui da semântica operacional estrutural da BPA. As variáveis x, x', y e y' nas regras de transição referem-se a coleção de termos básicos do processo, e v refere-se ao conjunto A de ações atômicas.

Tabela 2.1: *Regras de Transição do BPA*

$$\frac{}{v \xrightarrow{v} \surd}$$

$$\frac{x \xrightarrow{v} \surd}{x + y \xrightarrow{v} \surd} \qquad \frac{x \xrightarrow{v} x'}{x + y \xrightarrow{v} x'} \qquad \frac{y \xrightarrow{v} \surd}{x + y \xrightarrow{v} \surd} \qquad \frac{y \xrightarrow{v} y'}{x + y \xrightarrow{v} y'}$$

$$\frac{x \xrightarrow{v} \surd}{x \cdot y \xrightarrow{v} y} \qquad \frac{x \xrightarrow{v} x'}{x \cdot y \xrightarrow{v} x'y}$$

A especificação do sistema de transição do BPA fornece:

- A primeira regra determina que cada ação atômica v pode terminar com sucesso pela execução dela própria;
- As quatro regras seguintes expressam que o termo $t + t'$ executa t ou t'; e
- A duas últimas regras expressam que $t \cdot t'$ executa t até sua execução completa e, então, executa t'.

As regras de transição provam, por exemplo, que a transição $((a + b) \cdot c) \cdot d \xrightarrow{b} c \cdot d$ é correta. Esse exemplo pode ser visto na Tabela 2.2. Do lado direito, as regras de transição são aplicadas em passos consecutivos. As substituições são mostradas do lado esquerdo.

Tabela 2.2: *Exemplo de aplicação das regras de transição do BPA*

1)	$b \xrightarrow{b} \surd$	$(\dfrac{}{v \xrightarrow{v} \surd},$	$v := b)$
2)	$a + b \xrightarrow{b} \surd$	$(\dfrac{y \xrightarrow{v} \surd}{x + y \xrightarrow{v} \surd},$	$v := b,\ x := a,\ y := b)$
3)	$(a + b) \cdot c \xrightarrow{b} c$	$(\dfrac{x \xrightarrow{v} \surd}{x \cdot y \xrightarrow{v} y},$	$v := b,\ x := a + b,\ y := c)$
4)	$((a + b) \cdot c) \cdot d \xrightarrow{b} c \cdot d$	$(\dfrac{x \xrightarrow{v} x'}{x \cdot y \xrightarrow{v} x'y},$	$v := b,\ x := (a + b) \cdot c,\ x' := c,\ y := d)$

A avaliação da igualdade de dois processos por meio de grafos é um árduo trabalho. A Álgebra de Processo propõem axiomas que podem ser usados para avaliar a igualdade entre dois termos de processo. Na Tabela 2.3 são exibidos os axiomas referentes à álgebra de processos básica. As variáveis x, y e z são termos básicos de processo.

Tabela 2.3: *Axiomas para BPA*

A1	$x + y = y + x$
A2	$(x + y) + z = x + (y + z)$
A3	$x + x = x$
A4	$(x + y) \cdot z = x \cdot z + y \cdot z$
A5	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$

Suponha os termos $t_1 : (a + a) \cdot b$ e $t_2 : a \cdot b + a \cdot (b + b)$. Ao aplicar o axioma A3 em ambos termos, é possível obter $t_1 : a \cdot b$ e $t_2 : a \cdot b + a \cdot b$. Se a substituição for novamente aplicada para o termo t_2, obtém-se que $t_2 : a \cdot b$. Dessa forma, é possível demonstrar que os termos t_1 e t_2 são equivalentes.

Até agora, apenas os operadores de uma estrutura básica da Álgebra de Processos foram discutidos. A maioria das álgebras de processos contém outros operadores e, para ilustrar como modelar

o processo de reserva exibido na Seção 2.1, é necessário descrever também o operador binário $\|$ chamado de entrelaçamento, introduzido por Milner (Milner, 1982). O termo $s \| t$ indica que os termos s e t serão executados em paralelo. As regras de transição da Tabela 2.4 formalizam o comportamento do operador entrelaçamento.

Tabela 2.4: *Regras de Transição para o operador entrelaçamento*

$\dfrac{x \xrightarrow{v} \surd}{x \,\|\, y \xrightarrow{v} y}$	$\dfrac{x \xrightarrow{v} x'}{x \,\|\, y \xrightarrow{v} x' \,\|\, y}$
$\dfrac{y \xrightarrow{v} \surd}{x \,\|\, y \xrightarrow{v} x}$	$\dfrac{y \xrightarrow{v} y'}{x \,\|\, y \xrightarrow{v} x \,\|\, y'}$

Uma vez que exemplo da Figura 2.1 apresenta uma estrutura de repetição, sua representação em álgebra de processo é definido por uma expressão recursiva. Considerando isso, a expressão algébrica para o processo de reserva PR, ilustrado inicialmente na Figura 2.1, pode ser definida como:

$$PR := a_1 \cdot a_2 \cdot a_3 \cdot (((a_4 \,\|\, a_5) \cdot a_6 \cdot R) + a_8)$$
$$R := a_7 + a_9 \cdot a_8 + a_9 \cdot a_6 \cdot R$$

2.3 Modelos Declarativos

As abordagens formais apresentam uma estrutura rígida que dificulta a alteração dos processos de negócio comprometendo a evolução e a adaptação de instância em execução. Por esse motivo, os modelos declarativos têm se popularizado, uma vez que apresentam uma forma promissora e flexível de apoiar processos de negócio com características dinâmicas. Os modelos declarativos utilizam, basicamente, duas formas de especificar o processo de negócio: regras ou restrições.

Os modelos declarativos baseados em regras representam a estrutura lógica e as dependências de dados entre as execuções das atividades por meio de regras. O modelo deve ser capaz de examinar condições de controle e de dados optando pela ordenação mais adequada das atividades. Desse modo, o caminho de execução tomado por uma instância é dinâmico, não sendo pré-estabelecido como apresentado nas abordagens anteriores. Os modelos declarativos baseados em restrições assumem que todas as alternativas de execução são possíveis, exceto quando uma restrição seja explicitamente especificada. O modelo deve ser capaz de identificar se a sequência de execução das atividades não viola o conjunto de restrições especificado.

Em resumo, há modelos declarativos que especificam regras para expressar o comportamento esperado do processo, enquanto há modelos declarativos que especificam restrições para expressar o comportamento indesejado do processo. A seguir, na Seção 2.3.1, serão discutidas as características dos modelos declarativos baseados em regras. Na Seção 2.3.2 serão apresentados os principais conceitos dos modelos declarativos baseados em restrições.

2.3.1 Modelos Declarativos baseados em Regras

Um dos conceitos mais significativos utilizado nesses modelos é o de Evento-Condição-Ação (ECA). As regras expressadas por um ECA podem ser utilizadas para orientar a execução das

atividades em um processo de negócio. Historicamente, a comunidade de banco de dados foi uma das pioneiras na aplicação das regras ECA, na proposição dos sistemas de banco de dados ativos (Dayal, 1994). Nos sistemas de banco de dados ativos, as regras ECA são usadas para reagir as alterações em um banco de dados (tais como: inserção, remoção, ou modificação de dados). No contexto de processo de negócio, as regras ECA podem ser utilizadas para reagir aos diversos tipos de eventos inerentes ao processo (Casati *et al.*, 2000).

Uma regra ECA pode formalmente ser definida pela tripla (Lu e Sadiq, 2007):

$$T = \langle E, C, A \rangle$$

no qual:

- E é o conjunto de eventos que podem ativar uma regra. Um evento especifica quando uma regra deve ser avaliada, que indica a transformação de um estado de execução para outro. Um evento pode ser uma simples mudança do estado de execução de uma atividade (por exemplo, término da execução da atividade Autorizar reserva do veículo), ou um evento complexo de um processo de negócio (por exemplo, 1 dia antes da realização da viagem).
- C é o conjunto de condições que devem ser avaliadas antes da ativação da ação. Normalmente, referem-se a verificação da disponibilidade de algum dado para execução do processo (por exemplo, veículo disponível para viagem);
- A representa o conjunto de ações executadas. Normalmente, uma ação refere-se a execução de uma certa atividade. Entretanto, pode também referir-se ao disparo de uma outra regra ECA atuando como um novo evento.

A especificação de processos de negócio em uma abordagem baseada em regras ECA, apresentam as seguintes vantagens (Bry *et al.*, 2006):

- Requisitos de negócio são frequentemente especificados na forma de regras expressas em uma linguagem formal ou em uma linguagem natural. Por exemplo, a solicitação do produto (evento) será cancelada (ação) se o estoque for insuficiente (condição) para atender o cliente;
- Regras ECA são naturalmente flexíveis, pois atendem as necessidades de alteração dos processos de negócio;
- Regras ECA podem facilmente especificar a manipulação de erros e de situações excepcionais por meio de eventos especiais. Assim, uma exceção é tratada da mesma forma que em situações normais;
- Em abordagens tradicionais, atividades são inicializadas como uma reação a finalização de outras atividades e reações a partir de estados intermediários das atividades não são considerados. Ao invés disso, regras ECA com ênfase em eventos são mais flexíveis para especificar o fluxo de controle.

Nesta proposta, as regras ECA serão novamente mencionadas na Seção 2.4.2, ao discutir a abordagens WED-*flow*. Na próxima seção são discutidos os modelos declarativos baseados em restrições.

2.3.2 Modelos Declarativos baseados em Restrições

Esta seção introduz conceitos genéricos das abordagens baseadas em restrições. Atualmente, essas abordagens têm se popularizado, uma vez que apresentam uma forma promissora e flexível de apoiar processos de negócio dinâmicos. Ao contrário dos modelos procedimentais, os modelos baseados em restrições enfatizam que todas as alternativas de execução são permitidas, exceto quando são explicitamente proibidas.

A Figura 2.6 ilustra um interessante comparativo entre abordagens procedimentais e as abordagens baseadas em restrições. Qualquer modelo de *workflow* deve ser capaz apoiar o comportamento desejado de um processo, enquanto proíbe o comportamento indesejado. As abordagens procedimentais garantem bem o comportamento desejado. Entretanto, elas sacrificam a flexibilidade, uma vez que são propensas à especificações excessivas[1]. Ao invés disso, abordagens baseadas em restrições concentram-se em elicitar comportamentos obrigatórios preservando a flexibilidade ao evitar restrições adicionais e desnecessárias (Montali, 2010).

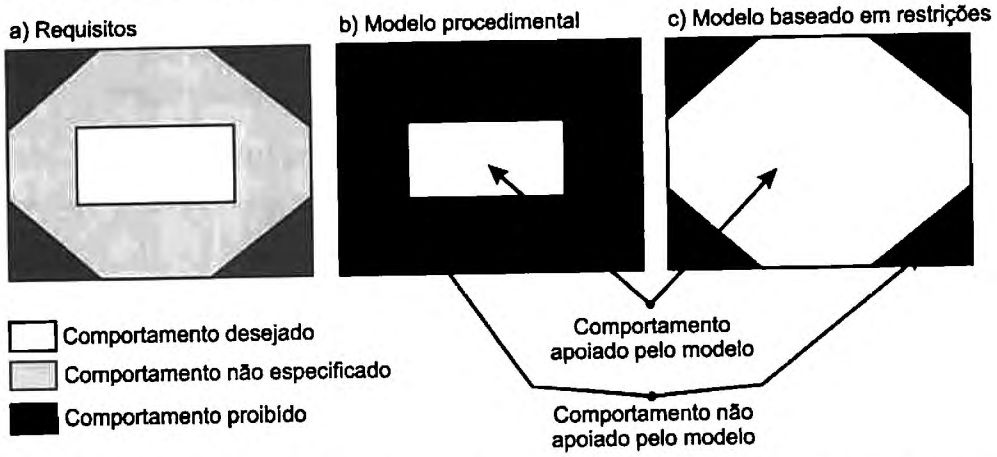


Figura 2.6: *Modelos de processos baseados em restrições e procedimentais. Adaptado de Montali (2010)*

Um esquema de *workflow* S baseado em restrições pode ser formalmente definido como:

$$S = \langle A, C \rangle$$

em que A consiste de um conjunto finito de atividades e C representa um conjunto finito de restrições que proíbem execuções indesejadas das atividades. Além disso, o conjunto de restrições $C = C_M \cup C_O$ pode ser dividido em dois conjuntos distintos. C_M representa restrições obrigatórias que devem ser respeitadas, enquanto C_O representa restrições opcionais que deveriam ser respeitadas, isto é, são recomendações.

A semântica operacional de uma abordagem baseada em restrições é definida por um conjunto de restrições que deve ser garantida durante a execução de uma instância. A seguir, será apresentado diferentes categorias de restrições que determinam o fluxo de controle das atividades (Reichert e Weber, 2012).

- **Restrições de Existência:** especificam a frequência que as atividades devem ser executadas para uma instância do processo. Entre as restrições dessa categoria, destacam-se:

  - `existence(a,n)`: expressa que a atividade a deve ser executada pelo menos n vezes;

[1] do inglês *over-specification*

- `at_most(a,n)`: expressa que a atividade a pode ser executada no máximo n vezes;
- `exactly(a,n)`: especifica que uma atividade a deve ocorrer exatamente n vezes;
- `init(a)`: expressa que a atividade a necessita ser a primeira executada em qualquer instância.

- **Restrições de Escolha:** permitem especificar escolhas de execuções em um conjunto de atividades. Por exemplo:
  - `choice(m − of − n, {a1, ..., an})`, $m \leq n$ especifica que pelo menos m atividades distintas do conjunto $a_1, \ldots, a_n$ devem ser executadas;
  - `exact_choice(m − of − n, {a1, ..., an})`, $m \leq n$, especifica que exatamente m atividades distintas do conjunto $\{a_1, \ldots, a_n\}$ devem ser executadas.

- **Restrições de Relação:** impõem restrições na relação entre as atividades a e b. Nessa categoria, destacam-se:
  - `response(a,b)` determina que se a atividade a é executada, a atividade b deve ser executada posteriormente (mas não necessariamente de forma consecutiva);
  - `precedence(a,b)` requer que a atividade b seja precedida pela atividade a (mas não necessariamente diretamente precedida);
  - `succession(a,b)` é a combinação das duas anteriores e requer que se a atividade a é executado, então a atividade b deve ser executado posteriormente. Além disso, a execução da atividade b deve ser precedida da atividade a.

- **Restrições de Negação:** definem restrições de negação entre as atividades. Por exemplo:
  - `neg_coexistence(a,b)` proíbe que as atividades a e b executem na mesma instância; isto é, a e b são mutualmente exclusivas;
  - `neg_response(a,b)` proíbe que a atividade b execute após a atividade a.

- **Restrições de Ramificação:** especificam restrições de relação e de negação envolvendo mais do que duas atividades. Exemplos de restrições de ramificação são:
  - `response(a, {b1, ..., bn})` expressa que se a é executada, pelo menos uma atividade do conjunto $\{b_1, \ldots, b_n\}$ deve ser posteriormente executada.
  - `succession({a1, ..., am}, {b1, ..., bn})` expressa que se uma atividade pertencente ao conjunto $\{a_1, \ldots, a_m\}$ é executada, então pelo menos uma atividade pertencente ao conjunto $\{b_1, \ldots, b_n\}$ deve ser executada. Além disso, para executar uma atividade do conjunto $\{b_1, \ldots, b_n\}$ pelo menos uma atividade do conjunto $\{a_1, \ldots, a_m\}$ deve ser executada.

A Figura 2.7 exemplifica um simples modelo baseado em restrições representado por quatro atividades distintas a, b, c e d. Além disso, duas restrições obrigatórias são consideradas: $c_1 =$ `init(a)` e $c_2 =$ `choice(2 − of − 3, {b, c, d})`. A primeira restrição impõe que todas instâncias devem ser

iniciadas pela atividade a e a segunda restrição determina que no mínimo duas distintas atividades do conjunto $\{b, c, d\}$ devem ser executadas. Suponha o histórico de execução de quatro diferentes instâncias: $\sigma_1 = \langle a, a, b, c, d \rangle$, $\sigma_2 = \langle a, b, b, c \rangle$, $\sigma_3 = \langle a, b \rangle$ e $\sigma_4 = \langle b \rangle$. As instâncias σ_1 e σ_2 são consistentes e respeitam as duas restrições c_1 e c_2. No entanto, as instâncias σ_3 e σ_4 não são consistentes e não podem ser instâncias válidas para esse simples exemplo. A instância σ_3 não respeita c_2, uma vez que apenas a atividade b foi executada. A instância σ_4 claramente viola a restrição c_1 uma vez que inicia com a atividade b.

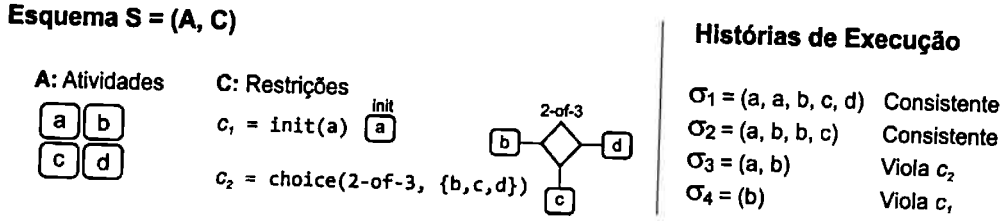


Figura 2.7: *Exemplo de um simples processo modelado usando restrições*

Uma instância em execução pode estar em um dos seguintes estados: *satisfeita*, *temporariamente violada* ou *violada*. Uma instância I está no estado satisfeita, se o seu histórico de execução parcial satisfaz todas as restrições do esquema S. Uma instância I está no estado de temporariamente violada se o seu histórico de execução parcial não satisfaz todas as restrições do esquema S, mas existe um sufixo que pode ser adicionado ao histórico tal que as restrições do esquema S sejam satisfeitas. Finalmente, uma instância está no estado de violada, se o histórico de execução parcial viola as restrições do esquema S e não há sufixo que possa ser adicionado ao histórico para satisfazer as restrições (Reichert e Weber, 2012). Para exemplificar os estados de execução, considere novamente as quatro instâncias da Figura 2.7. As instâncias σ_1 e σ_2 estão no estado de satisfeitas pois atendem ambas restrições do modelo. A instância σ_3 está no estado de temporariamente violada, uma vez que existe um sufixo válido que satisfaça a restrição c_2. Por último, a instância σ_4 está no estado de violada, uma vez que nenhum sufixo pode satisfazer a restrição c_1.

O mecanismo de execução de um modelo baseado em restrições deve evitar que instâncias violem as restrições do modelo. Esse mecanismo exibe ao usuário uma lista de atividades habilitadas para execução. Dessa forma, usuários podem apenas executar atividades que não conduzam a instância para o estado de violada (Reichert e Weber, 2012). A Figura 2.8 exibe a lista de atividades habilitadas considerando o mesmo esquema da Figura 2.7. Note que inicialmente a apenas a atividade a é habilitada para a execução. Logo após sua execução, o estado da instância é alterado para temporariamente violada e todas as outras atividades também são habilitadas para execução. O estado da instância é alterado apenas quando a restrição c_2 for satisfeita.

Em *Declare* (Aalst *et al.*, 2009), o mecanismo de execução descrito acima é alcançado ao mapear as restrições para uma Lógica Temporal Linear (LTL). Baseado em uma fórmula LTL que representa todas as restrições do modelo, é possível criar um autômato finito que pode ser usado para (i) determinar as atividades habilitadas para execução, (ii) dirigir a execução e (iii) determinar o estado de execução de cada instância. Essa estratégia é custosa, pois a construção do autômato finito é exponencial ao tamanho da fórmula LTL. Como consequência, modelos que apresentem um grande número de restrições apresentam problemas de desempenho.

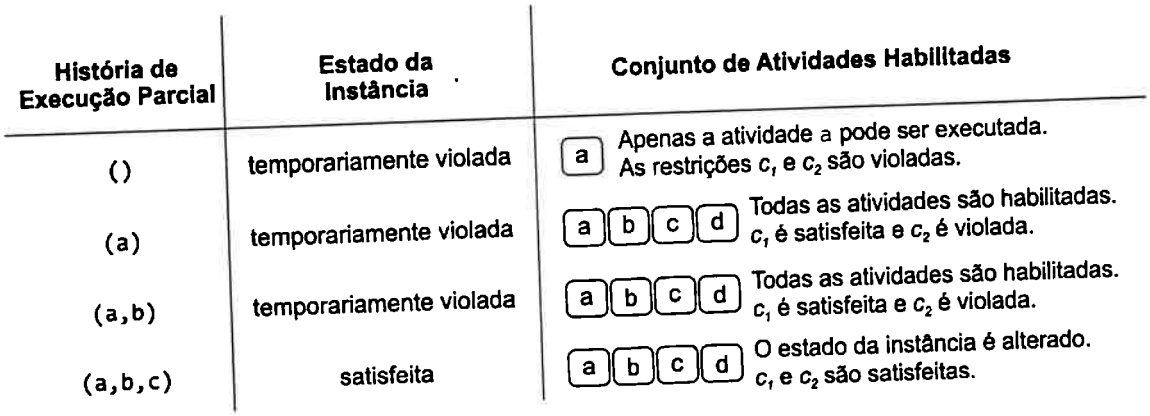


Figura 2.8: *Mecanismo de execução de uma instância*

2.4 Modelos Transacionais Avançados

2.4.1 Esferas de Controle

Ao longo dos anos, diversos autores têm enfatizado a importância de reavaliar as premissas e as propriedades do modelo transacional clássico, diante dos novos requisitos das aplicações modernas (Elmagarmid, 1992). Embora o modelo transacional clássico represente um marco na evolução das aplicações tradicionais, ele apresenta aplicabilidade limitada diante das características requeridas pelas aplicações modernas. Essas aplicações são normalmente complexas, executam em longos intervalos de tempo, possuem múltiplos passos executados em base de dados heterogêneas, invocam aplicações externas, apresentam atividades interativas, entre outras características. No contexto do Gerenciamento de Transações, as transações que reúnem essas características são conhecidas na literatura como transações longas ou transações de longa duração (García-Molina e Salem, 1987). As transações longas apresentam grandes desafios ao modelo transacional clássico. Entre os principais desafios estão:

- **Duração da execução:** Devido ao longo tempo de execução da transação, ela é mais suscetível a falhas. No modelo clássico, quando uma transação falha, a propriedade de atomicidade garante o desfazimento das ações realizadas ao restaurar o estado do banco de dados. Essa estratégia é factível para transações de curta duração. No entanto, a restauração do banco de dados não é aceitável para transações longas, pois muito trabalho será perdido se a transação abortar;
- **Acesso a itens de dados concorrentes:** Devido à propriedade de isolamento, a transação não pode liberar os itens de dados bloqueados para utilização de outras transações enquanto ela não finalizar (confirmação final). Assim, a propriedade de isolamento bem como os protocolos de bloqueio tradicionais, podem acarretar em inaceitáveis para as transações que aguardam pelo item de dado bloqueado.

Esses desafios confrontam as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) que fundamentam o modelo transacional clássico. Mais especificamente, a propriedade de atomicidade e isolamento são as que apresentam restrições mais significativas e precisam ser reavaliadas no contexto das transações longas. Kot *et al.* (2010) ratifica essa necessidade chamando a atenção da comunidade de Banco de Dados sobre a necessidade de transcender a propriedade de isolamento, em direção à abstrações que proporcionem novas características. Para auxiliar nessa

discussão, é necessário recuperar o conceito amplo e genérico que deu origem ao modelo transacional clássico: o conceito das Esferas de Controle proposto por Davies e Björk (Björk, 1973; Davies Jr, 1973, 1978). A Esfera de Controle é um conceito genérico para controlar execuções concorrentes em ambientes multiusuários distribuídos, bem como controlar os efeitos colaterais de possíveis falhas. O conceito é baseado em duas premissas básicas: (i) conhecer os efeitos das operações arbitrárias enquanto existir algum cenário em que elas possam ser revogadas; e (ii) monitorar as dependências estabelecidas durante a execução de forma que seja capaz de recuperar o histórico de execução em caso de falha. Essas premissas permitem que os dados produzidos por um processo atômico transcenda o limite imposto pelo seu término.

Embora o conceito das Esferas de Controle tenha sido fundamental ao evidenciar diretrizes no gerenciamento de execuções concorrentes, ele não teve o mesmo êxito sob a perspectiva prática. As Esferas de Controle não foram formalizadas ou totalmente desenvolvidas devido à sua generalidade. Nesse sentido, o modelo transacional clássico pode ser considerado uma Esfera de Controle que restringe significativamente o controle originalmente proposto pelo conceito. Essa restrição é determinada pelas propriedades ACID. Sob essa perspectiva, os modelos transacionais avançados têm apresentado diferentes esferas de controle a fim de minimizar as restrições impostas pelas propriedades ACID no contexto das aplicações modernas. De maneira geral, essas soluções ampliam a aplicabilidade do modelo clássico ao flexibilizar algumas das propriedades ACID para apoiar a execução de transações de longa duração. Dessa forma, as estratégias propostas pelos modelos transacionais avançados serão apresentadas sob a perspectiva de três classes restritas das Esferas de Controle. A primeira classe representa as abordagens que estendem a seriação realizando pequenos ajustes na proposta original. A segunda classe apresenta estratégias que flexibilizam a seriação ao permitir a exposição dos resultados parciais durante a execução das transações. A terceira classe expõe abordagens que substituem a seriação por um critério de correção alternativo.

As abordagens da primeira classe propõem estratégias baseadas em variações do protocolo de bloqueio. Essas variações são especialmente projetadas para flexibilizar a propriedade de isolamento enquanto mantém agendas seriáveis. Três abordagens destacam-se nessa classe. A primeira é o Modelo Transacional Hierárquico (*Nested Transaction Model*) (Moss, 1985) que propõe um controle de concorrência baseado na extensão do Protocolo de Bloqueio Bifásico (*Two-phase Locking Protocol*). Essa abordagem flexibiliza a propriedade de isolamento ao modelar uma transação por meio de uma estrutura hierárquica. Uma transação aninhada é uma árvore na qual a transação pai é a raiz e as sub-transações filhas são as folhas ou nós intermediários. Por meio dessa hierarquia, o gerenciador de bloqueio permite que a transação pai assuma os bloqueios das sub-transações filhas quando elas finalizam. Uma sub-transação, ao efetuar seu ponto de confirmação, somente produz efeito após seu pai também ter o seu ponto de confirmação efetuado. A falha de uma sub-transação filha não implica na falha imediata da transação pai. Neste caso, o pai pode ressubmeter a execução da filha ou executar uma sub-transação alternativa. Essa abordagem introduz um flexibilização sutil na propriedade de isolamento ao permitir que resultados intermediários sejam compartilhados entre as sub-transações. Essa flexibilização somente é possível porque a transação pai monitora as dependências criadas durante a execução das sub-transações e consegue reverter os efeitos caso seja necessário. No entanto, o controle de concorrência entre as transações que encontram-se na raiz da árvore é idêntico ao modelo transacional clássico.

A segunda abordagem é o Bloqueio Altruísta (*Altruistic Locking*) (Salem *et al.*, 1994), que es-

tende a seriação ao permitir que transações liberem seus bloqueios antes do ponto de confirmação. Quando uma transação tem o conhecimento que não utilizará mais o item de dado bloqueado, o gerenciador de bloqueio permite que ela doe seu bloqueio a outras transações. Como consequência, a transação que recebe a doação, torna-se dependente da transação doadora. Por exemplo, uma transação t_j que recebe a doação do bloqueio de um item de dado x da transação t_i tem acesso a resultados intermediários de x que somente serão efetivados quando t_i alcançar o ponto de confirmação. Portanto, t_j torna-se dependente de t_i. No caso de falha, o aborto de t_i implica no aborto de t_j.

Na terceira abordagem, o grupo de pesquisa Big Red Data da Cornell University propõe as Transações Emaranhadas (*Entangled Transactions*), como uma forma concreta para coordenar transações por meio da flexibilização do isolamento (Gupta *et al.*, 2011b). As transações emaranhadas estendem as transações clássicas ao incluir pontos de sincronizações chamados de Consultas Emaranhadas (*Entangled Queries*) (Gupta *et al.*, 2011a). As consultas emaranhadas permitem uma comunicação controlada entre as transações para que a execução possa prosseguir de forma coordenada. Essa comunicação provê uma flexibilização pontual na propriedade de isolamento para permitir a sincronização das transações. Essa abordagem se difere das outras com relação ao motivo pelo qual o isolamento é flexibilizado. Enquanto as abordagens anteriores flexibilizam o isolamento para minimizar o gargalo causado pela seriação, as transações emaranhadas flexibilizam o isolamento para permitir execução coordenada entre transações.

Em resumo, essas três abordagens apresentam diferentes estratégias para estender o modelo transacional clássico enquanto preservam a consistência por meio da seriação. No entanto, os protocolos de bloqueio tradicionais ainda são muito restritos, principalmente em cenários que necessitam expor resultados intermediários entre as transações. Para superar essa limitação, alguns modelos transacionais avançados consideram aspectos semânticos das aplicações com objetivo de tornar os protocolos de bloqueio mais flexíveis. A seguir, serão objetivo de tornar os protocolos de bloqueio mais flexíveis. A seguir, serão apresentadas duas abordagens que flexibilizam a seriação enquanto garantem a consistência do banco de dados.

Na primeira abordagem, García-Molina e Salem (1987) propuseram o modelo SAGAS para resolver problemas de longas esperas pelas transações. O principal objetivo foi propor uma alternativa para evitar que a execução das transações longas fossem realizadas de forma atômica. Para isso, a propriedade de atomicidade foi generalizada para somente garantir a terminação da transação. Ou seja, a execução de uma transação é totalmente executada ou totalmente cancelada (tudo ou nada), mas não em um passo atômico. Assim, uma transação longa pode ser representada por uma sequência de passos independentes conhecidos como passos SAGAS. Um passo SAGA é uma transação clássica que pode confirmar ou abortar. Quando um passo SAGA efetua o ponto de confirmação, seus resultados são expostos a outros passos SAGAS. Essa exposição não leva o banco de dados a um estado inconsistente uma vez que o modelo SAGAS assume que os passos são independentes e não acessam itens de dados compartilhados. Portanto, a abordagem pressupõe que dependências não serão criadas durante a execução de uma transação longa. Quando uma transação aborta, a abordagem reverte todos os efeitos produzidos pelos passos SAGAS já executados por meio de passos compensatórios. Dessa forma, para cada passo SAGA p_i, deve existir um passo compensatório c_i. O passo compensatório c_i desfaz semanticamente as ações realizadas por p_i. A principal contribuição dessa abordagem foi a concepção do conceito de passos compensatórios. No entanto,

ela é vulnerável a inconsistências em cenários nos quais os itens de dados são acessados de forma concorrente pelas transações longas.

A segunda abordagem é o Modelo Hierárquico Aberto (*Open Nested Model*) (Weikum e Schek, 1992), que estende o modelo hierárquico original (Moss, 1985) ao levar em consideração a semântica da aplicação para flexibilizar a seriação. Mais especificamente, o modelo permite que o projetista defina operações em um nível mais abstrato agrupando operações de mais baixo nível, tais como leitura e escrita. Cada operação é mapeada como uma sub-transação na hierarquia. Além disso, o projetista deve prover a relação de conflito entre as operações. Por exemplo, suponha as operações `add` e `remove` que atuam sobre um conjunto de dados implementado por meio de uma lista encadeada. A operação `add(i)` insere um elemento i no conjunto e uma operação `remove(j)` remove um elemento j do conjunto. Se $i = j$, as operações `add` e `remove` são conflitantes e não podem ser executadas em paralelo por transações distintas. Por outro lado, se $i \neq j$ as operações `add` e `remove` não conflitam e podem ser executadas paralelamente por transações distintas, ou seja, elas são comutativas. Ao considerar a comutatividade entre as operações, a abordagem permite a exposição segura de resultados intermediários com a garantia de que não serão criadas dependências entre as transações. Assim, somente as sub-transações que comutam entre si podem acessar os mesmos itens de dados sem comprometer a consistência do banco de dado em caso de falha.

Essas abordagens flexibilizam a seriação ao considerar aspectos semânticos das aplicações. Por um lado, o Modelo SAGAS flexibiliza a atomicidade e assume que não haverá compartilhamento de itens de dados entre as transações longas. Portanto, as sub-transações são independentes e podem ser executadas sem causar inconsistências. O modelo garante que a transação longa finalizará com a execução de todos os seus passos ou, em caso de falha, o desfazimento dos passos já executados. Por outro lado, o Modelo Hierárquico Aberto, permite o acesso aos dados compartilhados ao considerar operações mais abstratas que comutam entre si. A seguir será apresentada uma abordagem que substitui a seriação por um critério de correção alternativo representando, assim, a terceira classe restrita das esferas de controle.

O Modelo ConTract (Reuter *et al.*, 1997; Wächter e Reuter, 1992) foi proposto para superar as limitações do modelo SAGAS, permitindo o compartilhamento de dados entre transações longas. Um ConTract é uma transação longa representada por ações pré-definidas (passos) que são executadas de acordo com uma descrição do fluxo de controle (*script*). Assim, como no modelo SAGAS, cada passo tem um passo compensatório que desfaz semanticamente seus efeitos. O critério de corretude é definido por meio de invariantes. Cada passo possui um invariante de entrada que deve ser satisfeito para habilitar sua execução. Além disso, cada passo também possui um invariante de saída que deve ser satisfeito no término da sua execução. Um invariante é um predicado normalmente baseado em itens de dados compartilhados no banco de dados. Esses invariantes mantêm a consistência do banco de dados sob o ponto de vista semântico. Dessa forma, uma inconsistência é caracterizada quando o invariante do passo a ser executado é violado. Essa abordagem flexibiliza o isolamento entre a execução dos passos, pois não controla as transações nesse intervalo. Resumindo, o Modelo ConTract substitui o protocolo de bloqueio tradicional por um mecanismo que garante a consistência sob o ponto de vista semântico por meio da avaliação de regras.

2.4.2 A Abordagem WED-*flow*

A abordagem WED-*flow* (*Workflow, Event processing and Data-flow*)(WED-flow, 2016), inicialmente proposta em (Ferreira *et al.*, 2010a,b), é uma alternativa para modelagem e implementação de processos de negócio e vem sendo mantida pelo grupo de pesquisa DATA (do inglês, *Database Modeling, Transactions, and Data Analysis*) do IME-USP. Normalmente, os elementos dos processos de negócio, como eventos, condições, transições e dados, são vistos a partir de diferentes perspectivas. A abordagem WED-*flow* propõe uma integração desses elementos de acordo com três importantes regras (Ferreira *et al.*, 2010a): (i) para cada evento, um conjunto distinto de transições que modifica o estado de dado atual deve ser descrito; (ii) qualquer estado de dado que for consultado, modificado ou gerado por uma transição, deve ser explicitamente representado e armazenado permanentemente; e (iii) todas as transições que alteram um estado de dado devem ser disparadas por eventos quando um conjunto específico de condições for satisfeito.

O fluxo das atividades das instâncias dos processos de negócio na abordagem WED-*flow* é determinado por *triggers* que capturam o acontecimento dos eventos materializados por estados de dados. Com essa finalidade, as *triggers* são determinadas por um conjunto de pares condição-transição que determinam quais atividades devem ser executadas em uma instância do processo de negócio. Assim, toda vez que um evento ocorre e satisfaz um conjunto de condições sobre os valores dos atributos, isto é, os estados de dados, a execução de uma transição é habilitada. Dessa forma, os eventos são tratados pela execução de transições que produzem novos estados de dados, os quais também serão avaliados e podem disparar outras transições. Portanto, o fluxo das atividades é determinado dinamicamente durante a execução das instâncias.

Mais recentemente, a abordagem WED-*flow* foi expandida em (Ferreira *et al.*, 2012), para fornecer recuperação transacional por meio da evolução incremental da manipulação de exceções. Em (Garcia *et al.*, 2012a), um arcabouço foi devidamente implementado de acordo com os aspectos teóricos da abordagem WED-*flow*.

Nesta seção, serão descritos os fundamentos teóricos apresentados em (Ferreira *et al.*, 2012) e a implementação da ferramenta WED-*tool* proposta em (Garcia *et al.*, 2012a). Além disso, alguns aspectos a respeito do mecanismo para manipulação de exceções propostos em (Ferreira *et al.*, 2012) e devidamente implementados em (Silva, 2013) também serão discutidos.

Definições

- **WED-*attributes***
  Um processo de negócio pode envolver diferentes bancos de dados autônomos. Entretanto, nem todos os atributos desses bancos são de interesse do processo de negócio. Cada aplicação possui um conjunto de atributos que são utilizados para definir o fluxo de atividades e que são alterados durante a execução do processo de negócio. Com isso, é denotado por WED-*attributes* a tupla $\mathcal{A} = \langle a_1, a_2, \ldots, a_n \rangle$, em que cada a_i (com $1 \leq i \leq n$) é um atributo de interesse da aplicação.

- **WED-*state***
  Um WED-*state* é uma tupla $\langle v_1, v_2, \ldots, v_n \rangle$, em que v_i (com $1 \leq i \leq n$) é o i-ésimo atributo de $\mathcal{A}$. Ao considerar os WED-*attributes* como atributos de uma relação de um banco de dados, um WED-*state* é uma tupla específica (ou seja, uma linha) dessa relação. O conjunto $\mathcal{S}$ é o

conjunto de todos os possíveis WED-*states* de uma aplicação e é definido formalmente como:

$$\mathcal{S} = \{\langle v_1, v_2, \ldots, v_n\rangle \mid \forall i \in [1; n], v_i \in domain(a_i)\},$$

no qual a_i é o i-ésimo atributo de $\mathcal{A}$ e $domain(a_i)$ é o conjunto de valores que esse atributo pode assumir.

- **WED-*condition***
  Uma WED-*condition* é um conjunto de predicados definidos sobre os WED-*attributes* de uma aplicação. O conjunto $\mathcal{C}$ representa todas as WED-*conditions* definidas para a aplicação. Seja um WED-*state* $s \in \mathcal{S}$ e uma WED-*condition* $c \in \mathcal{C}$, é dito que s satisfaz c se os valores dos atributos de s tornam os predicados de c verdadeiros.

- **WED-*transition***
  Uma WED-*transition* t é uma função $t : \mathcal{S} \rightarrow \mathcal{S}$, ou seja, uma função que recebe um WED-*state* como entrada e produz outro WED-*state* como saída. A especificação de uma WED-*transition* t inclui um conjunto de atributos $\mathcal{U}_t = \langle u_1, u_2, \ldots, u_m\rangle$, tal que u_i (com $1 \leq i \leq m \leq n$) é um elemento de $\mathcal{A}$ o qual indica quais atributos são atualizados por t. É denotado por $\mathcal{T}$ o conjunto de todas as WED-*transitions* definidas para a aplicação. Uma WED-*transition* pode ser vista como uma transação convencional que é implementada com base no modelo SAGAS. Assim, para uma transição $t \in \mathcal{T}$ pode ser projetada uma WED-*compensation*, definida como uma função $c_t : \mathcal{S} \rightarrow \mathcal{S}$ que compensa t, desfazendo semanticamente os efeitos de sua execução. Dessa maneira, a execução de c_t sobre um WED-*state* gerado pela execução de t deve produzir um WED-*state* equivalente ao usado como entrada para a execução de t. Ou seja, se $t(s_i) = s_f$, então $c_t(s_f) = s'_i$, sendo s'_i equivalente a s_i (com s_i, s_f e $s'_i \in \mathcal{S}$).

- **WED-*trigger***
  Um WED-*trigger* é um par $g = \langle c, t\rangle$, no qual c é uma WED-*condition* de $\mathcal{C}$ e t é uma WED-*transition* de $\mathcal{T}$. Quando um WED-*state* $s \in \mathcal{S}$ satisfaz a condição c, g é responsável por disparar a transição t. O conjunto $\mathcal{G}$ representa o conjunto de todos os WED-*triggers* definidos para a aplicação.

- **WED-*flow***
  Um WED-*flow* é uma tripla $\langle \mathcal{G}', c_i, c_f\rangle$, onde $\mathcal{G}' \subseteq \mathcal{G}$ é um conjunto de WED-*triggers* e c_i e c_f são WED-*conditions*. A condição c_i é a condição inicial do WED-*flow*, a qual define o conjunto de WED-*states* iniciais válidos, e a condição c_f é a condição final do WED-*flow*, a qual define o conjunto de WED-*states* finais válidos. Quando um novo WED-*state* é fornecido como inicial, uma nova instância do WED-*flow* é criada para manipular esse estado.

- **WED-*state* AWIC-*consistent***
  As Restrições de Integridade da Aplicação (AWICs, do inglês *Application-Wide Integrity Constraints*) são condições lógicas que abrangem todos os bancos de dados autônomos utilizados pela aplicação. Essas restrições também incluem as regras semânticas que expressam as regras de negócio (por exemplo, limite financeiro para operações no cartão de crédito). Uma AWIC pode ser definida por meio de predicado sobre os WED-*attributes* da aplicação. O conjunto

$\mathcal{W}$ representa todas AWICs da aplicação, tal que $\mathcal{W} \subseteq \mathcal{C}$. Um WED-*state* é dito ser AWIC-*consistent* se ele respeita todas as AWICs definidas pela aplicação. O WED-*state* inicial de uma instância do WED-*flow* não é AWIC-*consistent* e o WED-*state* final é AWIC-*consistent*. Se um WED-*state* não é AWIC-*consistent*, então alguma ação deve ser tomada com a intenção de prover um novo estado AWIC-*consistent*.

Quando um novo WED-*state* s é gerado na aplicação por algum evento externo e s não é AWIC-*consistent*, uma nova instância do WED-*flow* pode ser criada para manipular esse estado. O conjunto $\mathcal{F}_s$ de WED-*flows* capaz de tratar s é definido como:

$$\mathcal{F}_s = \{f = \langle \mathcal{G}_f, c_{i_f}, c_{f_f} \rangle \mid f \in F \wedge s \; satisfies \; c_{i_f}\},$$

isto é, todos os WED-*flows* da aplicação no qual s representa um estado inicial válido. Se $\mathcal{F}_s = \emptyset$ então a aplicação permanecerá paralisada em s. Caso contrário, um WED-*flow* f será selecionado a partir de $\mathcal{F}_s$ para ser instanciado, gerando uma instância i.

- **WED-*state* *Transaction-Consistent***
  Assuma um WED-*state* $s \in S$ que pertence a uma instância i. Seja $f = (\mathcal{G}_f, c_{i_f}, c_{f_f})$ o WED-*flow* associado a instância i. O WED-*state* s é um *transaction-consistent* se ele respeita pelo menos uma das seguintes propriedades:
  - $\exists \langle c_g, t_g \rangle \in \mathcal{G}_f$, tal que s satisfaz c_g. Em outras palavras, se s habilita o disparo de pelo menos uma transição na instância i;
  - Existe pelo menos uma transição em execução na instância i. Isso significa que, embora s não pode disparar uma transição de estado, ele não será o último estado gerado na instância, pois cada uma das transições em execução, ao terminar a execução, produzirá um novo estado para i.

  O WED-*state* inicial de uma instância deve ser *transaction-consistent*. Caso contrário, o WED-*flow* não será capaz de evoluir esse estado para um estado AWIC-*consistent*.

- **WED-*state* inconsistente**
  Um WED-*state* é inconsistente se ele não é AWIC-*consistent* nem *transaction-consistent*.

  Um WED-*state* inconsistente na instância pode ser resultado da ocorrência de um dos seguintes eventos:
  - o aborto de uma WED-*transition* devido ao término do tempo máximo de execução;
  - a interrupção de uma WED-*transition* devido à uma decisão tomada em tempo de execução. Por exemplo, um cancelamento solicitado pelo usuário da aplicação; e
  - um erro no projeto do WED-*flow*.

- **História de uma instância WED-*flow***
  Um registro na história armazena a execução de uma WED-*transition* e pode ser representada pela 4-tupla $\langle s_c, t, s_i, s_o \rangle$, em que:
  - $s_c \in S$ indica o WED-*state* que satisfez a condição que disparou t;

- $t \in T$ é a WED-*transition* (passo SAGA) executado;
- $s_i \in S$ indica o WED-*state* fornecido como entrada para t; e
- $s_o \in S$ indica o WED-*state* gerado como saída de t.

A história de uma instância WED-*flow* é uma sequência $e_1, e_2, \ldots, e_k$ em que cada e_i (com $1 \leq i \leq k$) é um registro na história.

- **WED-S^{-1}**
  Um passo de recuperação *backward*, denotado por WED-S^{-1}, é um passo SAGA especialmente definido para transformar um WED-*state* inconsistente de uma instância em um novo WED-*state transaction-consistent*, análogo ao último estado *transaction-consistent* da instância. Seja $s \in S$ um WED-*state* inconsistente de uma instância gerada por uma WED-*transition* $t \in T$. O passo SAGA associado a um WED-S^{-1} definido para s deve executar uma compensação de t, entre outras ações corretivas.

- **WED-S^{+a}**
  Um passo de recuperação *forward*, denotado por WED-S^{+a}, é um passo SAGA especialmente definido para transformar um WED-*state* inconsistente de uma instância em um novo WED-*state transaction-consistent*. Ao contrário da recuperação *backward*, o novo estado gerado pela execução do WED-S^{+a} não está relacionado com os estados anteriores da instância.

- **Caminho de recuperação**
  Um caminho de recuperação é constituído de um dos seguintes elementos:
  - um passo de recuperação *backward* acompanhado por uma sequência de simples passos de recuperação (passos de compensação SAGAs) associados a cada WED-*transition*.
  - um passo de recuperação *forward*.

A partir dessas definições, Ferreira *et al.* (2012) derivaram três propriedades de consistência para a abordagem WED-*flow*:

1. Caminho normal.
   O caminho normal é uma sequência de WED-*transitions* executada que não gera estados inconsistentes, transformando o banco de dados de um estado inicial *transaction-consistent* em um estado final AWIC-*consistent*.

2. Consistência do caminho de recuperação.
   Um caminho de recuperação é uma sequência de passos SAGAS que conduz o banco de dados a um estado AWIC-*consistent*. Se a execução de uma instância desvia do seu caminho normal (isto é, produz um estado inconsistente), uma exceção é gerada. A abordagem WED-*flow* trata exceções em dois níveis. No nível sintático, como uma WED-*transition* é um passo SAGA, o mecanismo de recuperação reverte as atualizações, tal como o modelo clássico de transação. Em um nível semântico, a reconciliação é alcançada por meio de recuperação *backward* ou passos de recuperação *forward*. Cada um desses passos conduzirá a uma continuação do caminho de recuperação (normalmente, uma sequência de WED-*transitions*) que finaliza em um estado AWIC-*consistent*. Cada passo do caminho de recuperação gera um ou mais WED-*states* que dispara(m) a(s) próxima(s) WED-*transition(s)*.

3. Reuso dos passos de recuperação.
   Como os caminhos de recuperação são composições de passos de recuperação, um mesmo conjunto de recuperação pode ser reusado em diferentes caminhos de recuperação.

Uma vez descritas as definições que dão sustentação para a abordagem WED-*flow*, a seguir, será discutido como os processos de negócio são modelados e implementados usando a abordagem.

Modelagem

Na abordagem WED-*flow*, a modelagem de um processo de negócio é realizada de forma declarativa, por meio da especificação dos elementos básicos que compõem um WED-*flow*: WED-*transitions*, WED-*attributes*, WED-*conditions* e WED-*triggers*. O processo de modelagem utilizando a abordagem WED-*flow* é composto por três fases principais, detalhadas em (Ferreira *et al.*, 2012):

- Separação do processo de negócio em duas partes: caminho normal (*happy path*) e tratamento de exceções (aquelas esperadas);
- Modelagem de ambas as partes em uma composição dos quatro conceitos fundamentais (eventos, dados, condições e transições);
- Escrita do modelo WED-*flow* utilizando uma linguagem concreta de especificação. O resultado será interpretado e convertido automaticamente em programas executáveis por meio de ferramentas de software.

Na primeira fase da modelagem são separados os eventos do caminho normal e os eventos das exceções esperadas. O conjunto de eventos do caminho normal representa aquele conjunto mínimo de eventos necessários para atingir o objetivo principal do processo de negócio. Na abordagem WED-*flow*, os estados de dado representam a materialização dos eventos da aplicação. O caminho normal é obtido a partir das transições entre esses estados de dados. Todos os demais eventos são considerados eventos de exceção. Exceções esperadas caracterizam a ocorrência de desvios previstos do caminho ideal. Eventos não esperados podem ocorrer em qualquer ponto do caminho normal e também desviam do objetivo do processo de negócio. A principal diferença é que os eventos não esperados são aqueles imprevisíveis em tempo de projeto e são tratados em tempo de execução pelo gerenciador de recuperação do WED-*flow*.

Os eventos normais estão vinculados a diferentes entidades ou classes de dados, sendo que cada uma possui uma sequência específica de estados de dados válidos. No processo de reserva ilustrado na Figura 2.1, existem três entidades (reserva, veículo e motorista) que apresentam distintas sequências de estados de dados, conforme ilustrado pela Figura 2.9. Nessa figura, os estados de dados são representados pelos retângulos com cantos arredondados e os tempos de duração dos estados de dado pertencentes às sequências são variados. Isso é ilustrado pelas variações de tamanho de cada retângulo. Dessa maneira, as sequências correspondentes às entidades precisam ser normalizadas. Nesse procedimento, os estados das diferentes sequências são vinculados de acordo com a ordenação temporal, permitindo a criação dos WED-*states*. As linhas verticais tracejadas representam o procedimento de normalização e cada coluna formada por elas representa um WED-*state* do caminho normal.

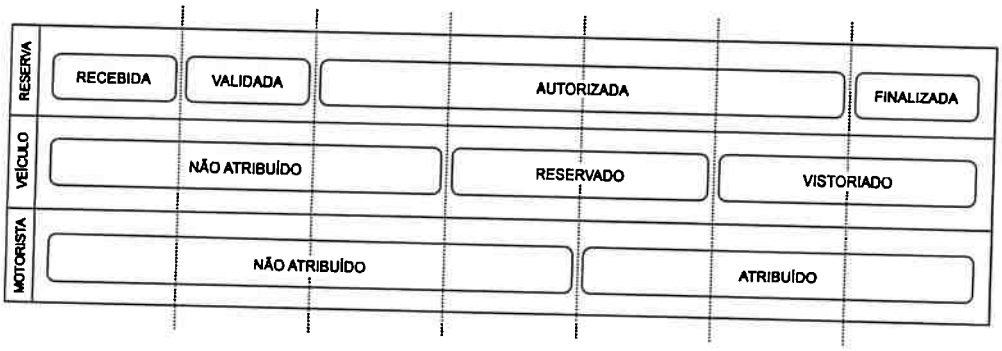


Figura 2.9: *Entidades participantes do processo de reserva do veículo. As linhas pontilhadas representam o procedimento de normalização*

A modelagem do tratamento de exceções esperadas é semelhante à modelagem do caminho normal descrita acima. Entretanto, existem algumas sutilezas que precisam ser consideradas. As exceções indicam que o fluxo de execução do processo saiu do caminho ideal e, portanto, há a produção de WED-*states* que não pertencem ao caminho normal. A ocorrência de uma exceção pode levar o processo ao cancelamento ou à escolha de um caminho alternativo que ainda conduza a sua execução para o objetivo.

Baseada no processo de reserva da Figura 2.1, a Figura 2.10 apresenta os estados de dados levando em consideração uma situação de exceção. Uma vez que o veículo reservado não estava em condições para a viagem (exceção), um outro veículo foi reservado em seu lugar (caminho alternativo). Isso ocorre porque existia um veículo disponível para reserva. Caso contrário, essa exceção causaria o cancelamento da reserva.

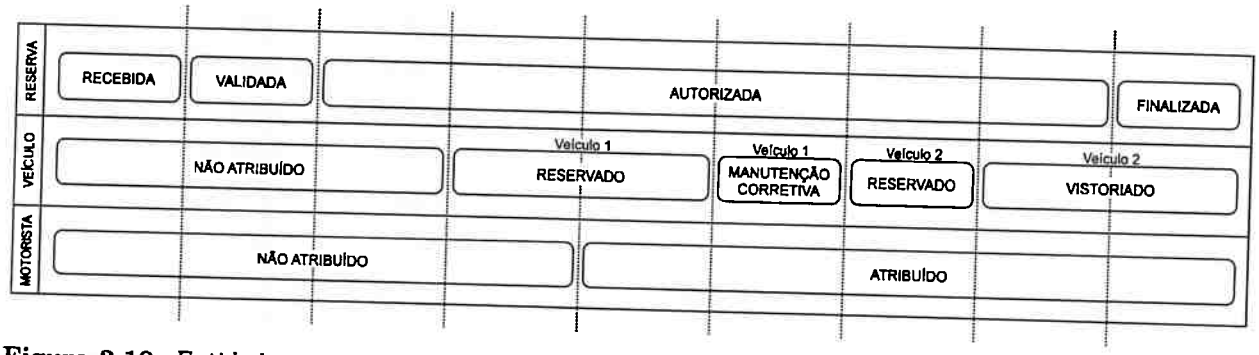


Figura 2.10: *Entidades participantes do processo de reserva do veículo com caminho alternativo. As linhas pontilhadas representam o procedimento de normalização*

A segunda fase prioriza a modelagem dos componentes básicos da abordagem WED-*flow*. Para cada evento identificado na primeira fase, WED-*states*, WED-*conditions*, WED-*transitions* e WED-*triggers* devem ser definidos. Baseada no processo de reserva de veículos, a Figura 2.11 exibe uma sequência de WED-*states* (caminho normal) referentes à execução de uma instância de WED-*flow*. Os valores sublinhados indicam aqueles que satisfazem as WED-*conditions*. As WED-*transitions* são ilustradas pelas setas e têm as seguintes atribuições:

- t_1 - Validar dados da reserva;
- t_2 - Autorizar reserva de veículo;
- t_3 - Reservar veículo;
- t_4 - Atribuir motorista;

- t_5 - Realizar a vistoria do veículo; e
- t_6 - Finalizar a reserva.

	RESERVA		VEÍCULO		MOTORISTA	
	id	status	id	status	id	status
S_0	R01	Recebida	-	Não atribuído	-	Não atribuído
S_1	R01	Validada	-	Não atribuído	-	Não atribuído
S_2	R01	Autorizada	-	Não atribuído	-	Não atribuído
S_3	R01	Autorizada	V01	Reservado	-	Não atribuído
S_4	R01	Autorizada	V01	Reservado	M05	Atribuído
S_5	R01	Autorizada	V01	Vistoriado	M05	Atribuído
S_6	R01	Finalizada	V01	Vistoriado	M05	Atribuído

Figura 2.11: *WED-*states *para o exemplo de reserva do veículo*

O WED-*state* S_0 inicial dispara a WED-*transition* t_1 que realiza a validação dos dados da reserva. Como resultado, a t_1 produz o WED-*state* S_1. A seguir, a WED-*transition* t_2 é disparada, autorizando a reserva e produzindo o WED-*state* S_2. Esse estado habilita o disparo de duas transições em paralelo. Apesar de ser indeterminado qual a ordem de término das duas transições, nesse exemplo, primeiramente, a WED-*transition* t_3 produz o WED-*state* S_3 e, logo após, a WED-*transition* t_4 produz o WED-*state* S_4. Note que existe uma WED-*condition* que avalia os dados produzidos pelas transições t_3 e t_4. Nesse caso, a WED-*condition* atua como uma junção das duas ramificações, permitindo o disparo da WED-*transition* t_5 apenas quando as duas ramificações (t_3 e t_4) encerraram sua execução. Então, a WED-*transition* t_5 realiza a vistoria do veículo e produz o WED-*state* S_5. Por fim, a WED-*transition* t_6 produz o estado final da instância.

Além dos WED-*states*, WED-*conditions* e WED-*transitions* ilustrados pela Figura 2.11, também é necessário modelar as WED-*triggers*, pares WED-*condition*-WED-*transition*, responsáveis pelos disparos das transições quando as condições associadas são satisfeitas.

Após a definição dos componentes básicos que englobam o caminho normal, é necessário definir os WED-*states*, WED-*conditions* e WED-*transitions* e WED-*triggers* que tratam os possíveis desvios do fluxo de execução. Para o processo de reserva de veículos, uma possível exceção é identificada quando o veículo, ao ser vistoriado, não é aprovado e um WED-*state* diferente do previsto para o caminho normal é produzido. Esse estado é identificado como o WED-*state* S_5 na Figura 2.12. Entretanto, como essa é uma exceção esperada, um WED-*trigger* é projetada especialmente para capturar esse evento. Desta forma, a WED-*transition* t_7 é disparada sendo responsável pela reserva de um veículo alternativo. Como resultado, o WED-*state* S_6 é produzido indicando que um veículo alternativo foi devidamente reservado. A execução da WED-*transition* t_7 pode ser vista como uma correção do fluxo de execução que, devido à exceção, foi desviado.

A última fase do processo de modelagem prevê a codificação dos componentes WED-*flow* em uma linguagem concreta que permita a execução do processo de negócio. A seguir, as principais características do núcleo da ferramenta WED-*tool* serão descritas resumidamente.

WED-*tool*

A WED-*tool* é uma ferramenta que implementa o arcabouço WED-*flow*. Essa ferramenta vem sendo desenvolvida pelo grupo de pesquisa DATA do IME-USP. A WED-*tool* é implementada utilizando a linguagem de programação orientada a objetos Ruby. O núcleo dessa ferramenta foi apresentado em (Garcia *et al.*, 2012a). As principais funções do núcleo são controlar a execução de processos transacionais e fornecer a estrutura necessária para apoiar a implementação do módulo de recuperação.

A Figura 2.13 apresenta um visão geral do funcionamento da WED-*tool*. Há basicamente três módulos implementados: configuração inicial, controle de execução e o gerenciador de recuperação. Esta seção descreve brevemente o funcionamento de cada um desses módulos.

O modelo WED-*flow*, definido nas fases anteriores da modelagem, é traduzido em uma linguagem concreta. Mais especificamente, os elementos que compõem um WED-*flow*, isto é, WED-*attributes*, WED-*conditions*, WED-*transitions* e WED-*triggers* são descritos usando a sintaxe XML. Assim, um arquivo XML com esses elementos é usado como base para o módulo de configuração. Com esse arquivo, a WED-*tool* é capaz de criar a estrutura do banco de dados necessária para o controle da execução das instâncias. Além disso, para cada WED-*transition* do modelo do processo de negócio descrito no arquivo XML, o módulo de configuração gera um modelo de classe em Ruby para que o projetista possa descrever o código de cada transição. Descrições detalhadas do *XML Schema* e do modelo de classes Ruby podem ser encontradas em (Garcia, 2013).

Uma vez especificado o modelo WED-*flow* e configurada as estruturas necessárias para o funcionamento da ferramenta, a WED-*tool* é capaz de instanciar processos de negócio e controlar a execução das instâncias. No módulo de controle de execução, para criar uma instância, um usuário deve fornecer uma valoração para os WED-*attributes* que produzirá o WED-*state* inicial. De acordo com as definições descritas na Seção 2.4.2, esse WED-*state* não é um AWIC-*consistent*. Isso faz com que um WED-*flow* seja escolhido para tratar esse WED-*state*. Se esse estado satisfaz a condição inicial de algum WED-*flow* configurado, a WED-*tool* inicia uma nova instância que deverá tratá-lo. Caso contrário, esse estado é inconsistente e será tratado pelo gerenciador de recuperação.

Na abordagem WED-*flow*, o fluxo de execução é determinado pelas WED-*triggers*, ou seja, pela avaliação das WED-*conditions* sob os WED-*states* e o disparo das WED-*transitions* quando a

	RESERVA		VEÍCULO		MOTORISTA	
	id	status	id	status	id	status
S_0	R01	Recebida	-	Não atribuído	-	Não atribuído
S_1	R01	Validada	-	Não atribuído	-	Não atribuído
S_2	R01	Autorizada	-	Não atribuído	-	Não atribuído
S_3	R01	Autorizada	V01	Reservado	-	Não atribuído
S_4	R01	Autorizada	V01	Reservado	M05	Atribuído
S_5	R01	Autorizada	V01	Manutenção Corretiva	M05	Atribuído
S_6	R01	Autorizada	V02	Reservado	M05	Atribuído
S_7	R01	Autorizada	V02	Vistoriado	M05	Atribuído
S_8	R01	Finalizada	V02	Vistoriado	M05	Atribuído

Figura 2.12: *WED-*states *para o exemplo de reserva do veículo considerando um caminho alternativo*

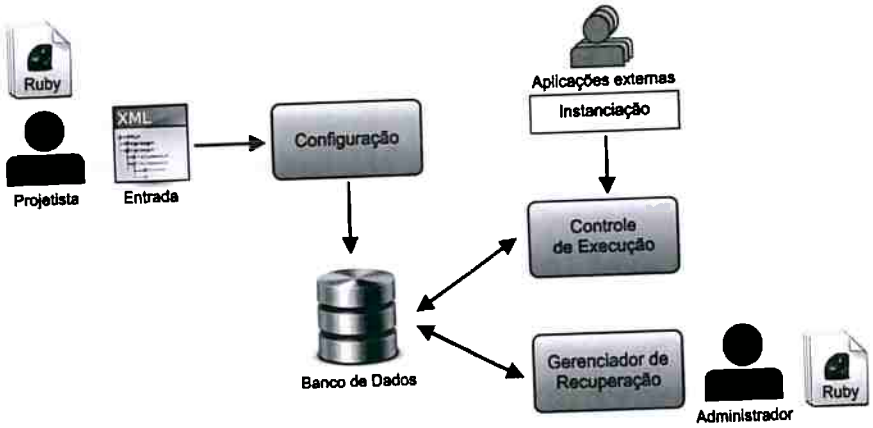


Figura 2.13: *Visão geral do funcionamento da WED-tool*

condição for satisfeita. Dessa forma, os WED-*triggers* atuam como monitores dos WED-*states*. A implementação dos conceitos de WED-*trigger* foi baseada nos fundamentos das consultas contínuas (Liu *et al.*, 2000) e das regras ECA (Bry *et al.*, 2006). Dessa forma, quando um WED-*state* s é produzido, cada WED-*trigger* $g = \langle c, t \rangle$ deve verificar se a WED-*condition* é satisfeita pelos valores de s e, em caso verdadeiro, deve disparar a WED-*transition* t. Para isso, cada WED-*trigger* possui uma fila de WED-*states* que são periodicamente avaliados. Toda vez que um WED-*state* é produzido, ele é inserido nas filas das WED-*triggers* para posterior análise. Isso é determinante para manter a característica assíncrona da WED-*tool*.

O objetivo de um WED-*flow* é conduzir uma instância do estado de dado inicial até o estado final. Entretanto, durante o fluxo de execução, uma transição pode produzir um estado de dado intermediário que, ao ser avaliado no processamento das filas, não satisfaz nenhuma condição das WED-*triggers*. Pelas definições da Seção 2.4.2, esse WED-*state* é dito ser inconsistente. Tal inconsistência é ocasionada por uma exceção inesperada e deve ser tratada pelo gerenciador de recuperação. Para exemplificar o gerenciamento das exceções, suponha que no exemplo do processo de reserva, ao avaliar as condições do veículo, o motorista indique uma manutenção preventiva (por exemplo, a troca de um pneu). A transição t_5 produz um WED-*state* S_5, ilustrado na Figura 2.14, que não havia sido inicialmente previsto no processo de modelagem. Esse WED-*state* não satisfaz nenhuma WED-*condition* e não é AWIC-*consistent*, ou seja, é um WED-*state* inconsistente.

	RESERVA		VEÍCULO		MOTORISTA	
	id	status	id	status	id	status
S_0	R01	Recebida	-	Não atribuído	-	Não atribuído
S_1	R01	Validada	-	Não atribuído	-	Não atribuído
S_2	R01	Autorizada	-	Não atribuído	-	Não atribuído
S_3	R01	Autorizada	V01	Reservado	-	Não atribuído
S_4	R01	Autorizada	V01	Reservado	M05	Atribuído
S_5	R01	Autorizada	V01	Manutenção Preventiva	M05	Atribuído
S_6	R01	Autorizada	V01	Vistoriado	M05	Atribuído
S_7	R01	Finalizada	V01	Vistoriado	M05	Atribuído

Figura 2.14: *WED-*states *do processo de reserva considerando um caso de exceção*

Como consequência, o módulo de gerenciamento de recuperação é acionado para tratar desse

estado inconsistente. O gerenciador é capaz de executar mecanismos de recuperação *backward* e *forward*. A recuperação *backward* requer a implementação de um passo SAGA WED-S^{-1} definido para transformar esse estado inconsistente em um WED-*state transaction-consistent*. Para o exemplo, esse passo SAGA teria a função de transformar o WED-*state* S_5 em um WED-*state* equivalente ao WED-*state* S_4. Entretanto, nesse exemplo, o administrador optou por executar uma recuperação *forward* e, desta forma, é necessário disponibilizar um passo SAGA S^{+a} para transformar o WED-*state* inconsistente S_5 em um novo WED-*state transaction-consistent*. Esse passo é implementado pela WED-*transition* t_8. Na prática, a manutenção preventiva é concretizada e o veículo pode, então, ser reservado para a viagem. Como descrito na Seção 2.4.2, há diferentes razões para um WED-*state* ser inconsistente. Entre elas, por exemplo, a falha de uma WED-*transition* também requer a intervenção do gerenciador de recuperação. A discussão do módulo de recuperação a falhas do WED-*flow* é descrita mais detalhadamente em (Garcia *et al.*, 2012b; Silva, 2013).

2.5 Considerações Finais

Neste capítulo, distintos modelos que têm sido amplamente usados para modelar, analisar, executar e monitorar processos de negócio foram resumidamente descritos. Primeiramente, destacou-se os modelos formais, tais como Rede de Petri, Álgebra de Processos e Modelos baseados em Grafos. O objetivo desses modelos é uma especificação clara e sem ambiguidades do processo de negócio. Por esse motivo, os modelos formais geralmente permitem a obtenção de conhecimento, análise e a correção do esquema. Apesar dos modelos formais proporcionarem diversas características que favorecem o apoio à execução dos processos de negócio, eles requerem que, em tempo de projeto, o fluxo de controle das atividades seja explicitamente definido. Tal característica conduz a dificuldades em lidar com mudanças que ocorrem em tempo de execução.

Com o objetivo de prover flexibilidade, na literatura destacam-se os Modelos Declarativos apoiados por Regras ECA e os Modelos Declarativos baseados em Restrições. O incremento da flexibilidade é obtido de duas formas. Primeiramente, modelos formais facilmente alcançam fluxos de controle excessivamente restritivos, pois a especificação explícita de todos os possíveis caminhos de execução é uma atividade complexa. Nos modelos declarativos é permitida qualquer execução que respeite as regras ou restrições do esquema do *workflow*. Segundo, novas restrições em modelos formais podem causar grandes efeitos colaterais na estrutura do esquema, enquanto nos modelos declarativos apenas requerem que uma nova regra/restrição seja adicionada ao esquema. Em outras palavras, os modelos declarativos proveem flexibilidade em tempo de execução em relação as alterações exigidas pelo processo. Entretanto, os modelos declarativos apresentam problemas relacionado ao desempenho, uma vez que diferentes caminhos de execução são permitidos tornando ineficiente e complexa a verificação e validação das restrições. A discussão entre o equilíbrio entre flexibilidade e apoio no contexto do PAISs é também discutido em (Aalst *et al.*, 1999, 2009).

Os PAISs objetivam principalmente os aspectos de modelagem dos processos de negócio, de modo a capturar com precisão os requisitos de fluxo de dados e de controle entre as atividades que compõem um processo de negócio. Por outro lado, a comunidade de banco de dados tem apresentado diversos modelos transacionais avançados para tratar de aplicações de longa duração. Ao explorar a semântica das aplicações e utilizar critérios de correção mais flexíveis do que as transações clássicas, os modelos transacionais avançados oferecem recursos especiais que podem ser usados

pelos PAISs para o controle de concorrência e a recuperação a falhas. Neste capítulo foi apresentado uma breve descrição das principais contribuição a partir da perspectiva das Esferas de Controle. Em (Kamath e Ramamritham, 1996), os autores discutem a confluência das ideias entre as comunidades científicas ao proporem características adicionais que estão sendo gradualmente contempladas pelos PAISs. Nesse sentido, neste capítulo, destaca-se o WED-*flow* como uma abordagem declarativa para modelagem e execução de processos de negócio que fornece apoio transacional para recuperação e manipulação de exceções.

Capítulo 3

Trabalhos Relacionados

Diversas abordagens têm apresentado alternativas para apoiar as adaptações instâncias em tempo de execução no contexto de *workflows*. Aalst *et al.* (2009) afirmam que essas abordagem devem buscar um equilíbrio ao tratar de processos de negócio. Por um lado, existe o objetivo de controlar e evitar as execuções incorretas e indesejáveis dos processos. Por outro lado, é esperado que os processos sejam flexíveis e não restrinjam os usuários em suas ações. Esse paradoxo tem limitado a aplicação de PAISs, uma vez que eles são restritos e apresentam problemas ao efetivar as mudanças (Aalst e Jablonski, 2000). O objetivo deste capítulo é apresentar os limites dos PAISs tradicionais ao adaptar instâncias ao mesmo tempo que os esquemas de *workflow* evoluem. Com isso, pretende-se discutir que as abordagens tradicionais que usam modelos de processos procedurais são rígidas e enfrentam grandes desafios quando mudanças são necessárias.

Normalmente, as abordagens adaptam instâncias em execução seguindo critérios que tem como objetivo evitar que as elas apresentem erros ou inconsistências ao serem executadas sobre as definições estabelecidas no novo esquema. Baseado nisso, nesta proposta, os trabalhos relacionados são classificados em (i) abordagens baseadas em *snapshot* nas quais apenas as atividades ativas ou em execução no momento da alteração são utilizadas nos critérios de migração e (ii) abordagens cujos critérios são apoiados por toda história de execução da instância.

Há outras possíveis classificações dos trabalhos relacionados à adaptação, como apresentadas em (Rinderle *et al.*, 2003) e (Rinderle *et al.*, 2004b). Entretanto, essa divisão descrita acima atende os objetivos iniciais desta proposta.

3.1 Abordagens baseadas em *snapshot*

As abordagens que tratam a adaptação de instâncias considerando apenas o estado das atividades ativas ou em execução sem manter informações das execuções anteriores são conhecidos como abordagens baseadas em *snapshot*. Nesta seção serão descritos os trabalhos propostos em (Aalst e Basten, 2002), (Ellis *et al.*, 1995) e (Aalst, 2001).

3.1.1 WF-net

A abordagem WorkFlow net (WF-net) proposta em (Aalst e Basten, 2002) concentra-se nas alterações *ad-hoc* ao tratar de eventos raros no processo e nas alterações evolucionárias englobando modificações na estrutura do processo que afetam todas as instâncias em execução. As alterações

ad-hoc implicam no gerenciamento de diferentes versões das instâncias ativas. Por outro lado, as alterações evolucionárias acarretam a migração de instâncias ativas para o esquema modificado. Baseado nessas alterações, os autores apontam dois problemas a serem superados: (i) o problema da alteração dinâmica, referenciando as anomalias causadas pela migração das instâncias ativas; e (ii) o problema de prover uma visão global agregada das instâncias ativas provenientes de diferentes versões. No entanto, nesta seção será tratado somente primeiro problema. Para abordar esses problemas, os autores propõem um modelo de *workflow* baseado em uma variação da Rede de Petri original denominada WF-net. Nessa abordagem, um *workflow* N é representado pela tupla:

$$\langle P, T, F, \ell \rangle$$

em que P denota o conjunto finito de lugares, T indica o conjunto finito de transições, tal que $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ é um conjunto de arcos direcionados e ℓ é uma função de marcação que atribui um rótulo para cada transição. O modelo deve possuir um único lugar de entrada e um único lugar de saída. O lugar de entrada i corresponde ao estado inicial e o lugar de saída i ao estado final do processo. Os autores também apresentam uma propriedade chamada *soundness* para validar o esquema. Uma WF-net é considerada *sound* se ela é conectada, segura, livre de *deadlock*, livre de transições "mortas" (transições que nunca serão ativadas) e sempre finaliza corretamente (o estado final é sempre alcançado). Mais detalhes sobre a propriedade *soundness* e sobre a representação de *workflows* por meio de WF-nets podem ser encontradas em (Aalst e Basten, 2002; Aalst, 1998). Nesse modelo, o fluxo de controle é descrito pelas transições (atividades) e pelos lugares (condições para execução das tarefas), como ilustra a Figura 3.1.

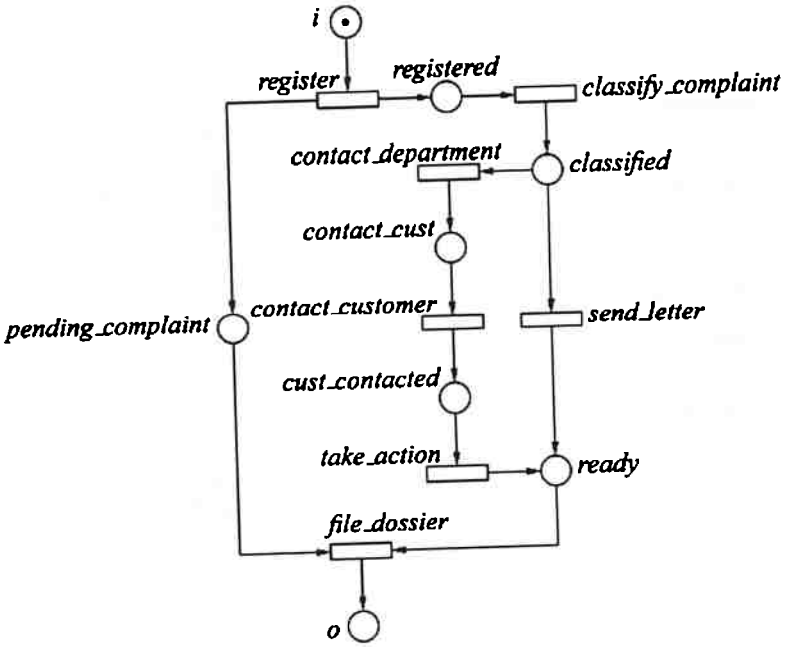


Figura 3.1: *Representação do fluxo de controle de um processo em WF-net. Fonte: Aalst e Basten (2002)*

O comportamento de uma instância é determinado pela sua estrutura e pelo seu estado. O estado de uma rede é representado por fichas sobre os lugares da rede. A distribuição das fichas sobre os lugares definem a marcação de uma rede. Portanto, uma instância é definida como (N, s), em que $N = (P, T, F, \ell)$ representa o esquema e s a marcação sobre o esquema. O fluxo de controle é

definido por regras de disparo de modo que uma transição t é disparada se cada um dos seus lugares de entrada possui uma ficha. Quando t executa, as fichas dos lugares de entrada são removidos e são adicionados nos seus lugares de saída. O fluxo de dados não é considerado nessa abordagem.

A abordagem utiliza o conceito de herança (proveniente do paradigma de orientação a objetos) para contornar os problemas das alterações estruturais no esquema. As possíveis modificações no esquema são restringidas de modo que sejam permitidas somente as alterações que preservem a relação de herança entre os esquemas. Um esquema S' é considerado uma subclasse do esquema S, se não é possível distinguir o comportamento entre S e S'. Dessa forma, um esquema pode ser modificado se, e somente se, sua relação de herança com o esquema original é preservada. Os autores propõem quatro tipos de relações de herança que englobam operações aditivas e subtrativas. Mais especificamente, as operações correspondem à construtores clássicos, tais como a adição ou remoção de estruturas cíclicas, sequenciais, ramificações paralelas ou ramificações alternativas. As relações de herança entre os esquemas são verificadas ao *bloquear* ou *ocultar* atividades que diferem do esquema original. O bloqueio implica na inibição de atividades durante a execução e o ocultamento atribui a atividade o rótulo de *tarefa silenciosa* τ . Uma tarefa silenciosa τ não tem seus efeitos visíveis e usados. A Figura 3.2 ilustra um exemplo, em que na parte (a), o esquema S pode ser mapeado para o esquema S' se a nova tarefa A é ocultada em S'. Na Figura 3.2(b), o esquema S pode ser mapeado para o esquema S' caso seja bloqueado a execução da nova tarefa B em S'. Em ambos os cenários o esquema modificado S' preserva sua relação de herança com o esquema original S. A abordagem define uma relação de equivalência entre os esquemas verificando se eles têm o mesmo comportamento (observável) por meio do conceito de *branching bisimilarity* (Aalst e Basten, 2002). Resumindo, o esquema modificado deve preservar o comportamento do esquema original por meio do bloqueio ou ocultamento de atividades.

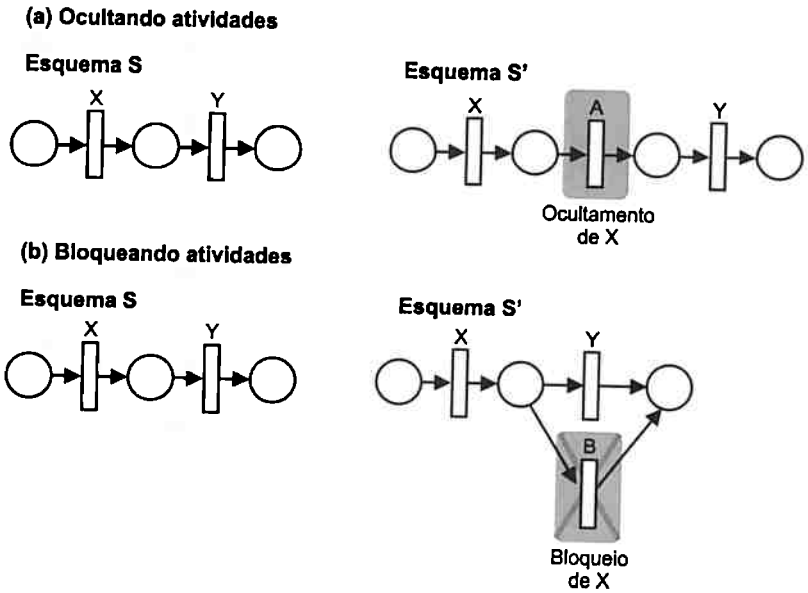


Figura 3.2: *Equivalência ao ocultar ou bloquear atividades*

Considerando a alteração Δ em que $S+\Delta = S'$, para verificar se Δ é uma alteração que preserva a relação de herança entre S e S', os autores impõem regras que devem ser respeitadas. A Figura 3.3 ilustra um exemplo de uma dessas regras, no qual a alteração Δ propõe a inserção de uma estrutura cíclica. A alteração é válida se (i) S e Δ compartilham um único lugar p e (ii) para toda ficha consumida por Δ a partir de p deve haver uma transição de Δ que retorne a ficha para p. A

Figura 3.3(a) ilustra a regra de forma visual e a Figura 3.3(b) ilustra um exemplo.

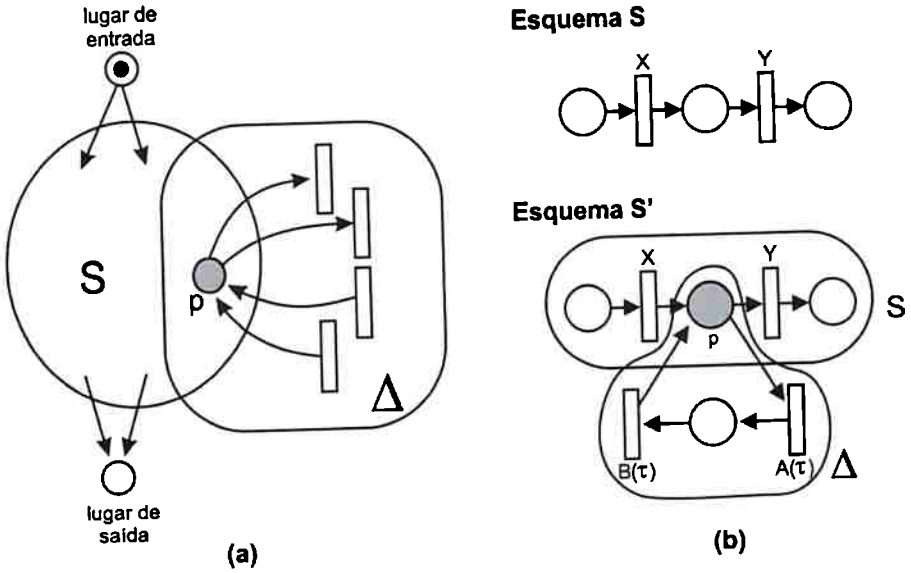


Figura 3.3: *Equivalência ao ocultar ou bloquear atividades*

Após verificar se uma instância I de S está em conformidade com o esquema alterado $S' := S + \Delta$, a abordagem disponibiliza regras de transferência (baseadas nas relações de herança) para efetivamente migrar I para S'. Dessa forma, ao preservar a relação de herança entre S e S', a abordagem garante a transferência de I. Portanto, essa abordagem não propõe uma solução ao problema da alteração dinâmica, e sim uma prevenção das anomalias decorrentes desse tipo de alteração.

Nessa abordagem, os autores apresentam um dos principais trabalhos que empregam Rede de Petri na especificação de *workflows*. O WF-net propõe o uso do conceito de herança para prover flexibilidade no apoio a adaptações estruturais do esquema. Mais especificamente, a abordagem restringe as possíveis alterações em um esquema para um grupo de alterações especiais que preservam a relação de herança entre o modelo original e o alterado. Essa restrição evita as anomalias inerentes do problema da migração de instâncias. Embora a utilização do conceito de herança represente uma contribuição importante ao evitar os efeitos colaterais da migração de instâncias, a abrangência da alteração é limitada pelas relações de herança entre os esquemas. As ações do projetista para alterar o esquema são restritas as quatro relações de herança definidas pela abordagem. Além disso, o fato da abordagem não considerar o fluxo de dados restringe ainda mais escopo em que essa solução pode ser aplicada.

3.1.2 Flow-Nets

Flow-net (Ellis *et al.*, 1995) foi uma das primeiras abordagens a contemplar o problema da migração de instâncias ao realizar alterações estruturais no esquema. A abordagem propõe uma estratégia baseada no isolamento da região do *workflow* que será alterada (*change region*). As alterações nessa região podem causar problemas na transferência das instâncias para o novo esquema. Portanto, a transferência somente é efetivada quando a instância estiver fora dessa região. Além disso, os autores propõem a verificação da correção para uma determinada classe de alterações no esquema.

A especificação do modelo do *workflow* é baseado em uma variação da Rede de Petri similar a abordagem anterior WF-net. O esquema é definido pela tupla:

$$M = \langle S, T, F, Lab; s_{in}, s_{out} \rangle$$

em que S denota um conjunto finito de lugares, T representa o conjunto de transições, $F \subseteq (P \times T) \cup (T \times P)$ é um conjunto de arcos direcionados que representa o fluxo de controle, Lab é uma função de marcação que atribui rótulos para as transições, $s_{in} \in S$ é o lugar de entrada e $s_{out} \in S$ é o lugar de saída de M. O conjunto $\{s_{in}, s_{out}\}$ representa a interface do esquema M. Uma instância $(M; m)$ consiste no esquema M e a marcação m por meio de fichas distribuídas em M. O fluxo de controle procede como na abordagem anterior: a ficha do lugar de entrada é transferida para o lugar de saída, ao executar a transição.

Nessa abordagem uma alteração Δ é definida pela substituição de uma sub-rede N_1 de M por outra sub-rede N_2, resultando em M'. Assim, N_1 é referenciada por *região antiga* e N_2 como *região nova*. A região antiga é definida pela menor sub-rede contendo todas as atividades afetadas pela alteração Δ. Essa região é conectada com o restante da rede por meio de sua interface $\{s_{in}, s_{out}\}$. A comunicação é restrita à troca de fichas pela sua interface. Os autores ressaltam que é sempre possível selecionar apropriadamente a região afetada pela alteração. Entretanto, eles não apresentam como identificar essa região. Portanto, o esquema M' é obtido a partir de M, ao remover N_1 de M e conectar N_2 na rede restante usando sua interface como encaixe. Ao substituir N_1 por N_2 as instâncias também devem ser transferidas ao novo esquema.

Os autores argumentam que o adiamento da migração das instâncias quando uma alteração ocorre é mais seguro do que fazê-la imediatamente. Esse adiamento é caracterizado por uma classe especial de alteração chamada *Synthetic Cut-Over Change* (SCOC). Ao aplicar esse tipo de alteração, a região antiga N_1 e a região nova N_2 são mantidas juntas em M', isto é, M' contém as duas versões da sub-rede modificada. Além disso, são apresentados dois cenários de alterações *downsizing* e *upsizing*. *Upsizing* significa que N_2 "faz mais" do que N_1, isto é, as transições de N_1 representam um subconjunto de N_2. *Downsizing* significa que N_2 "faz menos" do que N_1, ou seja, as transições de N_2 representam um subconjunto de N_1. A migração nas alterações de *upsizing* podem ser realizadas imediatamente, enquanto que as de *downsizing* devem ser adiadas.

Esse trabalho foi um dos primeiros a caracterizar e discutir o problema de alteração dinâmica, em que as instâncias em execução devem ser transferidas quando ocorre uma modificação estrutural no esquema. Os autores apresentam uma descrição formal para o problema e apresentam uma classe de alteração denominada *Synthetic Cut-Over Change* que mantém as versões (antiga e nova) da parte alterada do esquema. Essa estratégia permite que a migração de uma instância seja adiada, permitindo que ela continue sua execução no modelo original. Como exemplo, são apresentados dois tipos de alteração *downsizing* e *upsizing*. Nessa abordagem, os autores chamam a atenção para a definição da região crítica, a qual engloba todas as atividades afetadas pela alteração. No entanto, eles não apresentam uma forma de definir essa região.

3.1.3 Dynamic Change Bug

Aalst (2001) propõe uma abordagem para eliminar falhas na adaptação das instâncias. Mais especificamente, a abordagem apoia a migração segura das instâncias após a execução de alterações

no esquema. A abordagem modela o fluxo de controle do *workflow* com a variação da Rede de Petri chamada WF-net e não controla o fluxo de dados. Os detalhes de como o esquema é modelado e como a instância é representada por meio da WF-net foram descritos na Seção 3.1.1.

A ideia geral da abordagem é identificar a região afetada pela alteração e permitir a transferência das instâncias cujas marcações estejam fora dessa região. Essa região é identificada de forma automática e é composta pelos vértices afetados diretamente e indiretamente pela alteração. O conjunto dos vértices afetados diretamente pela alteração é chamado de região estática. A região estática é calculada ao comparar as relações de fluxo de ambos os esquemas. Todos os arcos que são removidos e/ou inseridos são registrados. O conjunto de todos os vértices (lugares e transições) conectados aos arcos que foram inseridos e/ou removidos constitui a região estática. No entanto, a identificação da região estática não é suficiente para garantir que a transferência das instâncias, cujas marcações estão fora dessa região, sejam efetuadas de forma segura.

Para garantir a transferência segura das instâncias, o autor propõe a identificação da região dinâmica estendendo a região estática. Essa expansão é resultado da execução de um algoritmo que, dado a região estática, adiciona os vértices que são indiretamente afetados pela alteração. A complexidade desse algoritmo é $O(n^4(n!)^2)$ para um *workflow* com n vértices. O autor argumenta que sob o ponto de vista prático, essa complexidade é aceitável, pois o algoritmo é executado uma única vez quando uma alteração ocorre, independente do número de instâncias em andamento.

Uma instância é considerada comportamentalmente consistente no novo esquema se, após a migração, o estado da instância pode ser alcançado a partir do vértice inicial. A região dinâmica calculada pelo algoritmo é utilizada para garantir a validade dessa condição. Isto é, somente as instâncias cujas marcações estão fora da região dinâmica podem ser transferidas para o novo esquema.

Nessa abordagem, o autor propõe uma estratégia para evitar falhas na adaptação das instâncias. Essa estratégia é caracterizada pela identificação automática dos vértices que são afetados diretamente e indiretamente pela alteração. A principal contribuição é o mecanismo que identifica os vértices que são afetados de forma indireta. O mecanismo identifica esses vértices por meio de um algoritmo que estende a região estática. Embora o algoritmo seja executado apenas uma vez quando a alteração ocorre, ele pode tornar-se custoso caso a alteração afete uma grande área do processo. Além disso, assim como nas outras abordagens baseadas em Rede de Petri, o fato da abordagem não considerar o fluxo de dados, restringe o escopo em que a solução pode ser aplicada.

3.2 Abordagens baseadas em história de execução

Nesta seção são apresentadas as abordagens que usam, de alguma forma, a história de execução para definir um critério de correção para a adaptação das instâncias ao novo esquema. Resumidamente, serão descritos os trabalhos de (Casati *et al.*, 1998), (Kradolfer e Geppert, 1999; Kradolfer *et al.*, 1999), (Reichert e Dadam, 1997, 1998), (Rinderle *et al.*, 2004a) e (Aalst *et al.* , 2009).

3.2.1 WIDE

O projeto WIDE (Casati *et al.*, 1998) destaca a importância do efetivo gerenciamento da evolução dos esquemas de *workflow* e da adaptação das instâncias em execução. Mais concretamente, os autores destacam que o principal desafio em modificar o esquema de um *workflow* é o gerenciamento

das instâncias que foram inicializadas de acordo com o antigo esquema. Para tratar desse problema, os autores propõem um modelo de *workflow* baseado em grafo. Dessa forma, um *workflow* W é formalmente descrito pela quíntupla:

$$\Sigma^W = \langle Tasks^W, Vars^W, \sigma^W, \varsigma^W, \theta^W \rangle$$

em que $Tasks^W$ é um conjunto de tarefas de W, Var^W é um conjunto de variáveis de W, σ^W é a função de sucessão associada a cada tarefa $t \in Tasks^W$, ς^W é a função de condição associada a cada tarefa $t \in Tasks^W$ e θ^W é a função que associa o tipo de ligação com a tarefa sucessora (isto é, direta, divisão, junção total, junção iterativa). As três funções (σ, ς e θ) são responsáveis por definir o fluxo de controle do *workflow*. Quando uma tarefa t é finalizada, o conjunto $\sigma(t)$ é considerado; para cada tarefa $p \in \sigma(t)$, se $\varsigma(p)$ é satisfeita, a tarefa p é inicializada. Uma condição $\varsigma(t)$ é expressa por um conjunto de pares $\langle varName, value \rangle$ sendo considerada verdadeira se, e somente se, para cada par no conjunto, o valor de $varName$ é igual a $value$.

Um esquema é considerado *legal* se, e somente se, o fluxo de controle permite que todas as tarefas sejam alcançáveis. Para formalmente definir *legalidade* para um esquema deve-se considerar as seguintes definições:

- **Fecho transitivo sucessor:** o fecho transitivo da função de sucessão, denotado por σ^* é definido como: $\sigma^*(t) = \{p : (p \in \sigma(t) \vee (\exists s : s \in \sigma(t) \wedge p \in \sigma^*(s)))\}$
- **Alcançabilidade:** uma tarefa t é alcançável se e somente se $t \in \sigma^*('start')$
- **Função de predecessor:** a função de predecessor, denotada por $\pi(s)$, representa o conjunto de tarefas que tem s como sucessor: $\pi(s) = \{t : s \in \sigma(t)\}$

A definição de alcançabilidade implica na existência de um caminho que conecta o ponto de início a toda tarefa no *workflow*. Entretanto, isso não evita que existam tarefas que nunca são executadas devido à condições que não sejam satisfeitas considerando todos caminhos.

Uma instância I de um esquema W pode ser descrita por W e por sua história de execução $\mathcal{H} = (\langle \epsilon^S_{I,0}, \mu^S_{I,0} \rangle, ..., \langle \epsilon^S_{I,i}, \mu^S_{I,i} \rangle)$, em que $\epsilon^S_{I,k}$ denota a k^{th} finalização da tarefa em I e $\mu^S_{I,k}$ denota um operação de escrita na variável executada por $\epsilon^S_{I,k}$.

A abordagem define primitivas WFML (*WorkFlow Modification Language*) que permitem modificar o esquema do *workflow*. O objetivo dessas primitivas é manter a consistência estrutural e a comportamental. A consistência estrutural implica que, após qualquer sequência de modificações, o esquema resultante mantém-se *legal*. A consistência comportamental refere-se às instâncias em execução e garante que qualquer conjunto de primitivas aplicadas sobre o esquema não implica em erros de execução das instâncias em andamento.

As primitivas WFML são divididas em duas categorias:

- **primitivas de declaração** alteram a declaração das variáveis e tarefas. As primitivas são *AddVar(VarName v, DefaultValue d)*, *RemoveVar(VarName v)* e *AddTask(Task t)*;
- **primitivas de fluxo** alteram o fluxo de controle do esquema, isto é, a sequência no qual as tarefas devem ser executadas. As primitivas de fluxo são *AddSuccessor(Task t, Task s)*, *RemoveSuccessor(Task t, Task s)*, *ModifyType(Task t, Type T)* e *ModifyCondition(Task t, Condition c)*.

Dado um conjunto de primitivas aplicadas ao esquema, a abordagem permite apenas a migração das instâncias que possam ser representadas no novo esquema. Uma instância i é dita complacente ao novo esquema W', se sua execução pode ser representada em W'. Para isso, a abordagem deve verificar se a história de execução da instância i poderia ser gerada no esquema W'. Essa estratégia pode ser custosa computacionalmente uma vez que a história de execução de uma instância pode ser muito grande na presença de estruturas de repetição. Entretanto, quando o esquema é acíclico, é possível realizar essa verificação em tempo $O(n)$. Caso uma instância não seja complacente, o administrador do sistema deve atuar com o propósito de adaptar as instâncias de forma *ad-hoc*.

Casati *et al.* (2000) estendem o modelo WIDE com o objetivo de aumentar o nível de flexibilização do *workflow*. Nesse trabalho, os autores propõem uma abordagem baseada em regras juntamente com a representação tradicional do fluxo normal das atividades. As regras são usadas para apoiar a evolução do esquema e o gerenciamento de exceções. Embora o uso de regras para especificar o que acontece na evolução e situações excepcionais facilita o processo de especificação e aumenta a flexibilidade do *workflow*, a abordagem permite apenas adaptações restritas relacionadas à eventos não esperados.

3.2.2 TRAMs

TRAMs (Kradolfer e Geppert, 1999; Kradolfer *et al.*, 1999) apresentam um arcabouço declarativo para evolução de esquemas de *workflow* baseado no versionamento do modelo e na migração das instâncias em execução. Uma nova versão sempre é criada quando um modelo existente é modificado. A abordagem provê mecanismos para apoiar a validação das alterações dinâmicas no esquema, bem como a transferência de instâncias em andamento para o modelo adaptado. Nesse arcabouço, as descrições do fluxo de controle e do fluxo de dados são realizadas de forma declarativa. Uma alteração é realizada pela execução de operações disponibilizadas previamente pelo arcabouço. O critério de correção estrutural do esquema é definido por invariantes, isto é, condições relacionadas ao esquema que devem ser preservadas. Uma alteração é considerada correta se os invariantes vinculados ao esquema não são violados. Além disso, cada operação de alteração definida pela abordagem, especifica uma condição de transferência que deve ser avaliada quando uma instância for migrada para o esquema alterado. Assim, o critério de correção comportamental de uma instância é definido pela concatenação das condições de transferência associadas às operações que modificaram o esquema. Uma instância é comportamentalmente consistente com o novo esquema e, consequentemente, pode ser transferida com segurança, se satisfaz todas as condições impostas pelas alterações aplicadas ao esquema. O arcabouço baseia-se no histórico de execução da instância para realizar essa validação.

Um modelo de *workflow* é especificado por meio de construtores disponibilizados pelo arcabouço. A Figura 3.4(a) ilustra os principais construtores da abordagem. Resumidamente, o modelo é definido por uma entidade denominada de tipo de *workflow* (*Workflow Type*). Um tipo de *workflow* é definido por dois tipos de atividades: tipos complexos (*Complex Workflow Type*) ou tipos atômicos (*ActivityType*). Um tipo atômico é caracterizado por uma atividade realizada em um único passo, na qual não especifica como a execução procederá. Um tipo complexo define um conjunto de *sub-workflows*, bem como o fluxo de controle e o fluxo de dados entre eles. Ambos os tipos mantêm versões decorrentes das possíveis alterações no modelo. Cada versão possui uma interface na qual são especificadas as informações do *workflow* que ficarão visíveis e, consequentemente, poderão ser

acessadas por outras versões. O fluxo de controle e o fluxo de dados são especificados por meio do construtor *body*, como ilustra a Figura 3.4(b).

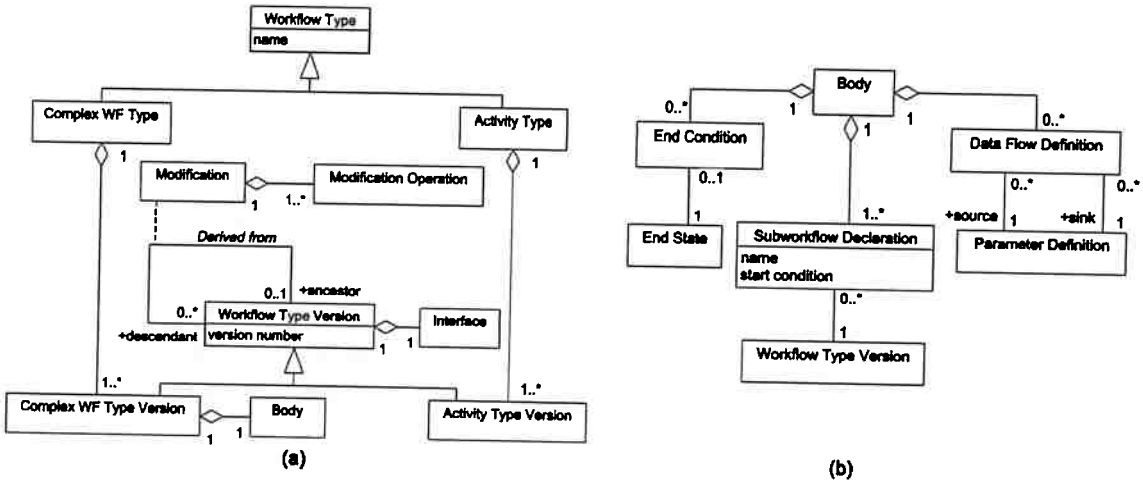
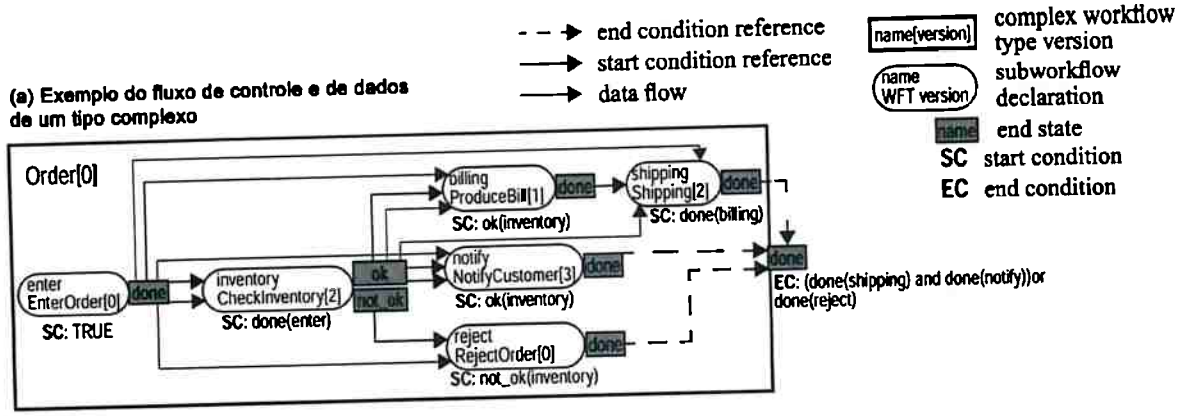


Figura 3.4: *Principais construtores do modelo de* workflow *TRAMs. Fonte: Kradolfer e Geppert (1999)*

O fluxo de controle é definido de forma declarativa ao especificar guardiões (*guards*) sobre os estados das transições do *workflow*. Esses guardiões são descritos na forma *condições de entrada* que habilitam a execução dos *sub-workflows*. As condições de entrada são satisfeitas pelos estados finais produzido pelos *sub-workflows*. Além disso, o *workflow* do tipo complexo também pode definir uma *condição final* para cada um dos seus estados finais. A condição final permite que o *workflow* seja finalizado. O fluxo de dados é estabelecido pela associação de um parâmetro de saída (*origem*) a um parâmetro de entrada (*destino*) de *sub-workflows* subsequentes. Os parâmetros de entrada e saída, tal como os estados finais, formam o conjunto de informações visíveis a outros *workflows* e são definidos por meio do construtor *interface*, como mencionado anteriormente.

A Figura 3.5(a) ilustra a especificação do fluxo de controle e do fluxo de dados de um *workflow* que gerencia pedidos de compra em uma empresa. Para cada *sub-workflow*, são definidas condições de entrada que disparam suas execuções; as setas sólidas na cor preta indicam quais estados são referenciados pelas condições de entrada. As setas tracejadas especificam os estados que formam a condição do estado final do *workflow*. O fluxo de dados é representado pelas setas sólidas na cor cinza que indicam a origem (parâmetros de saída do estado gerado por um *sub-workflow*) e o destino (parâmetros de entrada de um *sub-workflow* subsequente) do fluxo. O fluxo de dados é especificado explicitamente como ilustra a Figura 3.5(c). De forma superficial, as setas na cor preta representam o fluxo de controle e as de cor cinza representam o fluxo de dados do *workflow*. A Figura 3.5(b) ilustra a definição da interface do *workflow* e dos seus *sub-workflows*. O esquema é dito estruturalmente correto se o fluxo de controle e o fluxo de dados não violam os invariantes definidos. Os autores apresentam esses invariantes de forma textual e não especificam como elas são aplicadas ao esquema.

Uma instância é caracterizada por meio do histórico de execução do *workflow*. O histórico registra todas as alterações de estado durante a execução do *workflow*. Pode-se dizer que as alterações de estados refletem a ocorrência de eventos no processo. Portanto, também pode-se dizer que a instância é caracterizada pela história dos eventos ocorridos. Quando a condição de entrada de um *workflow* é satisfeita, um evento é iniciado. Quando o *workflow* finaliza e gera o estado, o evento é encerrado. A Figura 3.6 ilustra a história dos eventos ocorridos em uma instância do *workflow*



(b) Especificação da interface

WFT Version	In-Parameters	End State	Out-Parameters
Order[0]	-	done	-
EnterOrder[0]	-	done	string product_descr, integer customer_id
CheckInventory[2]	string product_descr	ok	integer product_id
		not_ok	-
ProduceBill[1]	integer customer_id, integer product_id	done	-
NotifyCustomer[3]	integer customer_id, integer product_id	done	-
Shipping[2]	integer customer_id, integer product_id	done	-
RejectOrder[0]	integer customer_id, string product_descr	done	-

(c) Especificação do fluxo de dados

Source	Sinks
Order[0].enter.done.product_descr	Order[0].inventory.product_descr, Order[0].reject.product_descr
Order[0].enter.done.customer_id	Order[0].billing.customer_id, Order[0].notify.customer_id, Order[0].shipping.customer_id, Order[0].reject.customer_id,
Order[0].inventory.ok.product_id	Order[0].billing.product_id, Order[0].notify.product_id, Order[0].shipping.product_id,

Figura 3.5: *Exemplo de um modelo de* workflow *especificado no arcabouço TRAMs. Fonte: (Kradolfer e Geppert, 1999)*

id	kind	WFT version	subworkflow declaration	id super-workflow	parameter value list	time-stamp
1	Start	Order[0]	-	-	-	25
2	Start	EnterOrder[0]	Order[0].enter	1	-	26
2	done	EnterOrder[0]	Order[0].enter	1	product_descr = "A 342 L", customer_id = 78	232
3	Start	CheckInventory[2]	Order[0].inventory	1	product_descr = "A 342 L"	233
3	not_ok	CheckInventory[2]	Order[0].inventory	1	-	349
4	Start	RejectOrder[0]	Order[0].reject	1	product_descr = "A 342 L", customer_id = 78	350
4	done	RejectOrder[0]	Order[0].reject	1	-	380
1	done	Order[0]	-	-	-	381

Figura 3.6: *História de uma instância. Fonte: (Kradolfer e Geppert, 1999)*

`Order` (Figura 3.5) cujo pedido foi rejeitado. Cada evento corresponde a duas linhas da história, uma quando o evento é iniciado e outra quando o evento é finalizado. Por exemplo, a segunda e terceira linha correspondem e execução do *sub-workflow* `enter`. Os atributos armazenados em cada evento são basicamente os parâmetros de entrada/saída e o tempo de chegada. A execução de um conjunto de *workflows* W é representado pela história H que, por sua vez, é uma ordem parcial $H = (E, \angle)$, em que E é o conjunto de eventos invocados pelos *workflows* em W e $\angle$ é uma relação binária transitiva e irreflexiva tal que $\forall e_1, e_2 \in E\,(timestamp(e_1) < timestamp(e_2)) \Leftrightarrow (e_1 \angle e_2)$. A história de um *workflow* específico é uma projeção em H, contendo os eventos gerados pelo *workflow* e por seus *sub-workflows*. O histórico também é utilizado para verificar a consistência comportamental de uma instância de acordo com os invariantes.

As operações de alteração disponibilizadas pela abordagem são estruturadas pela taxonomia a seguir.

1. Adicionar uma nova versão de um tipo complexo de *workflow*
   1.1. Alterações na interface do *workflow*
      1.1.1. Alterações nas definições de parâmetros
         1.1.1.1. Adicionar um novo parâmetro
         1.1.1.2. Remover um parâmetro
      1.1.2. Alterações em um estado final
         1.1.2.1. Adicionar um novo estado final
         1.1.2.2. Remover um estado final
   1.2. Alterações no corpo do *workflow*
      1.2.1. Alterações na declaração de um *sub-workflow*
         1.2.1.1. Adicionar um *sub-workflow*
         1.2.1.2. Remover um *sub-workflow*
         1.2.1.3. Modificar a condição de entrada de um *sub-workflow*
      1.2.2. Alterações na declaração do fluxo de dados
         1.2.2.1. Adicionar um fluxo de dados
         1.2.2.2. Remover um fluxo de dados
      1.2.3. Alterações na condição final do *workflow*
         1.2.3.1. Modificar uma condição final
2. Remover uma versão de um tipo complexo de *workflow*
3. Adicionar um novo tipo de *workflow*
4. Remover um tipo de *workflow*

Uma alteração no esquema é caracterizada pela execução atômica de uma ou mais operações da taxonomia. As operações da categoria 1 alteram diversas características do fluxo de controle e de dados, criando uma nova versão do esquema. Portanto, elas somente podem ser aplicadas a uma versão já existente. A operação da categoria 2 exclui uma versão do esquema somente se não houverem instâncias em execução. A operação representada na categoria 3 cria um novo *workflow*, ou seja, a primeira versão do processo de negócio. A operação da categoria 4 remove todas as versões de um *workflow* somente quando não existirem instâncias em execução.

Essas alterações levam a abordagem a enfrentar dois problemas: (i) garantir que o esquema se mantenha correto após a modificação e (ii) migrar as instâncias em andamento para o modelo modificado. Para solucionar o primeiro problema, a abordagem garante que o esquema se mantém estruturalmente consistente se, ao aplicar as operações de alteração, os invariantes não são violados. Caso contrário, a modificação é rejeitada.

Para solucionar o segundo problema, os autores reduzem o escopo ao considerar somente migrações que não requerem o desfazimento de passos já realizados. Assim, a instância deve ser migrada para a nova versão do esquema sem que nenhuma informação seja perdida. Considere o cenário em que a instância i foi inicializada no esquema S, executou até o tempo ct e precisa ser transferida para a nova versão S'. De modo geral, i pode ser migrada para o novo esquema S', se sua história ocorrida em S até ct for comportamentalmente consistente em S'.

Uma estratégia simples para verificar a consistência comportamental seria examinar se cada evento contido na história de i em S é compatível com S'. Entretanto, pode ser ineficiente para as instâncias cujo histórico seja muito grande. Os autores propõem uma estratégia alternativa que analisa somente as operações que alteram o esquema. Cada operação de alteração especificada na taxonomia possui uma *condição de migração* na forma de invariante. Assim, uma instância i pode ser migrada corretamente de S para S', se ela satisfaz todas as condições de migração impostas pelas operações de alteração que foram aplicadas em S para gerar S'. Por exemplo, ao executar a operação que remove um *sub-workflow* (1.2.1.2), a condição de migração é: Uma instância pode ser migrada de S para S', se o *sub-workflow* a ser removido não tiver sido iniciado. Mais detalhes sobre as condições de migração podem ser encontrados em (Kradolfer e Geppert, 1999).

TRAMs apresenta um arcabouço que proporciona a evolução em esquemas de *workflow* baseado no versionamento dos esquemas e na migração das instâncias em andamento. A declaração do fluxo de controle e do fluxo de dados é realizada de forma declarativa e a correção é garantida por meio de invariantes. A abordagem disponibiliza um amplo conjunto de operações para alterar o esquema e vincula a cada operação um invariante para que a migração ocorra de forma segura. Outra característica dessa abordagem, é o fato das instâncias permanecerem na versão antiga quando não puderem ser migradas.

3.2.3 ADEPT / WSM-net

Hensinger *et al.* (2000) apresentam o projeto denominado ADEPT (*Application Development Based on Encapsulated Premodeled Process Templates*) que teve início em 1994 na Universidade de Ulm na Alemanha com o objetivo de desenvolver um WfMS orientado a processo e adaptativo para cenários de aplicações complexas. O ADEPT é um dos mais importantes projetos de pesquisa que visam prover flexibilidade para os WfMSs. Esta seção destaca duas importantes contribuições do projeto ADEPT relacionadas a proposta. Primeiramente, é descrito o $\text{ADEPT}_{\text{flex}}$ que apresenta um conjunto de operações que apoiam a modificação de instâncias em execução ao mesmo tempo que preservam a corretude e consistência (Reichert e Dadam, 1997). Posteriormente, será descrito como tratar a adaptação de instâncias em execução usando uma *Well-Structured-Marking Nets* (WSM-Nets) apoiada por um histórico de execução (Rinderle *et al.*, 2004a).

$\text{ADEPT}_{\text{flex}}$

$\text{ADEPT}_{\text{flex}}$ é um conjunto de operações que apoiam a modificação de instâncias em execução em um modelo de *workflow* baseado em grafo que apresenta uma base formal em sua sintaxe e em sua semântica operacional (Reichert e Dadam, 1997, 1998). Embora o $\text{ADEPT}_{\text{flex}}$ disponibilize operações apenas para instâncias individuais, a maioria dos conceitos apresentados podem ser aplicados para evolução de esquemas e adaptação de instâncias em modelos baseados em grafos.

No ADEPT, o modelo de *workflow* apresenta estruturas de controle simétricas. Sequência de tarefas, ramificações (com diferentes semânticas para divisão e junção), e estruturas de repetição são especificadas como blocos simétricos com vértices de início e fim bem definidos. Esse blocos podem ser aleatóriamente aninhados, mas não é permitido sobreposição, isto é, o alinhamento deve ser regular. Além disso, o modelo também provê sincronização de tarefas de ramos paralelos.

Formalmente, um esquema de *workflow* P é representado como:

$$P = \langle N, E, S, D, DF \rangle, \tag{3.1}$$

em que as tarefas são abstraídas como um conjunto de vértices N (de diferentes tipos NT) e as dependências de controle entre as tarefas são especificadas por meio de arestas representadas pelo conjunto E. Há dois vértices especiais que representam o início (NT = STARTFLOW) e o fim (NT = ENDFLOW) do modelo P. Todos os demais vértices em N devem ser precedidos e sucedidos por, pelo menos, um vértice. As arestas de controle são do tipo ET = CONTROL_E. Há também arestas de sincronização ET = SOFT_SYNC e ET = STRICT_SYNC. Uma aresta $\langle n_1, n_2 \rangle \in E$ cujo NT = SOFT_SYNC é usada para especificar uma dependência de retardo, isto é, n_2 pode somente ser executado se n_1 for executado ou se ele não pode ser mais executado. Por outro lado, uma aresta $\langle n_1, n_2 \rangle \in E$ cujo NT = STRICT_SYNC requer que n_1 tenha sido executado para, então, permitir a execução de n_2. Há também três tipos de ramificações no fluxo de controle: paralelo (AND-split/AND-join), condicional (OR-split/OR-join), paralelo com seleção final (AND-split/OR-join).

O fluxo de dados é mapeado pelos conjuntos D e DF. O conjunto D representa os elementos de dados associado ao modelo P. Uma ligação de dado $df \in DF$ pode ser descrita pela 4-tupla:

$$df = \langle d^{df}, n^{df}, par^{df}, access\_mode^{df} \rangle \tag{3.2}$$

no qual $d^{df} \in D$ é um elemento de dado associado ao modelo P, $n^{df} \in N \cup S$ é uma tarefa ou serviço, $par^{df} \in PAR(n^{df})$ é um parâmetro para n^{df} e $access\_mode^{df} \in \{read, write\}$ indicando se df representa uma leitura ou uma escrita. É possível que existam diferenças no formato e na representação dos dados de tal modo que parâmetros de saída de uma tarefa sejam incompatíveis com os parâmetros de entrada das tarefas subsequentes. A fim de anemizar esse problema e evitar ajustes de dados internos, cada tarefa n pode ser associada a um serviço auxiliar $s \in S_n \subset S$. Um serviço auxiliar $s \in S_n := S_n^{prec} \cup S_n^{succ}$ é disparado quando n é iniciado ($s \in S^{prec}$) ou quando ele é finalizado ($s \in S^{succ}$).

Cada instância é controlada por um grafo de execução do *workflow* tal como o modelo apresentado em 3.1. Além disso, o estado da instância é definido pelas marcações nos vértices e nas arestas, pelos valores armazenados nos elementos de dados. O estado atual de uma tarefa n é descrito pela marcação NS^n em seu vértice ($NS \in$ {NOT_ACTIVATED, ACTIVATED, RUNNING, COMPLETED, FAILED, SKIPPED }), o número de execuções anteriores e dados relevantes as execuções. Finalmente, cada aresta e de um grafo de execução está em um dos estados $ES^e \in$ {NOT_SIGNALED, FALSE_SIGNALED, TRUE_SIGNALED}. Quando uma instância é criada, o grafo de execução é inicializado. Todos os vértices são configurados para NOT_ACTIVATED e todas arestas são marcadas como NOT_SIGNALED. Além disso, os dados de entrada são armazenados nos elementos de dados correspondentes.

O ADEPT estabelece algumas regras para garantir a consistência estrutural do esquema. Basicamente, em (Reichert e Dadam, 1998), existem apenas duas regras relacionadas ao fluxo de controle que levam em consideração um grafo acíclico livre de arestas de sincronização. São elas:

- Cada tarefa $n \in N$ deve ser alcançável a partir do vértice inicial. Isto é, existe uma sequência válida de eventos que conduzem a marcação inicial do grafo de execução até a ativação de n.

- A partir de cada tarefa $n \in N$ executada deve ser possível atingir o estado final. Isto é, existe uma sequência válida de eventos que conduzem a partir de qualquer estado atual marcado até estado final.

Por outro lado, existem outras duas regras bem definidas que garantem a corretude do fluxo de dados do esquema. A seguir essas duas regras são detalhadas.

Regra 1: Seja $P = (N, E, S, D, DF)$ um esquema de *workflow*. Para cada $n \in N \cup S$, considere V^n o conjunto de tarefas e serviços cujo elementos precedem n no fluxo de controle e que são completados antes de n ser inicializado. Para $n \in N \cup S$ e $d \in D$ é, então, requerido:

$$reads(n, d) \Rightarrow (\forall V \in V^n : \exists n^* \in V : writes(n^*, d))$$

O predicado $reads(n, d)$ (respectivamente $writes(n, d)$) expressa que um parâmetro de entrada (respectivamente um parâmetro de saída) de $n \in N \cup S$ é conectado em d por uma ligação de dados $df \in DF$. Informalmente, a Regra 1 determina que para n efetuar um leitura em um elemento de dado d deve existir uma tarefa ou serviço executado anteriormente que realizou uma operação de escrita sobre d.

Regra 2: Seja $P = (N, E, S, D, DF)$ um esquema de *workflow*. Para $n_1, n_2 \in N$ com $writes(n_1, d)$ e $writes(n_2, d)$ é necessário que:

1. $(n_1 \in \overline{succ}(n_2) \vee n_2 \in \overline{succ}(n_1))$ ou
   $(\exists n_s \in N : n_s \text{ is OR-join} \wedge n_s \in M : \overline{succ_c}(n_1) \cap \overline{succ_c}(n_2)) \wedge \forall n \in M, n \neq n_s : n \in \overline{succ_c}(n_s))$
2. $n_2 \in \overline{succ}(n_1) \Rightarrow \exists n_3 \in (\overline{succ}(n_1) \cap \overline{pred}(n_2)) \cup \{n_2\}$ com $reads(n_3, d)$

São omitidas operações de escrita dos elementos do conjunto S. Informalmente, o item (1) afirma que se n_1 e n_2 escrevem no mesmo elemento de dado e não sucedem um ao outro no fluxo de controle, então n_1 e n_2 devem pertencer a diferentes ramos com um OR-join n_s. Portanto, tarefas de diferentes ramos de processamento paralelo (com AND-join) não podem escrever em um mesmo elemento de dado, exceto se eles forem serializados com o uso de uma aresta de sincronização. O item (2) tem como objetivo evitar escrita perdida (*lost update*). Portanto, a Regra 2 determina que deve existir uma ordem prévia para escrita sobre elementos de dados evitando que tarefas que são executadas em paralelo concorram sobre um mesmo elemento de dado.

$\text{ADEPT}_{\text{flex}}$ apresenta um conjunto de operações para alterar o grafo de execução de uma instância de forma que mantenham a correção do esquema inicial. De forma geral, Reichert e Dadam (1998) propõem:

- adicionar novas tarefas ou blocos de tarefas em qualquer ponto durante a execução da instância;
- sincronizar a execução de uma tarefa inserida com a execução de outras tarefas a partir do grafo de execução;
- inserir tarefas em regiões que ainda não foram executadas;
- mapear novos ou existentes elementos de dados para as tarefas; e

- remover ou ignorar tarefas ainda não executadas.

Com esse objetivo, ADEPT$_{\text{flex}}$ fornece uma operação única e genérica que é completa no sentido de ser capaz de realizar cada possível forma de inserção. Tal operação permite adicionar uma nova tarefa X, juntamente com serviços auxiliares S_x, elementos de dados D_x e ligações de dados DF_x em um grafo de execução ao sincronizar X entre dois conjuntos de vértices M_{before} e M_{after}. A Figura 3.7 ilustra a operação *insert* ao inserir um nova tarefa x na instância Π. Essa operação transforma o grafo de execução (N, E, S, D, DF) e os estados (NS, ES) de uma instância em um novo grafo (N', E', S', D', DF') e novos estados (NS', ES'). Essa transformação deve resultar em um novo esquema da instância sintaticamente correto e com estados legais. Com essa finalidade, a seguir é apresentado um pseudo-algoritmo com os passos para alteração do grafo de execução.

- Encontrar o sub-grafo $B \subseteq (N, E)$ mínimo que contém todos vértices de $M_{before} \cup M_{after}$. Seja n_{begin} e n_{end} os vértices inicial e final de B;
- Inserir um vértice n_1 (AND-split) como predecessor direto de n_{begin} e inserir um vértice n_2 (AND-join) como sucessor direto de n_{end}. Ambos n_1 e n_2 são tarefas nulas sem qualquer ação associada, sendo apenas utilizadas como entrada e saída de B;
- Inserir um novo vértice x como uma ramificação entre os vértices n_1 e n_2 e sincronizar x com as tarefas de M_{before} e M_{after}. Isto é, para cada tarefa $b \in M_{before} \neg\{STARTFLOW\}$ adicionar uma aresta de sincronização $\langle b, X\rangle$ e para cada tarefa $a \in M_{after} \neg\{ENDFLOW\}$ adicionar uma aresta de sincronização $\langle X, a\rangle$. Ambas arestas são ET = SOFT_SYNC;
- Aplicar as regras de redução e reavaliar o estado dos vértices e arestas.

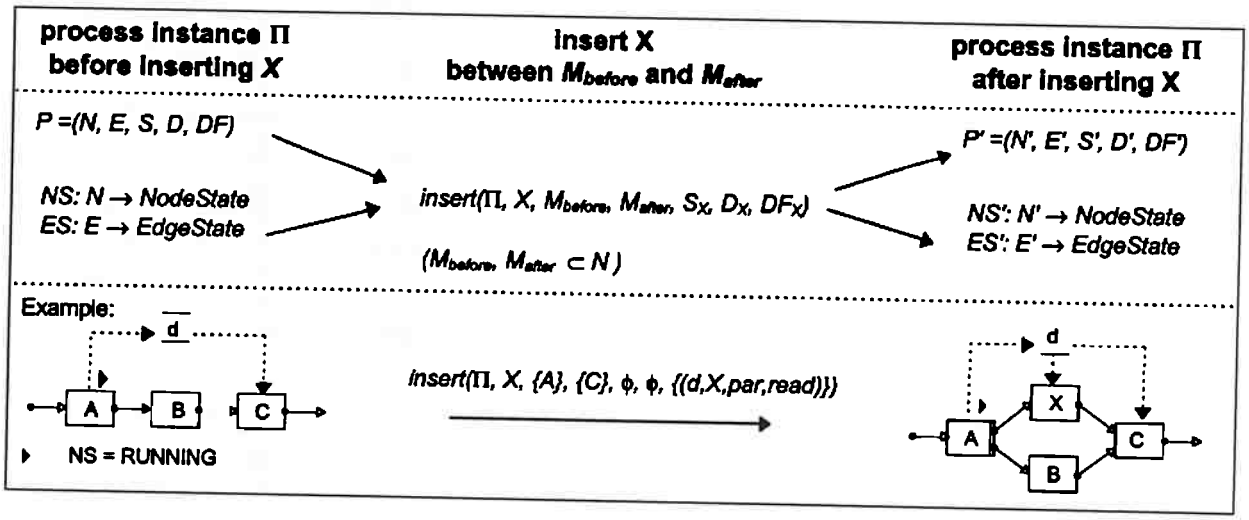


Figura 3.7: *Inserção de uma nova tarefa x associada aos serviços auxiliares S_x, elementos de dados D_x e ligações de dados DF_x entre dois conjuntos de tarefas M_{before} e M_{after}. Fonte: Reichert e Dadam (1998)*

Para garantir que a alteração produza um grafo sintaticamente correto, algumas restrições devem ser consideradas. Primeiro, para todo $n_a \in M_{before}$, $n_b \in M_{after}$, o vértice n_a deve preceder n_b no fluxo de controle. Segundo, a região compreendida pelos vértices M_{before} e M_{after} pode somente conter estruturas de repetição completas.

A Figura 3.8 exemplifica todos os passos do algoritmo que executa a alteração no grafo de execução. O exemplo ilustra a inserção de uma tarefa X entre dois conjuntos de vértices $\{C, D\}$ e

$\{F\}$. O primeiro passo é encontrar o bloco mínimo que contém todos os vértices C, D e F (veja Figura 3.8b). A seguir, um vértice n_1 (AND-split) é inserido representando uma tarefa nula entre o predecessor A e o vértice B inicial do bloco mínimo. Similarmente um vértice n_2 (AND-join) é inserido. Logo após, X é inserido como uma ramificação entre n_1 e n_2 e sincronizado como os vértices C, D e F pelas arestas de sincronização $\langle C, X\rangle$, $\langle D, X\rangle$ e $\langle X, F\rangle$ (veja Figura 3.8c). Finalmente, as regras de redução são aplicadas resultando no novo esquema da instância na Figura 3.8d. Todas as regras de redução podem ser encontradas em Reichert e Dadam (1998) e são omitidas nesta proposta.

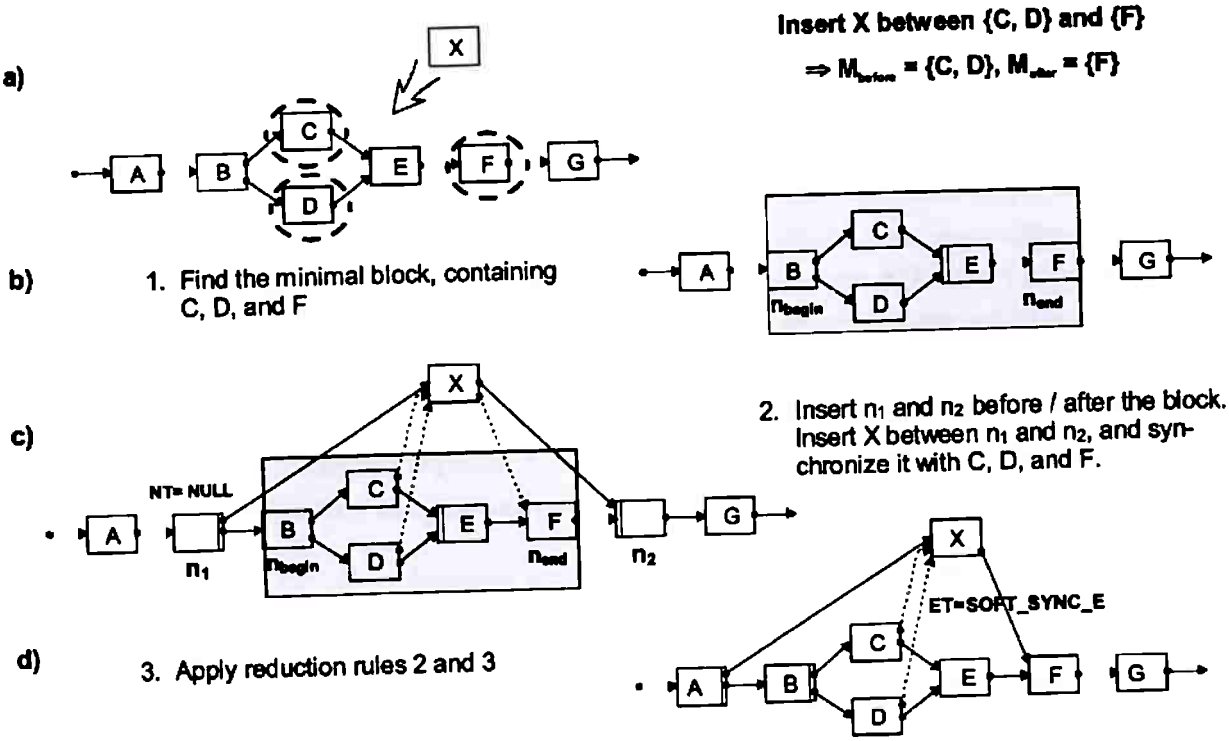


Figura 3.8: *Inserção de uma nova tarefa entre dois conjuntos de vértices. Fonte: Reichert e Dadam (1998)*

Um outro aspecto crucial para que a alteração do grafo de execução seja bem-sucedida é o estado de execução da instância alterada. Para garantir que a nova tarefa seja executada, é determinante que todos os elementos de M_{after} estejam em um dos estados NOT_ACTIVATED ou ACTIVATED. Caso contrário, a execução da nova tarefa pode ser incompatível com as demais tarefas. Por outro lado, é permitido que os elementos de M_{before} estejam em qualquer estado de execução.

Finalmente, é necessário analisar e ajustar o fluxo de dados a fim de respeitar as regras estabelecidas anteriormente. Geralmente, a inserção de uma tarefa não acarreta apenas na alteração do fluxo de controle e do estado de execução da instância. Essa alteração pode também requerer que os conjuntos D, S e DF sejam estendidos. Todos os parâmetros de tarefa inserida devem ser fornecidos antes que ela seja executada conforme estabelecido pela Regra 1. Uma simples forma para alcançar isso seria requisitar ao usuário os dados de entrada. Isso pode ser feito por meio de um serviço predecessor s cujo parâmetros de saída são os parâmetros de entrada da nova tarefa. Uma outra forma seria mapear os parâmetros da nova tarefa para os elementos de dados já existentes em D. Entretanto, esse mapeamento não pode violar a Regra 1. Para isso, suponha os elementos de dados $C_X \subseteq D$ sejam requisitados como parâmetros de entrada de uma tarefa X. De acordo com a Regra 1, temos que:

$$C_X = \{d \in D | \forall V \in V^X : \exists n^* \in V : writes(n^*, d)\}$$

Similarmente, deve-se garantir que nenhum parâmetro de saída da tarefa X viole o descrito pela Regra 2. Isto é, não pode existir uma tarefa de outra ramificação que possua um parâmetro de saída sobre o mesmo elemento de dado. Em (Reichert e Dadam, 1998) não há uma forma algorítmica de garantir a Regra 2.

Tarefas podem ser removidas ou ignoradas quando as condições para sua execução tornam-se desnecessárias.

A remoção de uma tarefa X de uma instância em execução é somente possível se X esta em um dos estados `ACTIVATED` ou `NOT_ACTIVATED`. As tarefas nos estados `RUNNING`, `COMPLETED`, `FAILED`, ou `SKIPPED` não podem ser removidas.

Em relação ao ajuste do fluxo de controle, a operação de remoção é realizada ao substituir a tarefa por um tarefa nula. Essa estratégia pode ser tratada de maneira simples e eficaz, uma vez que a tarefa removida e suas arestas associadas são mantidas na estrutura.

Quando uma tarefa X é removida, seus serviços auxiliares e as suas ligações de dados também devem ser removidos dos conjuntos S e DF, respectivamente. Isso pode conduzir a uma violação da Regra 1 uma vez que os parâmetros de entrada das tarefas sucessoras a X podem ser afetados. Reichert e Dadam (1998) propõem algumas alternativas para contornar esse problema. Dessa forma, suponha N^* o conjunto de tarefas sucessoras a X cujo para parâmetros de entrada não são corretamente providos devido à remoção de X. As alternativas para tratar esse problema são:

- Remoção das tarefas do conjunto N^* (remoção em cascata);
- Inserção de uma tarefa X^{prox} no fluxo que de controle que assuma as ligações de dados das tarefas removidas e deve ser completada antes de qualquer tarefa do conjunto N^* ser inicializada;
- Inserção de serviços auxiliares $S_n^{prec}, n \in N^*$ para fornecer as ligações de dados removidas junto com a tarefa X;
- Cancelamento da operação de remoção.

O ADEPT$_{flex}$ é um conjunto de operações usadas para alterar o esquema de instância individuais. Note que as regras sobre o fluxo de controle e de dados garantem que as instâncias alteradas não apresentam inconsistência no andamento da execução. Além disso, as alterações são aplicadas em partes do esquema que ainda não foram executados. Embora isso faça as instâncias tirarem proveito das alterações, a mesma ideia aplicada ao problema de migração de instâncias para um novo esquema é muito rígida. Tal rigidez faz com que muitas instâncias fossem impedidas de efetuar a migração com segurança. A seguir, será discutido uma abordagem que estende ADEPT$_{flex}$ ao utilizar a história de execução como alternativa para minimizar os efeitos colaterais da evolução de esquemas.

WSM-Nets

Rinderle *et al.* (2004a) argumentam que a migração de instâncias para um novo esquema é correta e segura se o novo esquema possui a capacidade de representar todos os passos e ações realizados pelas instâncias. Dessa forma, para atingir esse objetivo é essencial manter a história

de execução das instâncias. Entretanto, Rinderle *et al.* (2004a) ressaltam que analisar o histórico completo da instância para realizar a migração conduz a dois importantes efeitos colaterais em modelos que permitem estruturas de repetição: (i) o tamanho do histórico devido a estrutura de repetição pode tornar-se muito grande levando a problemas de desempenho; e (ii) se a alteração é realizada dentro da estrutura de repetição, apenas as instâncias que não completaram a primeira iteração são aptas para migrarem ao novo esquema. Para superar esses problemas, Rinderle *et al.* (2004a) propõem que, na presença de estruturas de repetição, a abordagem armazene apenas a informação sobre a execução da última iteração das instâncias em execução. Dessa forma, esse histórico reduzido é usado para avaliar se uma instância pode ser migrada de forma correta e segura. Formalmente, essa característica é dada pela propriedade de complacência descrita pelo Axioma 3.1.

Axioma 3.1 (Propriedade de complacência). *Seja I uma instância do esquema S com história de execução $\mathcal{H}$ e história de execução reduzida $\mathcal{H}_{red}$. Assuma também que uma alteração Δ transforma o esquema S em um esquema correto S'. Então, I é dito ser complacente com S' se e somente se:*

- *$\mathcal{H}_{red}$ pode ser reprodutível por S', isto é, $\mathcal{H}_{red}$ poderia ser produzido por uma instância executada de acordo com S'; e*
- *cada atividade iniciada ou finalizada da instância deveria ler e cada atividade finalizada deveria escrever os mesmos valores no esquema S'.*

A seguir será descrito detalhes do modelo de *workflow*, da forma de execução, da forma de manter a história reduzida de execução e como essa história pode apoiar os critérios para migração de instâncias.

Um WSM-Net (Well-Structured-Marking Net) é um modelo de *workflow* baseado em grafos semelhante ao modelo usado pelo ADEPT$_{\text{flex}}$ com algumas variações no fluxo de controle e de dados. A Figura 3.9a ilustra uma representação gráfica de um esquema com algumas atividades e elementos de dados. Formalmente, o esquema do fluxo de controle S pode ser representado por:

$$S = (N, D, NT, CtrlE, SyncE, LoopE, DP, EC) \tag{3.3}$$

em que:

- N é o conjunto de atividades e D é o conjunto de elementos de dados;
- $NT : N \mapsto \{\texttt{StartFlow}, \texttt{EndFlow}, \texttt{Activity}, \texttt{AndSplit}, \texttt{AndJoin}, \texttt{XOrSplit}, \texttt{XOrJoin}, \texttt{StartLoop}, \texttt{EndLoop}\}$. NT atribui um tipo para cada vértice do WSM-Net;
- $CtrlE \subset N \times N$ representa as dependências de controle normais entre as atividades;
- $SyncE \subset N \times N$ representa a relação de precedência entre atividades de ramos paralelos;
- $LoopE \subset N \times N$ representa as arestas de retorno das estruturas de repetição;
- $DP : N \mapsto D \cup \{\texttt{UNDEFINED}\}$. Suponha um vértice n tal que $NT(n) = \texttt{XOrSplit}$. $DP(n)$ corresponde aos elementos de dados que indicam qual o ramo a ser selecionado. Obtém-se $DP(m) = \texttt{UNDEFINED}$ para cada vértice m com $NT(m) \neq \texttt{XOrSplit}$;

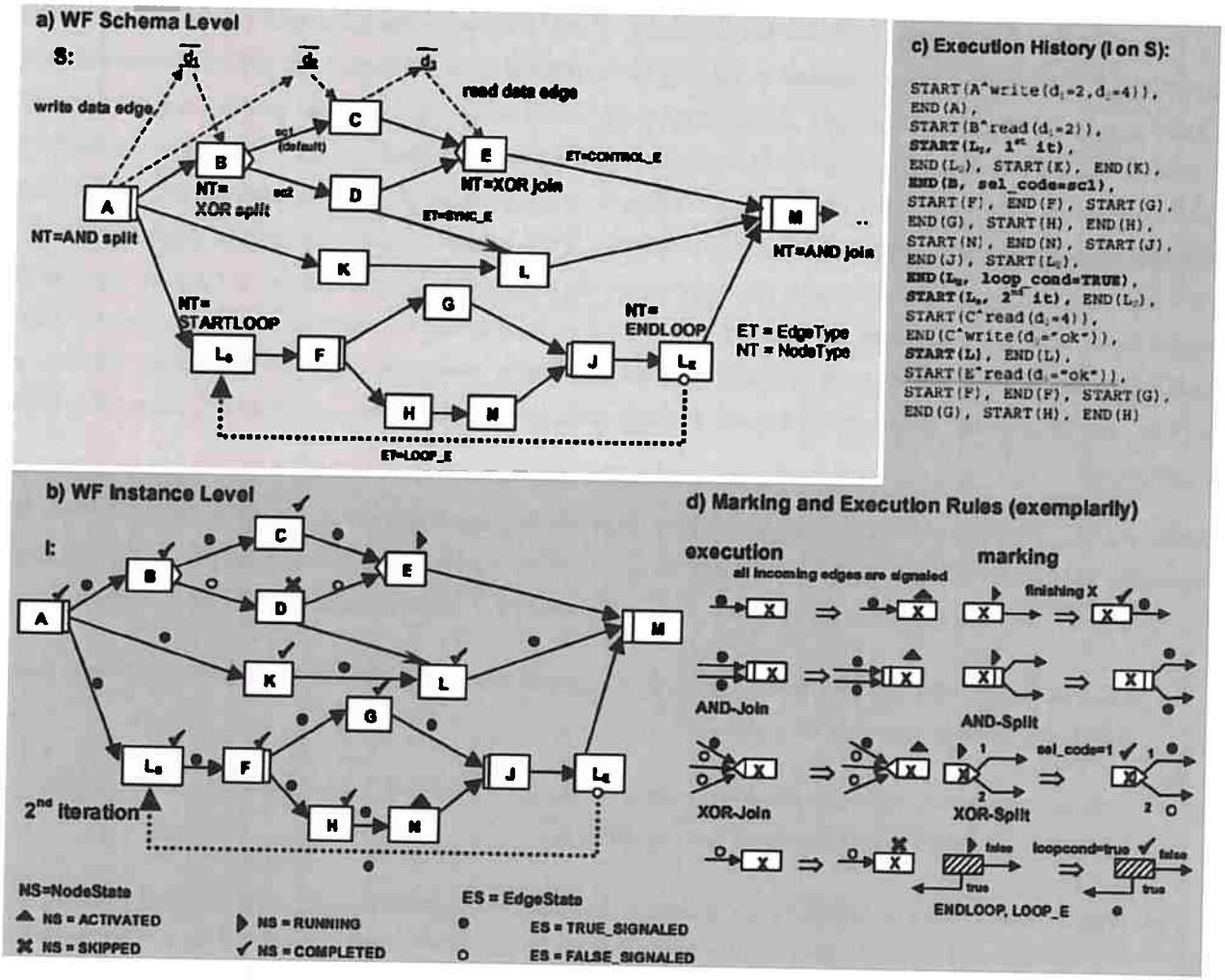


Figura 3.9: *Modelagem e execução de* workflows *usando WSM-Nets. Fonte: (Rinderle et al., 2004a)*

- $EC : CtrlE \mapsto EdgeCode \cup \{\texttt{UNDEFINED}\}$. $EC(e)$ atribui o código de seleção para as arestas de controle do XOR-Split.

Informalmente, o esquema de fluxo de controle é correto se e somente se:

- $S_{fwd} = (N, CtrlE, SyncE)$ é um grafo acíclico, isto é, o uso de arestas de controle e de sincronização não resultam em ciclos indesejados que conduzem para *deadlocks*;
- para cada vértice de divisão (`AndSplit`, `XOrSplit`) há um único vértice de junção (`AndJoin`, `XOrJoin`) e para cada vértice de início da estrutura de repetição (`StartLoop`) existe um equivalente de final de estrutura de repetição (`EndLoop`);
- S respeita o conceito de blocos estruturados. Blocos de controle (sequências, ramificações e estruturas de repetição) podem ser aninhados mas não pode existir sobreposição.

A execução de um WSM-Net também é semelhante ao descrito na seção do $\text{ADEPT}_{\text{flex}}$. As marcações também seguem as regras de execução determinadas pelos construtores (veja Figura 3.9c e 3.9d). A principal contribuição do WSM-Net é manutenção de um histórico reduzido de execução das instâncias que é usado como base para manter a propriedade de complacência durante a migração das instâncias.

Uma instância I baseada no WSM-net pode ser definida pela tupla $(S, M^S, Val^S, \mathcal{H})$ tal que:

- $S = (N, D, NT, CtrlE, SyncE, LoopE, DP, EC)$ como visto em 3.3 denota o esquema da instância I .
- $M^S = (NS^S, ES^S)$ descrevem vértices e arestas de marcações I:
  - $NS^S : N \mapsto$ {`NotActivated`, `Activated`, `Running`, `Completed`, `Skipped`}
  - $ES^S : (CtrlE \cup SyncE \cup LoopE) \mapsto$ {`NotSignaled`, `TrueSignaled`, `FalseSignaled`}
- Val^S é uma função sobre D. Para cada elemento de dado $d \in D$, a função reflete o valor atual para a instância I ou, quando d ainda não foi escrito, o valor é `UNDEFINED`.
- $\mathcal{H} = \langle e_0, \ldots, e_k \rangle$ é a história de execução de I na qual $\langle e_0, \ldots, e_k \rangle$ denotam os eventos de início e fim das execuções das atividades. Além disso, são registrados os elementos de dados lidos quando as atividades são iniciadas e os elementos de dados escritos quando as atividades são finalizadas. A Figura 3.9c apresenta um exemplo de história de execução que indica o início e o fim das atividades.

Como dito anteriormente, Rinderle *et al.* (2004a) propõem uma história de execução que, na presença de estruturas de repetição, armazena apenas a última iteração. Dessa forma, considere uma instância I com uma história de execução completa $\mathcal{H} = \langle e_0, ..., e_k \rangle$, em que $e_0, ..., e_k$ denotam os eventos inicial e final de todas as execuções das atividades relacionadas a I. A história de execução reduzida $\mathcal{H}_{red}$ é idêntica a $\mathcal{H}$ na ausência de estruturas de repetição. Caso contrário, $\mathcal{H}_{red}$ é derivado de $\mathcal{H}$, mantendo informações apenas da última iteração (estrutura de repetição completa) ou da iteração atual (estrutura de repetição em execução).

Uma vez descrito o modelo de *workflow* e a forma de execução, é possível entender as condições formais para manter a propriedade de complacência quando novas atividades, arestas de controle, arestas de sincronização são inseridas no esquema. O Teorema 3.1 descreve essas restrições ao considerar operações que alteram o fluxo de controle. A prova formal para o teorema pode ser encontrada em (Rinderle *et al.*, 2004a).

Teorema 3.1 (Alteração no fluxo de controle). *Seja* $S = (N, D, NT, CtrlE, SyncE, LoopE, DP, EC)$ *um esquema de* workflow *correto e* I *uma instância de* S *com história de execução reduzida* $\mathcal{H}_{red}$ *e com marcações* $M^s = (NS, ES)$. *Além disso, assuma que uma operação de alteração* Δ *transforma* S *em um esquema correto* $S' = (N', D', NT', CtrlE', SyncE', LoopE', DP', EC')$.

a) Δ *insere uma atividade* n_{insert} *(com arestas de controle e de sincronização) em* S. *Então:*

I *é complacente com* $S' \iff$

$$\forall n \in \{x \in N \mid n_{insert} \rightarrow x \in (CtrlE' \cup SyncE')\} :$$
$$NS(n) \in \{\texttt{NOT\_ACTIVATED}, \texttt{ACTIVATED}, \texttt{SKIPPED}\} \wedge$$

n_{insert} *é inserido em um ramo já ignorado de um XOR.*

b) Δ *insere uma aresta de controle* $n_{src} \rightarrow n_{dest}$ *em* S. *Então:*

I *é complacente com* $S' \iff NS(n_{dest}) \in \{\texttt{NOT\_ACTIVATED}, \texttt{ACTIVATED}, \texttt{SKIPPED}\}$

c) Δ *insere uma aresta de sincronização* $n_{src} \rightarrow n_{dest}$ *em* S *(*n_{src} *e* n_{dest} *são paralelos). Então:*

I *é complacente com* $S' \iff$

$[NS(n_{dest}) \in \{\texttt{NOT\_ACTIVATED}, \texttt{ACTIVATED}, \texttt{SKIPPED}\}] \vee$

$[NS(n_{src}) = \texttt{COMPLETED} \wedge NS(n_{dest}) \in \{\texttt{RUNNING}, \texttt{COMPLETED}\}$ *com*

$\forall n \in N_{critical}$ *com* $NS(n) \neq \texttt{SKIPPED}$:

$\exists e_i = \texttt{START}(n_{dest}), e_j = \texttt{END}(n) \in \mathcal{H}_{red}$ *com* $j < i$),

em que $N_{critical} = (c\_pred^*(S, n_{src}) \neg c\_pred^*(S, n_{dest}))$

and $c\_pred^*(S, n))$ *denota todos os predecessores diretos ou indiretos de* n *em* S

considerando arestas de controle]

d) Δ *remove uma atividade* n_{del} *de* S *(incluindo arestas de controle para re-ligação). Então:*

I *é complacente com* $S' \iff$

$NS(n_{del}) \in \{\texttt{NOT\_ACTIVATED}, \texttt{ACTIVATED}, \texttt{SKIPPED}\}$

e) Δ *remove uma aresta de controle ou sincronização* $n_{src} \rightarrow n_{dest}$ *de* S. *Então:*

I *é complacente com* S'

O fluxo de dados entre as atividades é projetado ao conectar parâmetros de entrada e saída das atividades com variáveis globais (elementos de dados). Cada parâmetro de entrada de uma atividade é mapeado exatamente a um elemento de dado por uma aresta de leitura e, similarmente, cada parâmetro de saída da atividade é conectado a um elemento de dado por uma aresta de escrita. A Figura 3.9a ilustra algumas arestas de leitura e de escrita.

Além das condições descritas pelo Teorema 3.1, é necessário que as escritas e leituras das atividades sejam representadas no novo esquema para assegurar a propriedade de complacência descrita pelo Axioma 3.1. No caso de alterações no esquema de fluxo de dados, a propriedade de complacência de uma instância pode ser facilmente verificada baseada nas seguintes condições:

Teorema 3.2 (Alteração no fluxo de dados). *Seja* S = (N, D, NT, $CtrlE$, $SyncE$, $LoopE$, DP, EC) *um esquema de fluxo de controle correto com esquema de fluxo de dados* DFS *e* I *uma instância de* S *com história de execução reduzida* $\mathcal{H}_{red}$ *e com marcações* $M^s = (NS, ES)$. *Além disso, assuma que uma operação de alteração* Δ *transforma* S *em um esquema correto* S' = (N', D', NT', $CtrlE'$, $SyncE'$, $LoopE'$, DP', EC') *com esquema de fluxo de dados* DFS'

a) Δ *insere um elemento de dado* d *em* DFS. *Então:*

I *é complacente com* S'

b) Δ *remove um elemento de dado* d *de* DFS. *Então:*

I *é complacente com* $S' \iff$

não há arestas de leitura/escrita em d *de uma atividade com marcação* RUNNING *ou* COMPLETED.

c) Δ *insere ou remove uma aresta de leitura* $d \rightarrow n$. *Então,*

I *é complacente com* $S' \iff NS(n) \in \{\texttt{NOT\_ACTIVATED}, \texttt{ACTIVATED}, \texttt{SKIPPED}\}$.

d) Δ *insere ou remove uma aresta de escrita* $n \rightarrow d$. *Então:*

I *é complacente com* $S' \iff NS(n) \neq \texttt{COMPLETED}$.

Em resumo, esta seção descreve os trabalhos do projeto ADEPT. Esses trabalhos apresentam importantes abordagens que contribuem com o cenários dos *workflows* adaptativos. Primeiramente, o $\text{ADEPT}_{\text{flex}}$ disponibiliza um conjunto de operações que atuam sobre as instâncias individualmente. Mais tarde, o WSM-net apresenta um alternativa baseada em histórias de execução para verificar a complacência das instâncias quando uma alteração é realizada no esquema. Em ambos trabalhos o modelo de *workflow* é baseado em grafo. Tradicionalmente os modelos baseados em grafo classificados como estruturas rígidos que apresentam dificuldades para adaptação e evolução. Esse argumento é reforçado em (Aalst *et al.*, 2009), quando os autores afirmam que abordagens tradicionais dependem da estruturação explícita dos processos e, por essa razão, apresentam dificuldade para atender alterações realizadas no processo de negócio.

3.2.4 Declare

O Declare (Aalst *et al.*, 2009) é um PAIS baseado em restrições que fornece múltiplas linguagens declarativas. Ao invés de explicitamente definir em tempo de projeto a ordem de execução das atividades como nos modelos formais, o Declare usa restrições para implicitamente determinar as possíveis ordens de execução. No Declare, qualquer ordem de execução que não viola uma restrição é permitida. Os modelos baseados em restrições foram discutidos na Seção 2.3.2.

O PMS do Declare é representado por três componentes básicos: *designer*, *framework* e *worklist*. O *designer* é usado para projetar e verificar os modelos de processos usando restrições, tais como aquelas apresentadas na Seção 2.3.2. No Declare, cada restrição possui um nome único, uma representação gráfica e uma especificação formal de sua semântica em Lógica Temporal Linear (LTL, do inglês *Linear Temporal Logic*).

O *framework* é responsável pela execução das instâncias dos processos. Para cada instância, o Declare cria um autômato para o conjunto de fórmulas LTL considerando todas as restrições da instância. Esse autômato é conhecido como autômato da instância e é usado para:

- Orientar a execução. Uma atividade que é executada pela instância dispara uma ou mais transições que causam a mudança no estado do autômato;
- Determinar as atividades aptas para execução. Somente atividades que podem ser disparadas no estado atual do autômato são habilitadas;
- Determinar o estado da instância. Se o estado atual do autômato é aceitável, então a instância está satisfeita; se o estado atual do autômato não é aceitável, então a instância está em um estado temporariamente violado.

Além do autômato da instância, o *framework* também mantém um autômato individual para cada restrição da instância. Esses autômatos são usados para determinar os estados das restrições. Por exemplo, caso o estado atual do autômato é aceito, então a restrição é dita estar satisfeita; caso contrário, a restrição é dita estar temporariamente violada.

Enquanto o *framework* gerencia a execução de todas as instâncias, cada usuário usa seu *worklist* para acessar as instâncias ativas. O autômato da instância é usado para indicar quais atividades estão habilitadas para serem executadas pelo usuário.

O Declare propõem um mecanismo de identificação em tempo de projeto de dois tipos de erros no esquema de um processo: (i) identificação das atividades que nunca podem ser executadas e (ii)

detecção de conflito entre as restrições, isto é, detecção quando não há uma história de execução que satisfaça todas as restrições obrigatórias. Ambos erros são facilmente identificados na geração de um autômato que reúne todas as restrições do modelo.

O Declare permite que alterações do modelo do processo sejam executadas enquanto instâncias estejam em execução. Basicamente, uma alteração no esquema consiste de uma adição, remoção ou atualização de uma ou mais restrições. Uma alteração no esquema pode afetar instâncias específicas ou pode ser aplicada para todas instâncias em execução. Em ambos os casos, o autômato da instância é usado para verificar se a alteração é aplicável a instância. Desta forma, uma mudança é aplicável se, e somente se, a sua história de execução pode ser repetida sobre o autômato gerado para o novo modelo. Para isso, a cada nova alteração do modelo um novo autômato deve ser gerado e o histórico de execução deve ser percorrido.

3.3 Considerações Finais

Neste capítulo, abordagens que usam *snapshots* e o histórico de execução na avaliação da correção da migração de uma instância para o novo esquema foram brevemente discutidas.

As abordagens baseadas em *snapshots* apresentam soluções mais restritivas uma vez que o principal desafio é adaptar as marcações das instâncias após a alteração do esquema, sem o conhecimento da sua execução anterior (Rinderle *et al.*, 2004a). Por outro lado, abordagens que analisam o histórico de execução para migrar as instâncias garantem que o esquema alterado represente perfeitamente a instância migrada.

WIDE (Casati *et al.*, 1998) é um dos pioneiros a tratar o problema da evolução e adaptação de instâncias em execução. A principal contribuição é propor um conjunto mínimo de operações básicas para transformar um esquema S correto em outro esquema S' correto. Para migrar instâncias para S', pela primeira vez, a propriedade da complacência foi sugerida.

TRAM (Kradolfer e Geppert, 1999; Kradolfer *et al.*, 1999) apresentam a ideia de versionamento de esquema. Para gerenciar com eficiência uma migração da instância, os autores propõem a definição de condições de migração para cada operação de mudança da mesma forma que as regras complacência. Somente se as condições forem satisfeitas é que as instâncias podem ser migradas.

Os pesquisadores Peter Dadam e Manfred Reichert da Ulm University são líderes de um dos grupos de pesquisa mais atuante na área de adaptação de instâncias em PAIS. O grupo tem apresentado diversas contribuições de grande relevância. Entre elas, neste capítulo, destacam-se o ADEPT (Reichert e Dadam, 1997) e o WSM-Nets (Rinderle *et al.*, 2004a). Ambos trabalhos apresentam estratégias de alteração do esquema de *workflow* ao mesmo tempo que garante corretamente a adaptação das instâncias. O WSM-Nets apresenta uma solução ao analisar tanto as marcações como o histórico de execução para identificar as instâncias comportamentalmente consistente.

O Declare (Aalst *et al.*, 2009) é uma abordagem essencialmente flexível ao permitir qualquer execução desde que não seja expressamente proibida por meio de restrições. Entretanto, no mecanismo de adaptação proposto não há uma relação entre o antigo esquema ao novo esquema uma vez que toda alteração de esquema conduz para a criação de um novo autômato.

Como discutido no capítulo anterior, abordagens formais como Redes de Petri e Grafos são rígidas uma vez requerem que todos os possíveis caminhos de execução sejam definidos em tempo de projeto. Nesses sistemas, a evolução e adaptação das instâncias também são restritivas ao cons-

trutores apresentados pelas abordagens. Em contrapartida, os modelos declarativos baseados em restrições são extremamente flexíveis uma vez que permitem qualquer caminho de execução desde que não seja expressamente proibido. Entretanto, o mecanismo de verificação das regras proposto em Declare ainda não é eficiente ao tratar da evolução e adaptação.

Capítulo 4

Mecanismo Automático para Migração de Instâncias Não Complacentes

Neste capítulo, será apresentado a ideia intuitiva do mecanismo para propagar as modificações do esquema para todas as instâncias em execução. Tradicionalmente, dado uma modificação no esquema do processo, há três alternativas para tratar instâncias em execução (Casati *et al.*, 1998; Sadiq *et al.*, 2000; Song e Jacobsen, 2016):

Aborto: instâncias são abortadas e, novamente, criadas de acordo com a nova versão do esquema do processo. Como consequência, as atividades já concluídas das instâncias em execução são descartadas ou compensadas.

Descarga: instâncias em execução avançam e finalizam sua execução de acordo com o esquema do processo de origem, enquanto as novas instâncias são iniciadas na nova versão do esquema do processo. Consequentemente, as instâncias em execução não aproveitam das modificações do esquema.

Migração: instâncias em execução são migradas para a nova versão do esquema do processo. No entanto, elas não podem ser migradas de forma indiscriminada, porque podem estar em diferentes etapas de execução do processo. Portanto, uma instância em execução pode somente ser migrada para o novo esquema de processo, se seu estado de execução é complacente com o novo esquema de processo. Para evitar erros ou inconsistências, as instâncias não complacentes não são migradas e continuam suas execuções no esquema processo de origem.

Este trabalho observa principalmente cenários que requerem a propagação imediata das modificações do esquema para todas as instâncias, ou seja, cenários em que instâncias em execução devem obrigatoriamente ser migradas para o novo esquema. Mais especificamente, este trabalho apresenta uma alternativa para tratar as instâncias não complacentes com o novo esquema a fim de permitir a migração imediata.

Para avaliação do estado de complacência das instâncias em execução na abordagem WED-*flow*, será adotado um importante e bem conhecido critério de corretude baseado na verificação da história de execução das instâncias candidatas a migração, como apresentado em (Casati *et al.* , 1998; Rinderle *et al.*, 2003). Intuitivamente, esse critério define que uma instância WED-*flow* WF I em execução no esquema WED-*flow* WF W pode ser corretamente migrada para o novo

esquema WED-*flow* WF W', se a execução de WF I pode ser "simulada" em WF W'. Esse critério é formalmente definido a seguir.

> *Seja um esquema WED-flow WF W e um instância WED-flow WF I em WF W com histórico de execução H_I^W. Seja WF W transformado em um outro esquema WED-flow WF W' pela modificação Δ. Então, WF I é complacente com WF W' se H_I^W pode ser reproduzido em WF $W' =$ WF $W + \Delta$ na mesma ordem.*

Uma instância não-complacente é identificada quando pelo menos um de seus passos não pode ser reproduzido no novo esquema. O principal problema ocorre quando a modificação do esquema afeta direta ou indiretamente alguma atividade que já tenha sido concluída na instância em execução. Para exemplificar isso, a Figura 4.1 ilustra um esquema do processo dividido em três regiões distintas: (i) a região antes da modificação; (ii) a região afetada pela modificação; e (iii) a região após da modificação.

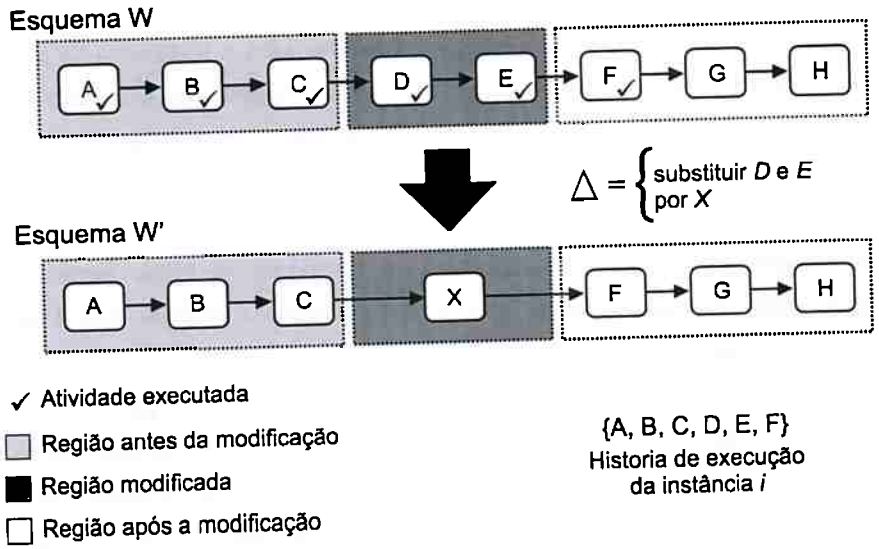


Figura 4.1: *Regiões afetadas pela modificação.*

A região *antes* da modificação é uma região segura. Isso significa que as instâncias em execução dentro dessa região podem sempre ser migradas. A *região modificada* é aquela cujas instância são diretamente afetada pela modificação. As instâncias dentro dessa região têm uma alta probabilidade de não ser complacentes com o novo esquema. A região *após* a modificação é indiretamente afetada pela modificação. As instâncias em execução que tiveram suas atividades concluídas afetadas pela modificação são propensas a não ser complacente com as novas especificações. Algumas abordagens consideram as instâncias da região (iii) complacentes com as novas especificações, se elas são capazes de avançar e finalizar suas execuções de acordo com o novo esquema (Aalst, 2001). Entretanto, mesmo se a instância é capaz de prosseguir de acordo com o novo esquema, nenhuma análise é feita sobre o seu passado. Suponha, por exemplo, a história de execução da instância WF I na Figura 4.1. A instância WF I dentro da região após a modificação é capaz de prosseguir e finalizar sua execução no novo esquema. Quando WF I finalizar no WF esquema W', o histórico de execução produzido pela WF I será inconsistente com o esquema WF W'. Apesar da instância WF I não aproveitar das modificações, a sua história de execução pode ser diferente do esquema de processo. Portanto, o principal desafio é como conduzir uma instância não complacente para um ponto complacente a fim de permitir a sua migração.

A partir dessa perspectiva, neste trabalho é proposto um mecanismo baseado em *partial rollback* para a migração automática de instâncias não complacentes a partir dessas regiões imprevisíveis. Além disso, a estratégia proposta garante a consistência entre o histórico de execução da instância e o esquema do processo. O *rollback* é um conceito bem conhecido introduzido primeiramente no modelo clássico de transação e, posteriormente, adaptado para o gerenciamento de transações de longa duração por meio da definição de transações de compensação (Elmasri e Navathe, 1999; García-Molina e Salem, 1987). Mais especificamente, em Transações de Longa Duração (LLT, do inglês *Long-Lived Transactions*), uma ação de compensação t^{-1} desfaz semanticamente os efeitos realizados por uma transação t (García-Molina e Salem, 1987). No contexto dos PAIS, bem como em LLT, as compensações podem ser usadas para desfazer o processo para um ponto anterior. Assim, algumas abordagens têm sugerido o *partial rollback* (por meio de compensação) para o tratamento de instâncias não complacentes no contexto PAIS adaptativos (Sadiq *et al.*, 2000). A estratégia consiste basicamente em compensar as atividades concluídas até que a instância atinja um ponto em complacência com a nova especificação e possa ser migrada para o novo esquema do processo. O *partial rollback* fornece uma solução automática e genérica para a migração de todas as instâncias não complacentes. Por exemplo, no cenário ilustrado pela Figura 4.1, a instância WF I pode ser recuperada para um ponto complacente (atividade C) para depois ser migrada e prosseguir em WF W'. Entretanto, um efeito colateral óbvio dessa estratégia é a perda de trabalho resultante das atividades de compensação. Essa perda de trabalho pode ser razoável para alguns cenários, mas também pode ser inviável para alguns outros cenários. Do ponto de vista teórico, o *partial rollback* propõe uma solução simples, mas de um ponto de vista prático, a solução não é uma tarefa trivial, porque requer o apoio de propriedades transacionais.

A estratégia proposta neste trabalho é flexibilizar o *partial rollback* para reduzir os seus efeitos colaterais e aumentar a capacidade de migração das instâncias. A migração automática consiste de três principais etapas. Para descrever essas etapas, assuma que o esquema WED-*flow* W é transformado em W' por uma modificação Δ. Além disso, assuma que uma instância em execução I em W é candidata para ser migrada para W'. As três etapas da migração são:

(a). **Avaliar o estado de complacência da instância.** O mecanismo aplica o critério de corretude descrito acima para avaliar se I é complacente com W'. Uma instância em execução I é dita ser complacente com W' se H_I^W pode ser reproduzido em W'. Neste caso, I é migrado para W' como descrito na etapa (c). Caso contrário, essa etapa identifica o último ponto no qual I é complacente com W', descrito como WED-*state* $\mathbf{s_{cp}}$.

(b). **Conduzir a instância para o ponto complacente.** Quando I não é complacente com W', o mecanismo recupera seu estado de execução para uma região segura. A instância I é recuperada para o WED-*state* $\mathbf{s_{cp}}$ encontrado na etapa (a). Essa recuperação *backward* é realizada ao disparar um encadeamento de WED-*compensations* até que o WED-*state* $\mathbf{s_{cp}}$ seja alcançado. Esse encadeamento de WED-*compensations* desfaz semanticamente os efeitos das atividades que foram executadas após o ponto de complacência $\mathbf{s_{cp}}$. A partir desse ponto, a instância I pode ser migrada em segurança.

(c). **Migrar a instância em execução.** Uma vez que, pelas etapas anteriores, é garantido que I é complacente com W', nesta etapa a instância pode ser transferida para o novo esquema sem causar erros ou inconsistências. Após ser transferida, I continua com sua execução em W'.

Para reduzir os efeitos colaterais do *partial rollback*, é proposto neste trabalho, o reuso dos estados de dados, levando em consideração alguns aspectos semânticos. Do ponto de vista pragmático, o mecanismo proposto permite que um WED-*state* gerado por uma instância I no esquema de origem W seja reusado no esquema de destino W'. Do ponto de vista de recuperação transacional, o reuso de estados de dados na abordagem WED-*flow* são passos de recuperação *forward* que desviam a instância do seu caminho normal. A Figura 4.2 ilustra intuitivamente a migração de uma instância I do esquema W para o esquema W' com o reuso de estados de dados. Após a migração, a instância I procede sua execução em W' e o reuso de WED-*states* é representado pelos passos de recuperação *forward* para o seu esquema original W. Como representado na Figura 4.2, o reuso de WED-*states* (ou seja, passos de recuperação *forward*) somente é habilitado em alguns pontos específicos da execução da instância. Esses pontos denotam as equivalências entre as atividades dos esquemas W e W'. Há dois tipos de equivalência: **equivalências imediatas** e **equivalências declaradas**. Uma atividade que não foi afetada pela mudança é considerada imediatamente equivalente no esquema de origem e de destino. Uma atividade que é diretamente afetada pela mudança pode ser declarada como equivalente a uma outra atividade, desde que produza resultados semanticamente equivalentes.

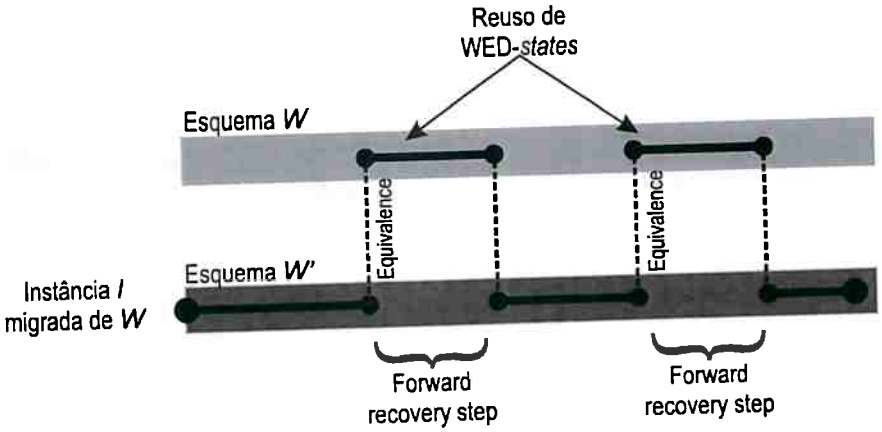


Figura 4.2: *Reutilização dos WED-states.*

Na abordagem WED-*flow*, esses conceitos são definidos a seguir.

Definição 1. ***Equivalência de WED-transitions.*** *Uma WED-transition t_i pertencente ao WED-flow WF W é equivalente a uma WED-transition t_j pertencente ao WED-flow WF W' se ambas produzem WED-states equivalentes.*

A equivalência de WED-*states* foi intuitivamente descrita em (Silva *et al.*, 2012) e pode ser formalmente descrita como:

Definição 2. ***Equivalência de WED-states.*** *Seja os WED-states s_n e s_m, o conjunto de WED-triggers G_n habilitadas por s_n e o conjunto de WED-triggers G_m habilitadas por s_m. Os WED-states s_n e s_m são equivalentes se $G_n = G_m$. Em outras palavras, s_n e s_m são equivalentes quando eles disparam as mesmas WED-transitions.*

Definição 3. ***Reuso de WED-states.*** *Um WED-state s_n pertencente ao WED-flow W pode ser reusado no WED-flow W' se o WED-state s_{n-1} dispara WED-transitions equivalentes em W e W'.*

Para exemplificar o reuso dos WED-*states* nas subseções seguintes é proposto duas modificações no esquema do processo de programa de pós-graduação ilustrado na Figura 4.3. Na primeira modificação, a atividade *Entrega de Artigo* é removida e as atividades *Prova Escrita* e *Apresentação Oral* são alteradas para executarem sequencialmente. Na segunda alteração, a atividade *Disciplinas de Núcleo Geral* é substituída por *Disciplinas de Núcleo Específico*. A seguir, será apresentado como o mecanismo é capaz de reusar WED-*states* sempre que identifica equivalências entre as atividades dos esquemas origem e destino.

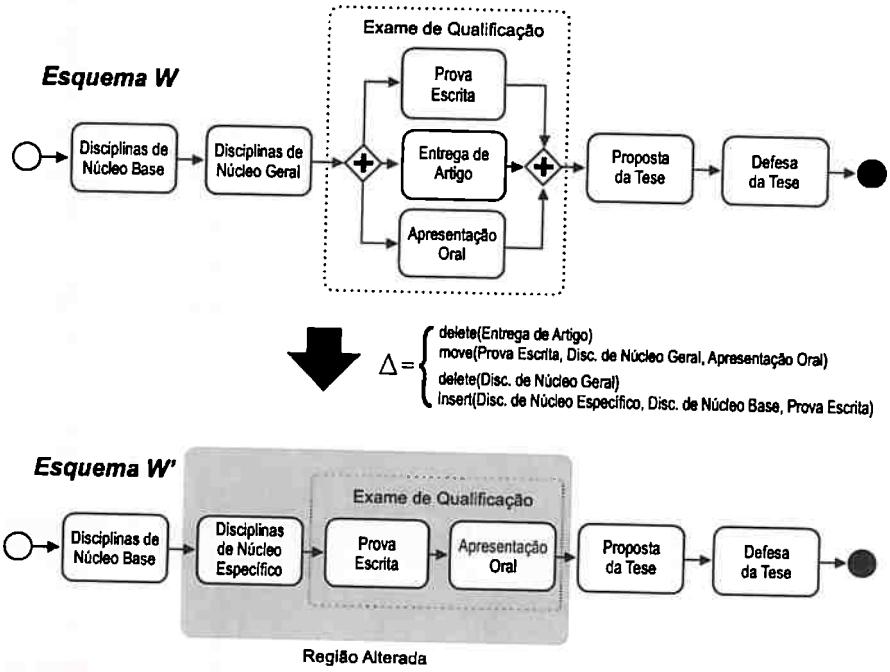


Figura 4.3: *Modificação no processo.*

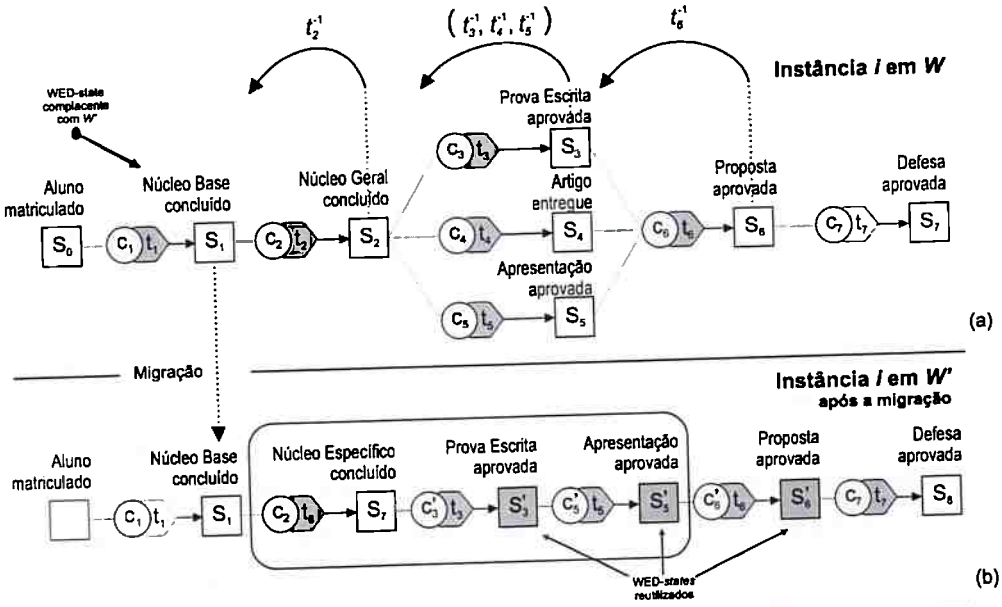
4.1 Reuso habilitado por equivalências imediatas

Alguns cenários de processo de negócio introduzem aspectos semânticos que deveriam ser considerados no problema de migração das instância em execução. Por exemplo, assuma o esquema processo ilustrado na Figura ?? que descreve um programa de pós-graduação. Se um aluno tiver sido aprovado no exame de qualificação, a sua aprovação deveria ser considerada durante a migração para um novo esquema uma vez que o exame de qualificação não é afetado pela modificação (ou seja, eles são equivalentes em ambos os esquemas). Por causa disso, não faz sentido que o aluno seja obrigado a realizar outro exame de qualificação no novo esquema. Em outras palavras, existem atividades quando realizadas permitem que os seus resultados possam ser reutilizados na migração para um novo esquema de processo. Neste trabalho, observa-se esses aspectos semânticos como um exemplo concreto para tratar a migração de instâncias não complacentes. Mais especificamente, a abordagem proposta reutiliza WED-*states* quando há uma equivalência imediata entre atividades em ambos os esquemas. Esse reuso, por sua vez, reduz a perda de trabalho e, consequentemente, relaxa os efeitos colaterais do *partial rollback*.

Para ilustrar o reuso do WED-*state*, observa-se uma instância em execução WF I iniciada a partir do esquema WED-*flow* WF W de acordo com a Figura 4.4(a). Essa instância em execução representa um estudante que completou a atividade *Proposta da Tese* e esta esperando para defender

sua tese. Nesse momento, o esquema WF W é modificado para o esquema WF W' como ilustrado na Figura 4.3. A modificação remove a atividade *Entrega de Artigo* e altera a ordem de execução entre as atividades *Prova Escrita* e *Apresentação Oral* que são agora executadas sequencialmente. De acordo com o critério de complacência, a instância WF I não é complacente com o esquema modificado W' e é excluída da migração porque sua história de execução não pode ser reproduzida em W'.

A Figura 4.4 ilustra o mecanismo para tratar da instância em execução I a fim de permitir sua migração para W'. Detalhes sobre os algoritmos são apresentados na Seção ??. A migração automática segue as três etapas descritas acima: (a) Avaliar o estado de complacência da instância; (b) Conduzir a instância para o ponto complacente; e (c) Migrar a instância em execução.



Histórico de execução da instância *i* em *W*

	Matrícula		Disciplinas de Núcleo Base		Disciplinas de Núcleo Geral		Exame de Qualificação			Proposta da Tese		Defesa da Tese
	ID Aluno	Status	Créditos	Status	Créditos	Status	Prova Escrita	Entrega do Artigo	Apresentação Oral	Entrega do Documento	Defesa Oral	Defesa Oral
S_0	1	matriculado	0	-	0	-	-	-	-	-	-	-
S_1	1	matriculado	5	aprovado	0	-	-	-	-	-	-	-
S_2	1	matriculado	5	aprovado	2	aprovado	-	-	-	-	-	-
S_3	1	matriculado	5	aprovado	2	aprovado	aprovado	-	-	-	-	-
S_4	1	matriculado	5	aprovado	2	aprovado	aprovado	entregue	-	-	-	-
S_5	1	matriculado	5	aprovado	2	aprovado	aprovado	entregue	aprovado	-	-	-
S_6	1	matriculado	5	aprovado	2	aprovado	aprovado	entregue	aprovado	entregue	aprovado	-

Histórico de execução da instância *i* em *W'*

	Matrícula		Disciplinas de Núcleo Base		Disciplinas de Núcleo Específico		Exame de Qualificação		Proposta da Tese		Defesa da Tese
	ID Aluno	Status	Créditos	Status	Créditos	Status	Prova Escrita	Apresentação Oral	Entrega do Documento	Defesa Oral	Defesa Oral
S_1	1	matriculado	5	aprovado	0	-	-	-	-	-	-
S_7	1	matriculado	5	aprovado	3	aprovado	-	-	-	-	-
S'_3	1	matriculado	5	aprovado	3	aprovado	aprovado	-	-	-	-
S'_5	1	matriculado	5	aprovado	3	aprovado	aprovado	aprovado	-	-	-
S'_6	1	matriculado	5	aprovado	3	aprovado	aprovado	aprovado	entregue	aprovado	-
S_8	1	matriculado	5	aprovado	3	aprovado	aprovado	aprovado	entregue	aprovado	aprovado

(c)

Figura 4.4: *Migração com reuso de WED-state.*

Na **etapa (a)**, o mecanismo avalia a instância I como não complacente com W' e returna o último estado complacente $s_{cp} = s_1$. Na **etapa (b)**, I é recuperado até o WED-*state* s_1 ao

executar um encadeamento de WED-*compensations* t_6^{-1}, t_5^{-1}, t_4^{-1}, t_3^{-1} e t_2^{-1}. Esta cadeia de WED-*compensations* é encontrada automaticamente pela gerenciador de recuperação do WED-*flow* baseado na história de execução de I. Para permitir o reuso dos WED-*states* de I, os resultados das compensações não são fisicamente armazenados. Para realizar isso, é aplicado uma estratégia similar a noção de *ação silenciosa* τ introduzida em (Aalst e Basten, 2002). Originalmente, uma ação rotulada como *ação silenciosa* não tem seus resultados observáveis. Na abordagem WED-*flow*, uma *ação silenciosa* é denotada por uma WED-*compensation* cujos resultados não são armazenados. Portanto, esses passos conduzem I para um ponto complacente (WED-*state* $\mathbf{s_1}$), sem desfazer fisicamente as atividades finalizadas, e permitindo que os WED-*states* $\mathbf{s_2}$, $\mathbf{s_3}$, $\mathbf{s_4}$, $\mathbf{s_5}$ e $\mathbf{s_6}$ sejam reusados in W'. Nesse ponto, a instância I é complacente com W' e é capaz de continuar com sua execução em W'.

Na **etapa (c)**, a instância I é migrada para W' pela criação do WED-*state* $\mathbf{s_7}$. Após migração, I continua sua execução a partir de $\mathbf{s_7}$ em W'. A WED-*condition* c_8 é satisfeita pelo WED-*state* $\mathbf{s_7}$ e dispara a WED-*transition* t_8 (atividade *Disciplinas de Núcleo Específico*). A WED-*transition* t_8 é o resultado da substituição da atividade *Disciplinas de Núcleo Geral*. Após o estudante completar o *Disciplinas de Núcleo Específico*, o WED-*state* $\mathbf{s_8}$ é gerado. O WED-*state* $\mathbf{s_8}$ habilita o estudante a submeter o Exame de Qualificação. No novo esquema W', esse exame tem um novo significado. O estudante é submetido a uma avaliação escrita (WED-*transition* t_3) seguida por uma apresentação oral (WED-*transition* t_5). Uma vez que o estudante já tenha sido aprovado nesses exames no esquema original W, os seus resultados podem ser reusados em W'. Na abordagem WED-*flow*, o reuso do WED-*state* é possível quando a mesma WED-*transition* é disparada em ambos os esquemas. Quando o WED-*state* $\mathbf{s_8}$ é gerado pela WED-*transition* t_8, este estado dispara t_3 em ambos esquemas. Como um resultado, o WED-*state* $\mathbf{s_3}$ (produzido por t_3 em W) pode ser reusado em W'. O mecanismo cria o WED-*state* $\mathbf{s_9}$ em W' copiando somente os valores dos atributos que são produzidos por t_3 (*aprovado* no atributo *Prova Escrita*). O WED-*state* $\mathbf{s_9}$ reusado habilita a execução da WED-*transition* t_5. O mecanismo identifica que os resultados produzidos por t_5 podem ser reusados em W' e cria o WED-*state* $\mathbf{s_{10}}$ em W' copiando os valores dos atributos produzidos por t_5. Em seguida, o WED-*state* $\mathbf{s_{10}}$ habilita a execução da WED-*transition* t_6 cujos resultados são reusados ao criar o WED-*state* $\mathbf{s_{11}}$. Finalmente, o WED-*state* $\mathbf{s_{11}}$ habilita a execução de t_7. Uma vez que t_7 não foi executado em W, ele deve ser disparado para produzir o WED-*state* final $\mathbf{s_{12}}$. Na próxima seção, será apresentado uma outra característica do mecanismo proposto para aumentar a flexibilidade ao tratar a migração de instâncias não complacentes.

4.2 Reuso habilitado por equivalências declaradas

A equivalência declarada entre WED-*transitions* é um outro aspecto semântico importante que aumenta a flexibilidade ao tratar de instâncias não complacentes. Por exemplo, quando o projetista propõe uma mudança de uma atividade para outra, provavelmente, há uma equivalência semântica entre elas que pode ser considerada durante a migração das instâncias. Suponha que a alteração proposta na Figura 4.3 em que a modificação Δ substitui a atividade *Disciplinas de Núcleo Geral* pela atividade *Disciplinas de Núcleo Específico*.

Com essa modificação, a atividade *Disciplinas de Núcleo Específico* é imposta para todas as instâncias. De acordo com o critério de corretude, qualquer instância em execução que tenha concluído

a atividade *Disciplinas de Núcleo Geral* é incapaz de migrar para o novo esquema. No entanto, o programa de pós-graduação decide que para estudantes que já tenham concluído a atividade *Disciplinas de Núcleo Geral* não é obrigatório a realização da atividade *Disciplinas de Núcleo Específico* no novo esquema. Em outras palavras, o projetista da modificação pode expressar que as atividades *Disciplinas de Núcleo Geral* e *Disciplinas de Núcleo Específico* são equivalentes para as instâncias em execução. Uma vez que essas atividades são declaradas equivalentes, o WED-*state* gerado por *Disciplinas de Núcleo Geral* pode ser reutilizado. Esta equivalência semântica é considerada neste trabalho para fornecer mais flexibilidade na migração de instâncias não complacentes.

Além de permitir o reuso de WED-*states*, a equivalência declarada pelo projetista também flexibiliza a avaliação do estado de complacência aumentando o número de instâncias migráveis sem a necessidade de recuperação *backward*. Como mencionado previamente, uma instância em execução é dita ser complacente se sua história de execução pode ser reproduzida no novo esquema. Portanto, a verificação das equivalências declaradas são incluídas na avaliação do estado de complacência. Por exemplo, assuma um esquema W' cujas atividades são $\{A, X, C, D\}$ e uma instância em execução I que tenha executado as atividades $\{A, B, C, D\}$. Considerando o critério de corretude original, a instância I não é complacente com W'. Na estratégia proposta, pela declaração de equivalência entre as atividades X e B, a instância I é avaliada como complacente com W'.

Para avaliar o estado de complacência das instâncias em execução usando a equivalência de atividades, assuma uma instância em execução I inicializada no esquema W de acordo com a Figura 4.5(a). Essa instância em execução representa um estudante que completou a atividade *Prova Escrita* do Exame de Qualificação. Nesse ponto, o esquema WF W é modificado para o esquema WF W' como ilustrado na Figura 4.3. Além disso, para as instâncias em execução, o projetista determina que a atividade *Disciplinas de Núcleo Geral* de W é equivalente à atividade *Disciplinas de Núcleo Específico* de W' ($t_2 \Leftrightarrow t_8$). Em outras palavras, o projetista assume que tais atividades produzem resultados semanticamente equivalentes (WED-*states* equivalentes de acordo com a Definição 2). A instância I tem os seguintes WED-*states* $\{\mathtt{s}_0, \mathtt{s}_1, \mathtt{s}_2, \mathtt{s}_3\}$ e executou as seguintes WED-*transitions* $\{t_1, t_2, t_3\}$. De acordo com critério de corretude e as equivalências entre as atividades *Disciplinas de Núcleo Geral* (t_2) e *Disciplinas de Núcleo Específico* (t_8), a instância I é complacente com W' e pode ser migrada porque seu histórico de execução pode ser reproduzido em W'. A migração de I para W' é alcançada pela transferência do WED-*state* atual ($\mathtt{s}_3$) da instância para o esquema W'. Esta transferência resulta em a novo WED-*state* ($\mathtt{s}_4$) em W'. Após a migração, a instância I continua a partir do WED-*state* $\mathtt{s}_4$ e finaliza sua execução.

Resumidamente, o projetista da modificação do esquema pode especificar equivalências entre WED-*transitions* que são consideradas durante a migração das instâncias em execução. O mecanismo proposto usa essas equivalências declaradas para reusar WED-*states* e flexibilizar o critério de migração aumentando o número de instâncias complacentes. Estratégias de flexibilização do critério de migração têm sido introduzidas a fim de aumentar o número de instâncias migráveis (Rinderle-Ma *et al.*, 2008; Song *et al.*, 2013).

Nesta seção foi descrito como a abordagem WED-*flow* é capaz de apoiar a evolução de processos em cenários no qual a migração imediata para todas as instâncias em execução é um requisito fundamental. A flexibilização da estratégia tradicional de *partial rollback* reduz os seus efeitos colaterais ao permitir o reuso dos estados de dados das instâncias em execução. Além disso, foi também descrito como as equivalências declaradas entre as WED-*transitions* flexibiliza o critério

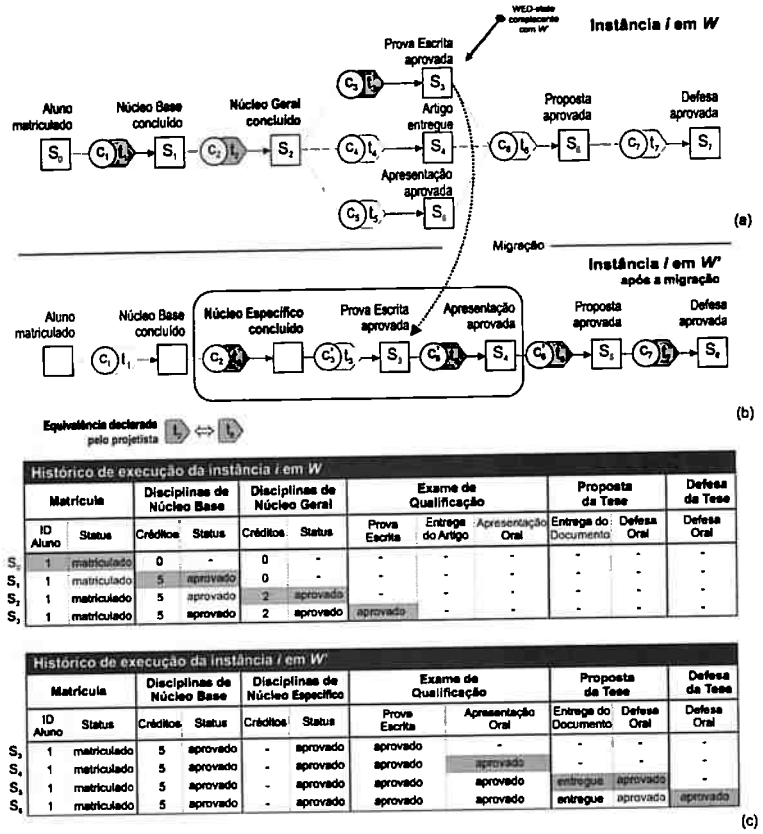


Figura 4.5: *Equivalência de WED-transitions.*

de complacência. Os algoritmos da migração automática são apresentados na próxima seção.

4.3 Algoritmo para migração automática

Nesta seção, será descrito como aumentar automaticamente a migração de instâncias não complacentes na evolução de esquemas. Esse mecanismo consiste em três etapas conforme descrito acima. Primeiramente, será descrito o algoritmo para a migração automática de todas as instâncias em execução e, em seguida, os algoritmos referentes as três etapas será descrito em detalhes.

A Figura 4.6 ilustra o procedimento para migração automática das instâncias em execução no qual os parâmetros de entrada são os esquemas W e W'. Na linha 2, o procedimento encontra todas as instâncias em execução no esquema W. Entre as linhas 3-8, um laço de repetição itera sobre todas as instâncias em execução a fim de migra-las para o esquema W'. Para cada instância em execução, a condição de migração é verificada na linha 4. A função `COMPLIANCE-STATE` retorna $\emptyset$ quando a instância i é complacente com o esquema W'. Caso contrário, a função retorna o último estado de dado (s_{cp}) no qual a instância i é complacente com o esquema W'. Se a instância em execução i não é complacente com W' então a recuperação *backward* conduz a instância para o ponto de complacência s_{cp} na linha 6. Na linha 7, a instância é migrada para o esquema W'. Os algoritmos para as três etapas envolvendo a migração das instâncias em execução serão descritos em mais detalhes a seguir.

Na etapa (a), a função `COMPLIANCE-STATE` descrita na Figura 4.7 avalia se uma instância em execução é complacente com o esquema modificado baseado no critério de complacência. A função tem três parâmetros de entrada: a instância em execução i e os esquemas W e W'. Primeiramente,

```
1: procedure MIGRATE-RUNNING-INSTANCES(W, W')
2:     instances ← RUNNING-INSTANCES(W)
3:     for all i ∈ instances do
4:         s_cp ← COMPLIANCE-STATE(i, W')                    ▷ Etapa (a)
5:         if s_cp ≠ ∅
6:             BACKWARD-RECOVERY(i, s_cp)                    ▷ Etapa (b)
7:         MIGRATE(i, W, W')                                 ▷ Etapa (c)
8:     end for
9: end procedure
```

Figura 4.6: *Procedimento para migração automática das instâncias em execução*

o procedimento obtém o histórico de execução H_i^W da instância i em W na linha 2. Em seguida, o procedimento invoca uma função nativa do WED-*flow* para obter todas os WED-*states* da instância i a partir do histórico de execução H_i^W. A função WED-STATES retorna uma fila de WED-*states* no qual a cabeça da fila é o primeiro WED-*state* gerado por i. Entre as linhas 4-10, um laço de repetição itera sobre a fila para verificar se todos os WED-*states* habilitam as mesmas WED-*transitions* em ambos os esquemas. A instância i é complacente com o esquema W' quando seus WED-*states* habilitam as mesmas WED-*transitions* em W'. Para cada WED-*state* s, na linha 6 e 7, T_{source} armazena as WED-*transitions* habilitadas em W e T_{target} armazena as WED-*transitions* habilitadas em W'. Para isso, a função nativa ENABLED-TRANSITIONS é invocada para retornar um conjunto de WED-*transitions* ao passar dois parâmetros de entrada: o WED-*state* e o esquema. A seguir, a função EQUIVALENT-TRANSITIONS verifica a equivalência entre T_{source} e T_{target}. Se eles não são equivalentes então a função COMPLIANCE-STATE é finalizada retornando o WED-*state* s como o último ponto de complacência da instância i em W' (linha 9). Caso contrário, o próximo WED-*state* da fila é verificado. A instância i é complacente com W' quando a fila inteira tiver sido consumida. Assim, a função COMPLIANCE-STATE finaliza ao retornar $\emptyset$ (linha 11).

```
1: function COMPLIANCE-STATE(i, W, W')
2:     H_i^W ← EXECUTION-HISTORY(i, W)
3:     statesQueue ← WED-STATES(H_i^W)
4:     while statesQueue ≠ ∅ do
5:         s ← statesQueue.dequeue()
6:         T_source ← ENABLED-TRANSITIONS(s, W)
7:         T_target ← ENABLED-TRANSITIONS(s, W')
8:         if ¬ EQUIVALENT-TRANSITIONS(T_source, T_target)
9:             return s
10:    end while
11:    return ∅
12: end function
```

Figura 4.7: *Procedimento para avaliar o estado de complacência*

Para verificar a equivalência entre as WED-*transitions*, a Figura 4.8 descreve a função EQUIVALENT-TRANSITIONS que recebe dois conjuntos de WED-*transitions* como parâmetros de entrada: T_a e T_b. Essa função retorna *true* se os conjuntos de WED-*transitions* são equivalentes, ou $false$, caso contrário. Como descrito acima, assume-se tanto **equivalências imediatas** e **equivalências declaradas**. Nas linhas 2 e 3, a função elimina todas WED-*transitions* que são imediatamente equivalentes. Assim, A é o conjunto de todas as WED-*transitions* de T_a que não são elementos de

T_b, e B é o conjunto de todas WED-*transitions* de T_b que não são elementos de T_a. Na linha 4, a função verifica as equivalências declaradas. A função retorna *true* se para todas WED-*transition* em A existe uma WED-*transition* declarada como equivalente pelo projetista em B, e para toda WED-*transition* em B existe uma WED-*transition* declarara equivalente em A. Caso contrário, a função retorna $false$.

```
1: function EQUIVALENT-TRANSITIONS(T_a, T_b)
2:     A ← T_a − T_b
3:     B ← T_b − T_a
4:     return ∀x ∈ A, ∃y ∈ B | (x ≡ y)∧
                ∀y ∈ B, ∃x ∈ A | (y ≡ x)
5: end function
```

Figura 4.8: *Função que verifica a equivalência entre conjuntos de WED-transitions.*

Na etapa (b), a Figura 4.9 ilustra o procedimento BACKWARD−RECOVERY responsável por conduzir a instância para um ponto de complacência. Esse procedimento tem três parâmetros de entrada: o esquema W, a instância i e o WED-*state* s_{cp} que representa o ponto de complacência. Nas linhas 2 e 3, o procedimento obtém o histórico de execução H_i^W da instância i e seu WED-*state* atual s_c. A seguir, a função TRANSITIONS é invocada para retornar todas WED-*transitions* executadas entre os WED-*states* s_{cp} e s_c em H_i^W. As WED-*transitions* são armazenadas em uma pilha na qual a última WED-*transition* executada é o elemento no topo. Entre as linhas 5-8, um laço de repetição itera sobre os elementos da pilha recuperando as atividades executadas após s_{cp}. Na linha 7, o procedimento RECOVERY é responsável por compensar uma WED-*transition* ao executar uma WED-*compensation*. Nesta função, quando uma WED-*transition* é rotulada como *ação silenciosa* o resultado da WED-*compensation* não é armazenado. O procedimento BACKWARD−RECOVERY finaliza quando a pilha estiver vazia.

```
1: procedure BACKWARD-RECOVERY(W, i, s_cp)
2:     H_i^W ← EXECUTION-HISTORY(i, W)
3:     s_c ← CURRENT-STATE(i)
4:     stack ← TRANSITIONS(H_i^W, s_cp, s_c)
5:     while stack ≠ ∅ do
6:         t ← POP(stack)
7:         RECOVERY(t)
8:     end while
9: end procedure
```

Figura 4.9: *Recuperação* backward *para um WED-state complacente.*

Na etapa (c), a Figura 4.10 descreve a função MIGRATE responsável pela migração da instância para o esquema modificado. Os parâmetros de entrada são a instância i e os esquemas W e W'. Primeiramente, o procedimento obtém o estado atual s_c da instância i. Uma vez que a recuperação *backward* foi anteriormente invocada, o estado atual s_c é complacente com W'. Portanto, nas linhas 3 e 4, a instância i é instanciada em W' e o WED-*state* s_c é criado. Após a migração, a instância i é então monitorada até o fim de sua execução. Esse monitoramento tem como objetivo identificar possíveis reusos dos WED-*states* que são habilitados pelas equivalências da WED-*transitions* entre os esquemas.

O monitoramento da instância decorre no núcleo da abordagem WED-*flow*, mais especificamente

```
1: procedure MIGRATE(i, W, W')
2:     s_c ← CURRENT-STATE(i)
3:     CREATE-INSTANCE(i, W')
4:     INSERT-WED-STATE(s_c, i, W')
5: end procedure
```

Figura 4.10: *Migração da instância i para o esquema W'.*

no procedimento responsável pela execução das WED-*triggers*. A Figura 4.11 descreve uma versão simplificada do algoritmo de funcionamento da WED-*trigger*. O procedimento `WED-TRIGGER` tem quatro parâmetros de entrada descritos a seguir. A WED-*trigger* g, o WED-*state* para ser avaliado, a instância i, e o esquema W'. De acordo com a Definição [...], uma WED-*trigger* g é uma tupla $g = \langle c, t \rangle$, em que c é uma WED-*condition* e t e uma WED-*transition*. Quando um WED-*state* s é gerado e satisfaz c, então g dispara t. Portanto, o procedimento primeiramente avalia o WED-*state* s na linha 4. Se s satisfaz c, então o procedimento deve executar t. Entretanto, se t já tiver sido executado no esquema origem, então seus resultados podem ser reusados. Na linha 7, a função `FIND-EQUIVALENCE` procura por uma execução anterior da WED-*transition* t no esquema W. Se uma execução prévia de t é encontrada, a função retorna o WED-*state* s_{old} produzido por t em W ou, caso contrário, retorna $\emptyset$. Quando s_{old} é encontrado, o procedimento reusa s_{old} ao invocar o procedimento `REUSE` na linha 9. Caso contrário, a WED-*transition* t é executada (linha 11).

```
1: procedure WED-TRIGGER(g, s, i, W')
2:     c ← g.condition
3:     t ← g.transition
4:     if EVALUATE(c, s) then
5:         W ← i.sourceSchema
6:         H_i^W ← EXECUTION-HISTORY(i, W)
7:         s_old ← FIND-EQUIVALENCE(t, H_i^W)
8:         if s_old ≠ ∅ then
9:             REUSE(i, t, s_old, W')
10:        else
11:            EXECUTE(t)
12:        end if
13:    end if
14: end procedure
```

Figura 4.11: *WED-trigger.*

O procedimento responsável pelo reuso de um WED-*state* é descrito na Figura 4.12. O procedimento `REUSE` tem quatro parâmetros de entrada: a instância i, a WED-*transition* t, o WED-*state* s_{old}, e o esquema W'. Esse procedimento obtém o estado atual $s_{current}$ da instância i (linha 2) e o WED-*state* para ser reusado s_{old} (linha 3). O reúso é o resultado da fusão de $s_{current}$ e s_{old} que gera em um novo WED-*state* s_{new} que, por sua vez, é inserido no esquema W' (linha 4).

A fusão ocorre substituindo apenas o conjunto de atributos atualizados por t. Por exemplo, assuma que, no cenário ilustrado na Figura 4.4, a instância i continua sua execução em W' após sua migração. Quando a WED-*transition* t_3 é disparada, seu resultado pode ser reusado a partir de W. Neste ponto, o estado atual da instância i é s_7 e o WED-*state* para ser reusado é s_3. De acordo com o procedimento de reúso, esses WED-*states* são fundidos, substituindo o atributo atualizado por t_3 (*Prova Escrita*). Como resultado, o WED-*state* S'_3 é criado. Na próxima seção, apresentamos

1: **procedure** REUSE(i, t, s_{old}, W')
2: $s_{current} \leftarrow$ CURRENT-STATE(i)
3: $s_{new} \leftarrow$ MERGE($s_{current}, s_{old}$)
4: INSERT-WED-STATE(s_{new}, t, i, W')
5: **end procedure**

Figura 4.12: *Procedimento para o reuso.*

os resultados experimentais do nosso mecanismo.

Capítulo 5

Resultados Experimentais

Com o objetivo de avaliar a viabilidade prática da proposta apresentada nesta tese, o arcabouço WED-*flow* foi estendido para (i) habilitar alterações evolutivas em um WF esquema e (ii) gerenciar as instâncias que estão em execução quando uma alteração evolutiva é efetivada. O desenvolvimento desse mecanismo foi incorporado na ferramenta WED-*tool* (WED-tool, 2016) que representa a materialização da abordagem WED-*flow*, conforme descrito na Seção 2.4.2. Neste capítulo, será apresentado os experimentos realizados no mecanismo de migração para avaliar a solução proposta.

5.1 Descrição dos Experimentos

A execução dos experimentos é baseada na evolução de um processo em que seu esquema W é transformado em um novo esquema W' ao aplicar uma alteração Δ. Embora o processo de uma pós-graduação tenha sido apresentado e explorado no capítulo anterior, para os experimentos, optou-se por um processo hipotético cuja simplicidade contribui para a compreensão dos resultados. O cenário adotado para os experimentos é ilustrado pela Figura 5.1. Por uma questão de clareza, no decorrer da descrição dos experimentos, o esquema de origem será chamado de W e o esquema modificado será chamado de W'. Nesse cenário, o WF esquema W é composto por quatro WED-*transitions* (atividades) A, B, C e D sendo executadas sequencialmente. A alteração Δ remove a WED-*transition* C e insere a WED-*transition* X entre A e B. Com essa alteração, gera-se o WF esquema W' que é composto pelas WED-*transitions* A, X, B e D.

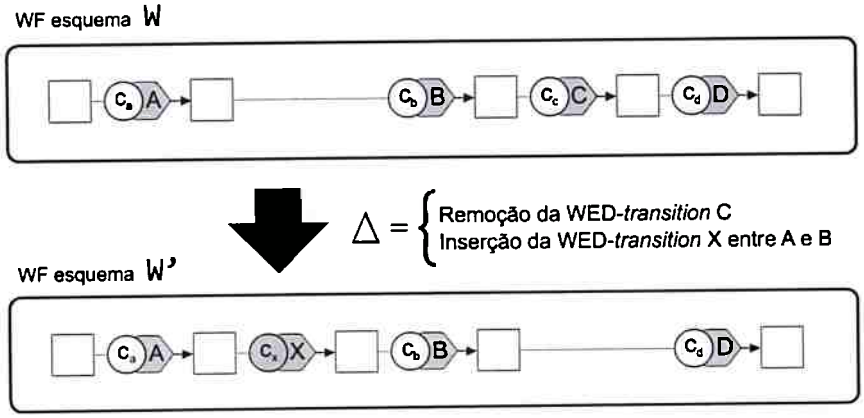


Figura 5.1: *Evolução do processo de negócio adotado para execução dos experimentos.*

Elaborou-se três experimentos para avaliar a solução proposta nesta tese. O primeiro experi-

mento tem como objetivo avaliar a capacidade de migração do mecanismo. O segundo experimento avalia a viabilidade de aplicação prática da solução com relação ao seu desempenho. Por último, o terceiro experimento analisa a diferença de desempenho do mecanismo de migração quando o reuso de WED-*states* está habilitado em comparação quando o reuso está desabilitado.

Os experimentos foram conduzidos em um servidor executando o Sistema Operacional Mac OS X 10.11.15 de 64 bits com processador Intel Core i5 com dois núcleos de 2,6 GHz, 8GB de RAM e 256GB de armazenamento SSD. Além disso, utilizou-se o PostgreSQL 9.3.10[1] e Ruby 2.3.0[2].

5.2 Experimento 1: Capacidade de migração

Neste experimento, objetivou-se avaliar a capacidade de migração do mecanismo proposto no Capítulo 4. A instância em execução somente pode ser migrada quando há garantia que ela não causará erros ou inconsistências ao continuar sua execução no novo esquema. A solução tradicional apresentada na literatura restringe-se a migrar somente as instâncias que são complacentes com as novas especificações propostas no novo esquema. As instâncias não complacentes são excluídas da migração e continuam suas execuções no esquema de origem. Portanto, neste primeiro experimento, analisou-se a capacidade de migração do mecanismo ao se deparar com a migração de instâncias complacentes e não complacentes. Para isso, um lote contendo instâncias inicializadas em W foi submetido ao mecanismo para que as instâncias fossem migradas para W'.

Configuração das instâncias

Para este experimento aplicou-se quatro configurações diferentes para o lote de instâncias. A Figura 5.2 ilustra as configurações utilizadas. Para cada lote, a distribuição das configurações entre as instâncias ocorreu da seguinte forma: 25% na configuração 1, 25% na configuração 2, 25% na configuração 3 e 25% na configuração 4.

Para as instâncias da configuração 1, o estado de dado inicial foi submetido ao WF esquema W habilitando a WED-*transition* A, mas nenhuma WED-*transition* foi executada. Na configuração 2, as instâncias executaram a WED-*transition* A habilitando a execução da WED-*transition* B. Na configuração 3, as instâncias executaram as WED-*transitions* A e B habilitando a execução da WED-*transition* C. Na última configuração, as instâncias executaram as WED-*transitions* A, B e C habilitando a execução da WED-*transition* D.

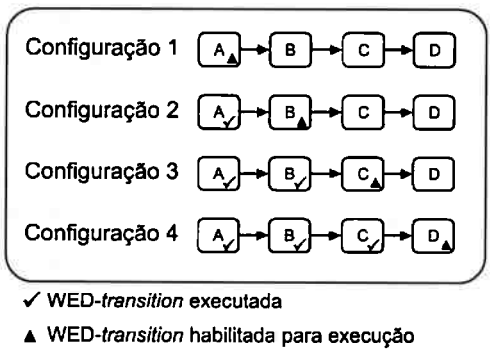


Figura 5.2: *Experimento 1 - Configuração das instâncias submetidas ao mecanismo de migração.*

[1] https://www.postgresql.org/download/
[2] https://www.ruby-lang.org/en/downloads/

Etapas do experimento

O experimento foi conduzido em duas etapas:

- **Execução em W.** As instâncias foram inicializadas em W e prosseguem com a execução até um ponto específico do processo. Nessa etapa, as instâncias foram interrompidas antes do seu término e ficam paradas nas posições definidas pela configuração aplicada ao lote. Ao final da etapa, simula-se o cenário em que uma alteração Δ é efetivada em W e as instâncias que estão em execução devem ser imediatamente migradas para W'.

- **Migração e Execução em W'.** Nessa etapa, cada instância é avaliada segundo o critério de migração e caracterizada como complacente ou não complacente com W'. Quando a instância é avaliada como complacente sua execução é transferida para W'. Se a instância é avaliada como não complacente inicia-se a recuperação parcial (*rollback* parcial) da instância até um ponto (WED-*state*) que seja complacente com W'. Após migrar a instância, o mecanismo supervisiona o restante da execução da instância em W' em busca de oportunidades para reutilizar WED-*states* produzidos em W.

Resultados

A Figura 5.3 ilustra os resultados obtidos pela execução de cinco lotes de instâncias. No primeiro lote 1.000 instâncias foram submetidas ao mecanismo de migração e executadas conforme as etapas descritas anteriormente. Para cada novo lote, acrescentou-se mais 1.000 até chegar ao total de 5.000 instâncias. Os resultados mostram que para todos os lotes, o mecanismo de migração foi capaz de migrar 100% das instâncias, independente da sua configuração.

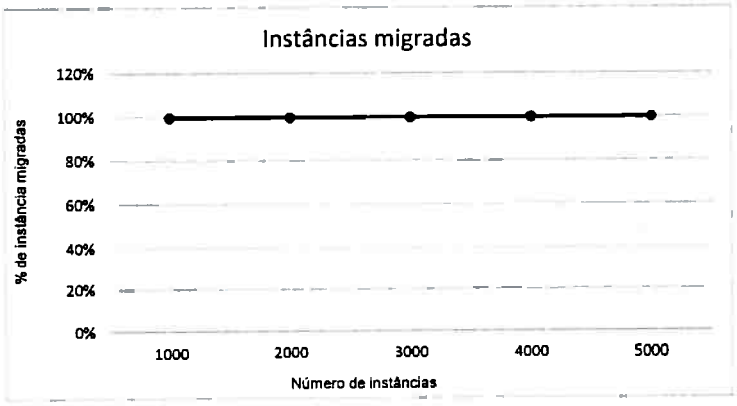


Figura 5.3: *Resultados obtidos no Experimento 1.*

Ao analisar as instâncias agrupadas por configuração, percebe-se que o mecanismo se deparou com diferentes cenários:

- **Configuração 1:** As instâncias ainda não executaram nenhuma WED-*transition* em W. Portanto, pelo critério de migração, elas são avaliadas como complacentes e são migradas. As instâncias prosseguem com a execução das WED-*transitions* A, X, B e D em W'.

- **Configuração 2:** As instâncias executaram a WED-*transition* A e também são avaliadas como complacentes pelo critério de migração e são transferidas para W'. As instâncias prosseguem com a execução das WED-*transitions* X, B e D em W'.

- **Configuração 3:** As instâncias não são complacentes com W', pois as WED-*transitions* A e B foram executadas e no novo esquema a WED-*transition* X é inserida entre A e B. Nessa configuração, o mecanismo de migração precisa acionar a recuperação parcial de cada instância para voltar até um WED-*state* complacente (WED-*state* gerado por A). Após a recuperação, a instância é migrada e continua sua execução em W'. A primeira WED-*transition* habilitada para execução em W' é X. As instâncias prosseguem com a execução da WED-*transition* X, seguido pelo o reuso do WED-*state* gerado por B em W e, por fim, com a execução de D.

- **Configuração 4:** As instâncias não são complacentes com W', pois executaram A, B e C em W. A solução aplicada para tais instâncias é semelhante a aplicada na configuração 3. As instâncias são recuperadas até um WED-*state* complacente com W' e migradas. Em W', as instâncias executam X, reaproveitam o WED-*state* gerado por B em W e finalizam com a execução de D. Nessa configuração, o WED-*state* produzido por C não é reaproveitado porque a atividade foi removida no esquema novo.

Os resultados mostraram que mecanismo de migração foi capaz de migrar todas as instâncias que estavam execução independente de sua configuração. Além disso, nota-se que a capacidade de migração do mecanismo não foi degradada com a submissão de lotes contendo grandes quantidades de instâncias. É importante ressaltar que todas as soluções aplicadas pelo mecanismo de migração foram geradas automáticamente e não requereram intervenção do projetista.

5.3 Experimento 2: Viabilidade de aplicação prática

Este experimento teve como objetivo analisar a viabilidade de aplicação prática do mecanismo proposto. Portanto, analisou-se o tempo gasto pelo mecanismo para tratar da migração tanto de instâncias complacentes quanto de instâncias não complacentes em um cenário de evolução do processo de negócio. Assim como no primeiro experimento, o tempo foi mensurado durante a migração de um lote de instâncias inicializadas em W para W'.

Configuração das instâncias

Neste experimento, utilizou-se as mesmas configurações adotadas no Experimento 1, como ilustrado na Figura 5.2.

Etapas do experimento

As etapas do experimento são semelhantes ao Experimento 1, salvo a etapa de **Migração e Execução em W'**. Neste experimento, a **Migração** e a **Execução em W'** foram consideradas como etapas distintas com o intuito de avaliar o desempenho separadamente. O experimento foi conduzido de acordo com as seguintes etapas:

- **Execução em W.** As instâncias são inicializadas em W. Nesta etapa, mensurou-se o tempo em que as instâncias levaram para chegar até o ponto definido pela configuração em W. Essa medida não é importante para analisar o desempenho do mecanismo de migração, mas será utilizada para fins de comparação.

- **Migração.** Todas as instâncias são migradas para W'. Porém, as instâncias não complacentes passam antes pela recuperação parcial para torná-las complacentes com W'. Nessa etapa, analisou-se o tempo que o mecanismo levou para migrar todas as instâncias do lote. Para que isso fosse possível, fez-se um ajuste no núcleo do WED-*flow* para que, após migrada, a instância não continuasse imediatamente sua execução em W'. O ajuste viabiliza o desligamento das WED-*triggers* de W' para que elas não processem os estados de dados gerados pelo mecanismo de migração.

- **Execução em W'.** As instâncias continuam com sua execução em W'. Nessa etapa, analisou-se o tempo gasto para que o lote de instâncias fosse finalizado em W'. Esta medida é importante para verificar o impacto da sobrecarga causado pelo monitoramento das instâncias após sua migração.

Resultados

O experimento foi realizado com cinco lotes de diferentes quantidades de instâncias. Iniciou-se com um lote de 1.000 instâncias e, para cada novo lote, acrescentou-se mais 1.000 instâncias até o total de 5.000 instâncias. A cada lote, as três etapas descritas anteriormente foram executadas. Além disso, para facilitar o entendimento e interpretação dos resultados, optou-se por realizar o experimento para cada configuração individualmente.

A Figura 5.4 ilustra os resultados obtidos do experimento com todas as instâncias no estado definido pela configuração 1. No gráfico, é possível observar que o tempo de execução em W' (linha com quadrados) foi bem superior ao tempo de execução em W (linha com círculos). Essa diferença é explicada pelo fato de que, nessa configuração, as instâncias foram somente inicializadas em W e logo transferidas para W'. O tempo de migração comparado ao número de instâncias foi consideravelmente baixo. Para avaliar e migrar 5.000 instâncias, por exemplo, o mecanismo levou aproximadamente 505 segundos. A linha com triângulos mostra como o tempo de migração foi pouco impactado pelo aumento do número de instâncias. O fato de as instâncias possuírem um histórico de execução com poucas entradas em W, contribuiu com que a execução fosse mais rápida.

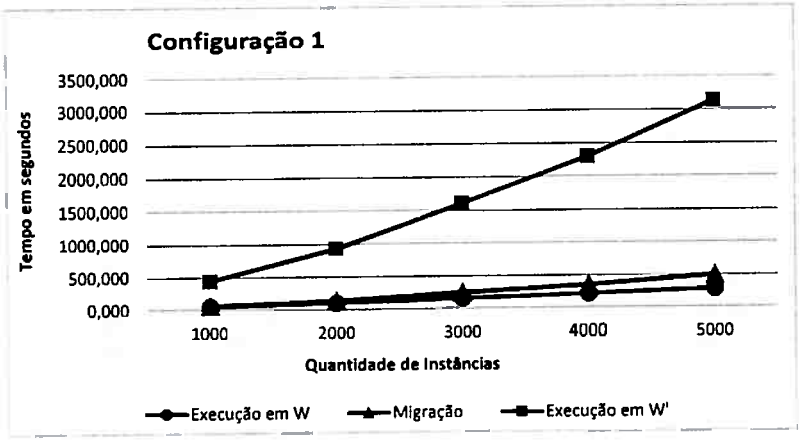


Figura 5.4: *Experimento 2 - Resultados obtidos com a Configuração 1.*

A Figura 5.5 mostra os resultados da execução do experimento com as instâncias na configuração 2. Nessa configuração, devido ao aumento do número de WED-*transitions* executadas em

W, observa-se uma diferença menor entre o tempo de execução em W (linha com círculos) e W' (linha com quadrados). Além disso, percebe-se que o tempo de migração também sofreu um pequeno aumento. A migração de 5.000 instâncias, nessa configuração, durou cerca de 870 segundos. Mesmo considerando que o aumento foi significativo, o gráfico ainda mostra que a curva do tempo de migração não cresce proporcionalmente com o aumento das instâncias.

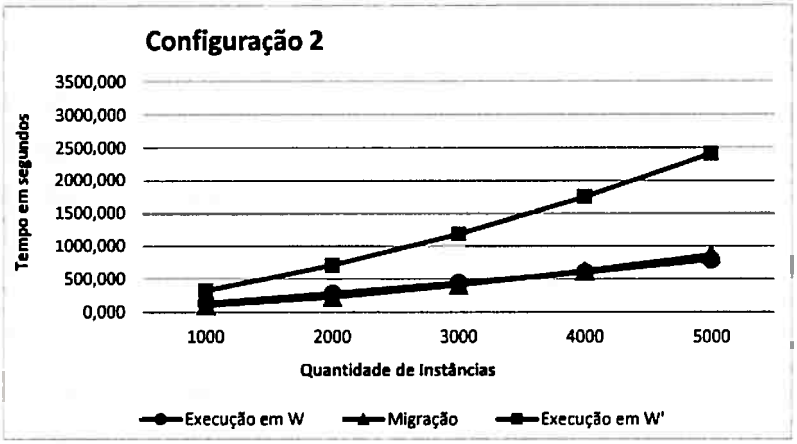


Figura 5.5: *Experimento 2 - Resultados obtidos com a Configuração 2.*

As Figuras 5.6 e 5.7 ilustram os resultados do experimento ao tratar as instâncias com as configurações 3 e 4, respectivamente. Nessas configurações, os resultados obtidos foram bem próximos. Na configuração 3, as instâncias executaram as WED-*transitions* A e B em W e a WED-*transition* D em W'. Na configuração 4, as instâncias executaram as WED-*transitions* A, B e C em W e a WED-*transition* D em W'. A principal diferença entre as configurações é o número de WED-*transitions* executadas em W. Essa diferença pode ser observada pela linha com círculos que é maior na Figura 5.7 por executar uma WED-*transition* a mais. Ao observar o tempo de migração, essa diferença não causou impacto significativo como mostra a linha com triângulos. Embora as instâncias da configuração 4 terem um histórico de execução maior, o tempo de migração se manteve bem próximo da configuração 3. Em ambas configurações, os tempos de execução em W' também foram bem próximos, uma vez que as instâncias executam a mesma quantidade de WED-*transitions* em W'.

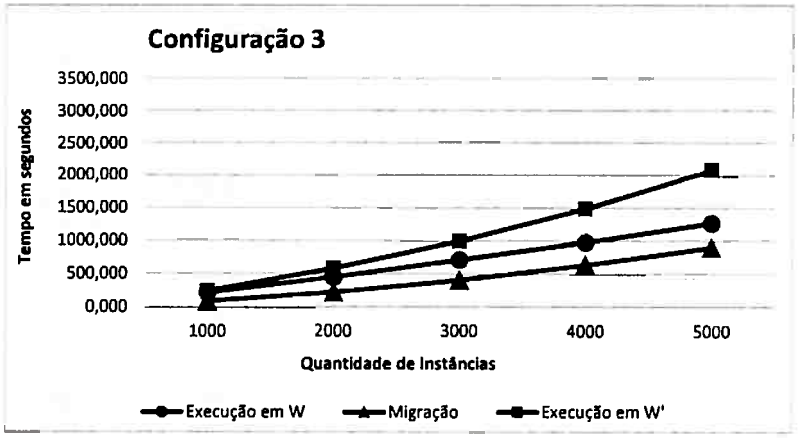


Figura 5.6: *Experimento 2 - Resultados obtidos com a Configuração 3.*

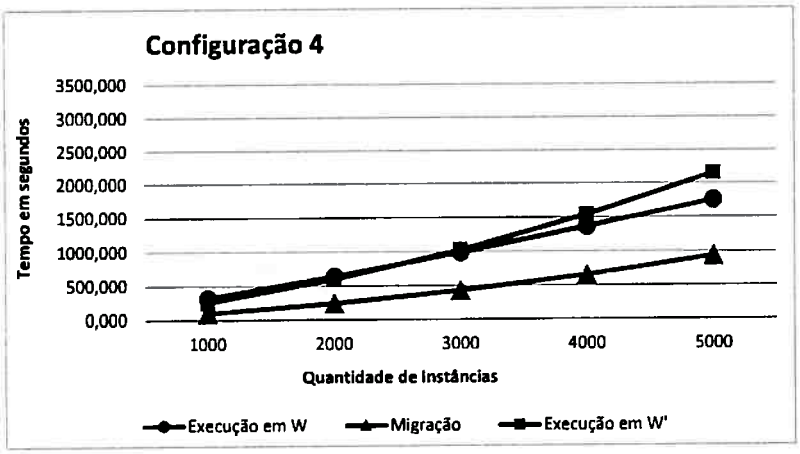


Figura 5.7: *Experimento 2 - Resultados obtidos com a Configuração 4.*

Neste Experimento 2, tentou-se analisar o desempenho do mecanismo de migração sob diferentes vertentes. Analisou-se como o mecanismo se comporta com o aumento da quantidade de instâncias a serem migradas, bem como seu comportamento diante de diferentes configurações de instâncias (complacentes e não complacentes). Em todas as vertentes analisadas, os resultados do tempo de migração se mostraram satisfatórios quando comparado ao tempo gasto para execução em W e W'. Diante desses experimentos, concluiu-se que os resultados indicam que o mecanismo de migração proposto poderia ser aplicado em processos de negócios reais.

5.4 Experimento 3: Reuso de WED-*states*

Este experimento teve como objetivo analisar a influência do reuso de WED-*states* no desempenho do mecanismo de migração. Para isso, analisou-se o tempo que o mecanismo levou para finalizar a execução das instâncias em W' com o reuso de WED-*states* habilitado comparado com o tempo gasto com o reuso desabilitado.

Configuração das instâncias

Neste experimento, todos os lotes de instâncias foram configurados com as definições da configuração 4 ilustrado na Figura 5.2. Utilizou-se somente essa configuração pelo fato de que, após a migração, todas as instâncias tem a oportunidade de reutilizar um estado de dado produzido em W.

Etapas do experimento

Utilizou-se as mesmas etapas definidas no Experimento 2. No entanto, para este experimento, somente considerou-se o tempo gasto para executar as instâncias em W'.

Resultados

O experimento foi realizado utilizando cinco lotes de instâncias como nos experimentos anteriores. A Figura 5.8 ilustra os resultados obtidos com o experimento. A linha com quadrados representa o tempo gasto para que todas as instâncias finalizassem suas execuções após a migração

com o reuso de WED-*states* desabilitado. A linha com círculos mostra o tempo para o término das instâncias com o reuso de WED-*states* habilitado. O gráfico mostra uma redução significativa do tempo quando as instâncias são assistidas pelo mecanismo de migração com o reuso de WED-*states* habilitado. É importante ressaltar que essa diferença é proporcional a quantidade de WED-*states* que são reutilizados e do tempo de processamento da WED-*transition* que gerou o WED-*state* reutilizado.

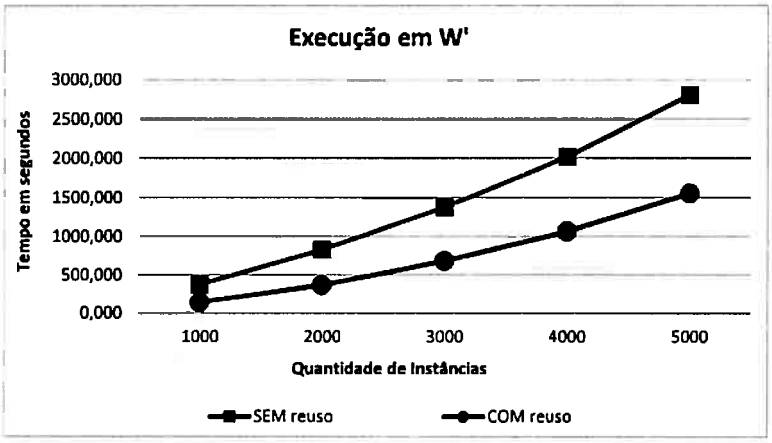


Figura 5.8: *Experimento 3 - Configuração das instâncias submetidas ao mecanismo de migração.*

Para fins comparativos, analisou-se também o tempo de execução em W e de migração quando o reuso de WED-*states* estava habilitado e quando estava desabilitado. Como era esperado, os resultados ilustrados pela Figura 5.9 comprovam que o reuso não exerceu influência alguma nessas etapas. Os resultados se mantiverem idênticos com o reuso habilitado e desabilitado.

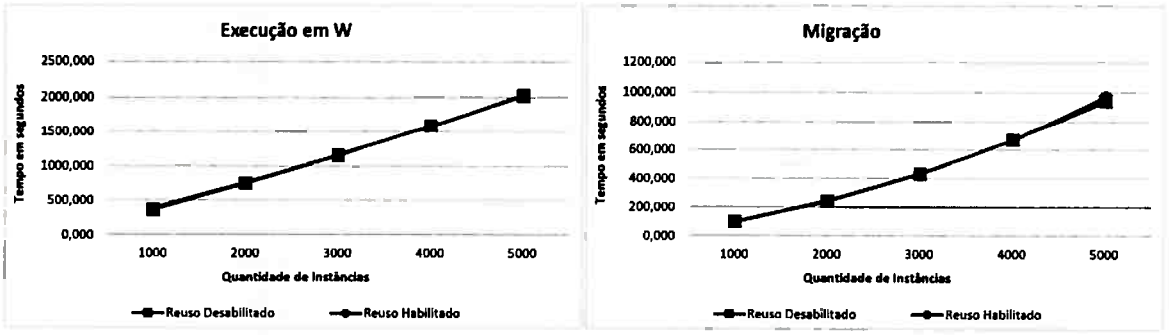


Figura 5.9: *Experimento 3 - Configuração das instâncias submetidas ao mecanismo de migração.*

Os resultados obtidos neste experimento mostraram que o reuso de WED-*states* contribui significativamente para a redução do tempo de processamento das instâncias depois de migradas. Esse resultado também mostra que o reuso de WED-*states* pode reduzir os efeitos colaterais da estratégia de *rollback* parcial no contexto de processos de negócio.

5.5 Discussão

Neste capítulo, o mecanismo de migração proposto foi analisado sob três perspectivas: (i) capacidade de migração; (ii) viabilidade de aplicação prática; e (iii) impacto causado pelo reuso de

WED-*states*.

No primeiro experimento, o mecanismo foi capaz de migrar 100% das instâncias em diferentes testes, mostrando uma eficiente capacidade de migração. Essa eficiência é ainda mais significativa ao levar em consideração que as migrações ocorreram de forma automática sem a intervenção do projetista. Portanto, o *rollback* parcial mostrou-se uma estratégia elegante e genérica para levar uma instância de um estado não complacente para um estado complacente com o novo esquema. Embora conceitualmente o *rollback* parcial aparente ser uma solução simples, sob a perspectiva prática a solução exige do PAIS requisitos não triviais de implementação no contexto de processos. A geração automática dos caminhos de retorno que levam as instâncias à um estado complacente, exige que o PAIS garanta propriedades transacionais adaptadas aos processos.

Outro aspecto a ser considerado é efeito colateral causado por uma solução baseada no *rollback* parcial. O principal ponto negativo está relacionado ao retrabalho. Mais especificamente, na repetição da execução de atividades envolvidas no caminho de volta. No mecanismo de migração proposto nesta tese, tentou-se minimizar esse efeito colateral ao reutilizar, quando possível, no esquema modificado, os WED-*states* produzidos no esquema de origem. O Experimento 3 teve por objetivo avaliar se o fato de reutilizar WED-*states* ameniza o retrabalho inerente do *rollback* parcial. Os resultados ilustrados pela Figura 5.8 foram satisfatórios ao mostrar que houve uma redução significativa no tempo de execução, quando habilitado o reuso de WED-*states*. Salienta-se que essa redução pode ser ainda maior quando há um maior número de WED-*states* reutilizados ou quando há WED-*transitions* com uma alta carga de processamento envolvidos na reutilização dos estados de dados.

Com os resultados do Experimento 2, concluiu-se que o mecanismo apresentou desempenho factível para ser aplicado em cenários reais. No entanto, esse desempenho pode ser ainda melhor com a otimização do mecanismo de migração. Considerando que o mecanismo realiza as migrações de forma sequencial e que não há dependência entre as instâncias, acredita-se que o processo de migração pode ser otimizado com técnicas de computação paralela.

Neste capítulo, apresentou-se os resultados obtidos pela execução de experimentos submetidos ao mecanismo de migração proposto nesta tese. Procurou-se avaliar o mecanismo sob diferentes perspectivas com o objetivo de validar sua concepção. De modo geral, em todas as perspectivas, os resultados dos experimentos mostraram-se satisfatórios e viáveis para implantação do mecanismo de migração em cenários reais. No próximo capítulo, serão apresentadas as principais conclusões deste trabalho, bem como algumas sugestões para trabalhos futuros.

Capítulo 6

Conclusões

Diante do dinamismo dos negócios atuais apresentado no meio corporativo, é crescente a exigência de que as organizações mantenham seus processos em contínua evolução para garantir sua competitividade no mercado. Nesse cenário, os PAIS têm exercido um papel fundamental fornecendo apoio flexível e dinâmico para o gerenciamento de processos por meio da distinção entre a lógica do processo e o código da aplicação. Embora essa distinção propicie um alicerce favorável para gerenciar a evolução do processo, a migração das instâncias em execução ainda apresenta desafios a serem superados pelos PAIS. Dentre esses desafios, este trabalho concentrou-se no tratamento das instâncias em execução que não são complacentes com o esquema modificado, mas que necessitam serem migradas.

O trabalho teve seu desenvolvimento guiado pela hipótese de que com a integração de diferentes perspectivas do processo aliado a conceitos de modelos transacionais avançados seria possível ampliar a capacidade de migração de um PAIS. Para validar essa hipótese, a abordagem WED-*flow* foi estendida com a inclusão de um mecanismo responsável pela migração das instâncias em execução quando uma alteração no esquema é efetivada. Resumidamente, o mecanismo define três etapas bem definidas: (i) avaliar o estado de complacência da instância; (ii) promover ajustes nas instâncias avaliadas como não complacentes; e (iii) migrar a instância para o esquema modificado. Na etapa (i), adotou-se um importante critério de migração proposto por (Casati *et al.*, 1998) que se baseia na verificação da história de execução da instância no esquema modificado. As instâncias avaliadas como não complacentes, na etapa (ii), são ajustadas de modo que elas se tornem complacentes com o esquema modificado. Esses ajustes são projetados de forma automática e personalizada de acordo com o estado de cada instância. Para isso, explorou-se a perspectiva transacional provida pelo arcabouço WED-*flow* para proporcionar rotinas de recuperação automática (*rollback* parcial). Dessa forma, encontrado o ponto de complacência da instância, o mecanismo é capaz de compor automaticamente o caminho de recuperação. Por fim, na etapa (iii), a instância é transferida para o esquema modificado.

Com a implementação da solução proposta, o mecanismo mostrou-se capaz de migrar qualquer instância para o esquema modificado. Diferentemente da solução tradicional que exclui as instâncias não complacentes da migração, o mecanismo foi capaz de migrá-las sem intervenção manual. Os resultados do Experimento 1, descritos na Seção 5.2, mostraram que o mecanismo habilitou a migração de 100% das instâncias em execução independente do seu estado. Com esses resultados, por um lado, argumenta-se que a capacidade de migração foi consideravelmente ampliada quando comparada com as soluções existentes na literatura. Por outro lado, é importante discutir sobre

um possível efeito colateral da solução proposta. O ajuste das instâncias não complacentes com base em *rollback* parcial pode acarretar em um desperdício de trabalho resultante das atividades de compensação. Esse possível desperdício pode ser assumido como o "preço" a ser pago pela autonomia e amplitude da solução. No entanto, esse aspecto traz à tona uma questão que transcende o escopo deste trabalho, a análise do custo-benefício da solução.

Motivado por essa discussão, explorou-se a influência que o dado exerce sobre a instância no WED-*flow* para tentar reduzir esse possível efeito colateral. Uma vez que a instância possui estados de dados bem definidos a cada execução de uma atividade, propôs-se reutilizar esses estados de dados durante a migração, ao levar em consideração alguns aspectos semânticos. Do ponto de vista pragmático, o mecanismo permite que um WED-*state* gerado pela instância no esquema de origem, seja reutilizado no esquema modificado. Sob uma perspectiva transacional, o reuso de WED-*states* representa passos de recuperação *forward* que desviam a instância do seu caminho normal. O reuso de WED-*states* somente é permitido quando há equivalências entre os esquemas. Os aspectos semânticos considerados foram: (1) atividades cujos resultados não necessitam ser compensados; e (2) equivalências declaradas pelo projetista. No primeiro aspecto, explorou-se o fato de que para algumas atividades (como no processo do programa de pós-graduação) não faz sentido desfazer os efeitos de sua execução no caminho da recuperação. Com isso, os resultados dessas atividades (estados de dados) podem ser reutilizados no esquema modificado. As compensações das atividades cujos resultados poderiam ser reutilizados foram rotuladas como uma *ação silenciosa* τ, tal como apresentado em (Aalst e Basten, 2002). No segundo aspecto, analisou-se atividades que são diferentes, porém produzem resultados semanticamente equivalentes. Essa equivalência é definida pelo especialista do negócio e especificada pelo projetista. Essa declaração de equivalências além de ampliar os pontos que habilitam o reuso dos estados de dados, flexibiliza o critério de migração reduzindo o número de instâncias classificadas como não complacentes.

O reuso dos estados de dados mostrou-se promissor com os resultados do Experimento 3, apresentados na Seção 5.4. Nesse experimento, como resultado quantitativo, percebeu-se uma redução significativa no tempo de execução das instâncias, quando habilitado o reuso dos WED-*states*. A solução mostra-se promissora, pois quanto maior for a carga de processamento da atividade cujo resultado será reutilizado, maior será a redução do tempo de execução. Outro aspecto que se deve levar em consideração, mas que não foi alvo deste trabalho, é a investigação do valor qualitativo representado pela redução de retrabalho com a reutilização dos estados de dados.

Por fim, a solução proposta nesta tese evidenciou no Experimento 2, desempenho factível para aplicação prática em cenários reais. O mecanismo foi desenvolvido para tratar da migração das instâncias de forma sequencial que podem ser aprimorados com técnicas de computação paralela. Assim, os resultados apresentados na Seção 5.3 ainda podem ser melhorados.

O limite do trabalho proposto pode-se resumir pela necessidade de características específicas do processo para que o mecanismo seja capaz de reutilizar estados de dados no procedimento de migração. O reuso dos estados de dados somente é possível em processos que possuem algumas atividades que não exijam compensação. Se o processo possui tais características, o mecanismo será capaz de migrar todas as instâncias não complacentes e amenizar o retrabalho causado pelo *rollback* parcial. Caso contrário, o mecanismo ainda será capaz de migrar as instâncias não complacentes, porém, não conseguirá evitar o retrabalho após a migração.

6.1 Contribuições

A principal contribuição deste trabalho é caracterizada pela **concepção de um mecanismo capaz de migrar todas as instâncias não complacentes diante de uma evolução do esquema do processo.** Além disso, esta tese apresenta outras contribuições secundárias. São elas:

- **Ambiente de recuperação no contexto dos PAIS.** Para promover ajustes automáticos nas instâncias não complacentes, foi implementado um ambiente de recuperação no contexto dos PAIS utilizando conceitos dos modelos transacionais avançados. Esse ambiente de recuperação pode representar um alicerce sólido para prover PAIS cada vez mais flexíveis e adaptativos.

- **Reuso dos resultados de uma atividade.** O reuso dos resultados de uma determinada atividade é uma situação que pode ocorrer na prática de um processo de negócio. O ciclo de um aluno de pós-graduação é um exemplo disso. Uma vez que o aluno está aprovado em alguma etapa do programa, no caso de uma migração, essa aprovação deveria ser reutilizada no novo esquema. No entanto, a implementação desse reuso somente foi possível pelo fato da instância ser composta de estados de dados bem definidos.

- **Flexibilização do critério de migração baseado na verificação do histórico de execução.** O critério de migração baseada na verificação do histórico proposta em (Casati *et al.*, 1998) tem sido criticada pela sua rigidez na avaliação do estado de complacência das instâncias (Rinderle-Ma e Reichert, 2010). A declaração da equivalência entre duas WED-*transitions* de esquemas distintos, flexibiliza o critério de migração ao reduzir o número de instâncias avaliadas como não complacentes.

6.2 Sugestões para Pesquisas Futuras

No decorrer do desenvolvimento desta tese de doutorado, outras sugestões surgiram para dar prosseguimento no trabalho desenvolvido. Algumas das sugestões já estão em andamento e outras estão em aberto. As sugestões são descritas a seguir.

Otimização do algoritmo de migração de instâncias. O procedimento responsável pela migração das instâncias em execução é sequencialmente executado, como descrito na Seção 4.3. Acredita-se que com alternativas de computação paralela é possível otimizar esse procedimento.

Sincronização entre esquemas. Em estudos realizados pelo grupo DATA, percebeu-se que em alguns cenários há a necessidade de sincronizar duas ou mais instâncias em um ponto específico no processo. Esta característica está em estudo na abordagem WED-*flow*.

Ampliação da expressividade da abordagem WED-*flow*. Como pode ser visto em (Liberato *et al.*, 2015) a abordagem WED-*flow* é capaz de representar diferentes padrões de comportamentos de divergência (*split*) e convergência (*join*). Apesar disso, nem todas as abstrações de controle de fluxo estão claramente descritas na linguagem WED-*flow*. Há também comportamentos mais específicos, como *milestone*, que apesar de facilmente representados pela abordagem

WED-*flow* ainda requerem uma investigação mais minuciosa. O principal desafio é representar um comportamento de controle fluxo e, ao mesmo tempo, manter a capacidade de recuperação quando alguma exceção ocorre.

Propriedades de corretude em modelos WED-*flow*. A verificação da corretude dos modelos de processo é uma tarefa fundamental durante a fase de modelagem do processo. Em tempo de projeto, deve-se garantir que os modelos de processo que estão sendo modelados poderão ser corretamente executados. Por exemplo, uma instância do processo quando executado pelo PAIS não deveria ser conduzida para estados espúrios devido à erros de modelagem. Ao verificar os modelos de processo em tempo de projeto, potenciais problemas podem ser identificados antecipadamente e o modelo pode ser corrigido antes de sua implantação. Técnicas para verificação de corretude de esquemas de processos modelados na abordagem WED-*flow* ainda é uma questão de pesquisa em aberto que necessita ser melhor explorada.

Ferramenta para modelagem visual de processos WED-*flow*. Atualmente, uma das iniciativas de pesquisa do grupo DATA tem sido a definição de uma linguagem declarativa baseada no padrão SQL para manipulação dos componentes primitivos da abordagem WED-*flow*. Entretanto, ferramentas visuais intuitivas que permitam a modelagem dos processos ainda necessitam ser desenvolvidas.

Monitoramento de execução dos processos WED-*flow*. Atualmente, o núcleo e o gerenciador de recuperação do WED-*flow* são implementados na forma de protótipos que são usados para execução de provas de conceito, como apresentado na Seção 5. Apesar disso, uma ferramenta visual para o monitoramento e acompanhamento da execução dos processos também ainda necessita ser melhor implementada.

Referências Bibliográficas

Aalst e Basten (2002) W. M. P. van der Aalst e T. Basten. Inheritance of workflows: An approach to tackling problems related to change. *Theoretical Computer Science*, 270(1):125–203. ISSN 0304-3975. Citado na pág. vi, 36, 37, 38, 66, 83

Aalst e Jablonski (2000) W. M. P. van der Aalst e S. Jablonski. Dealing with workflow change: identification of issues and solutions. *International Journal of Computer Systems Science and Engineering*, 15(5):267–276. Citado na pág. 36

Aalst e van Hee (2004) W. M. P. van der Aalst e Kees Max van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge, MA, USA. ISBN 0262720469. Citado na pág. 7, 10

Aalst *et al.* (2003a) W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski e A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51. ISSN 0926-8782. Citado na pág. 7

Aalst (1998) Wil M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66. Citado na pág. 10, 37

Aalst *et al.* (1999) Wil M. P. van der Aalst, Twan Basten, H. M. W. (Eric) Verbeek, Peter A. C. Verkoulen e Marc Voorhoeve. Adaptive workflow: On the interplay between flexibility and support. Em *Proceedings of the 1st International Conference on Enterprise Information Systems*, volume 2, páginas 353–360. Citado na pág. 34

Aalst *et al.* (2003b) Wil M. P. van der Aalst, Arthur H. M. ter Hofstede e Mathias Weske. Business process management: A survey. Em *Proceedings of the International Conference on Business Process Management*, BPM'03, páginas 1–12, Berlin, Heidelberg. Springer-Verlag. Citado na pág. 8

Aalst (2001) Wil van der Aalst. Exterminating the dynamic change bug: A concrete approach to support workflow change. *Information Systems Frontiers*, 3(3):297–317. ISSN 1387-3326. Citado na pág. 2, 36, 40, 61

Aalst *et al.* (2005) W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, N. Russell, H.M.W. Verbeek e P. Wohed. Life after bpel? Em Mario Bravetti, Leïla Kloul e Gianluigi Zavattaro, editors, *Formal Techniques for Computer Systems and Business Processes*, volume 3670 of *Lecture Notes in Computer Science*, páginas 35–50. Springer Berlin Heidelberg, Berlin, Heidelberg. Citado na pág. 8

Aalst *et al.* (2009) W.M.P. van der Aalst, M. Pesic e H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development*, 23 (2):99–113. ISSN 1865-2034. Citado na pág. 20, 34, 36, 41, 57, 58

Björk (1973) Lawrence A. Björk. Recovery scenario for a db/dc system. Em *Proceedings of the ACM annual conference*, ACM'73, páginas 142–146, New York, NY, USA. ACM. Citado na pág. 22

Bry *et al.* **(2006)** François Bry, Michael Eckert, Paula-Lavinia Pătrânjan e Inna Romanenko. Realizing business processes with eca rules: Benefits, challenges, limits. Em *Proceedings of the 4th International Conference on Principles and Practice of Semantic Web Reasoning*, PPSWR'06, páginas 48–62. Citado na pág. 17, 33

Carlsen *et al.* **(1997)** Steinar Carlsen, John Krogstie, Arne Solvberg e Odd Ivar Lindland. Evaluating flexible workflow systems. Em *Proceedings of the 30th Hawaii International Conference on System Sciences*, volume 2, páginas 230–239. Citado na pág. 1

Casati *et al.* **(1998)** F. Casati, S. Ceri, B. Pernici e G. Pozzi. Workflow evolution. *Data & Knowledge Engineering - Special issue on ER '96*, 24(3):211–238. Citado na pág. 2, 41, 58, 60, 82, 84

Casati *et al.* **(2000)** F. Casati, S. Castano, M. Fugini, I. Mirbel e B. Pernici. Using patterns to design rules in workflows. *IEEE Transactions on Software Engineering*, 26(8):760–785. ISSN 0098-5589. Citado na pág. 17, 43

Coalition (1995) Workflow Management Coalition. The workflow reference model, document number tc00-1003, 1995. URL http://www.wfmc.org/standards/docs/tc003v11.pdf. Citado na pág. 7

Curtis *et al.* **(1992)** Bill Curtis, Marc I. Kellner e Jim Over. Process modeling. *Communications of the ACM - Special issue on analysis and modeling in software development*, 35(9):75–90. ISSN 0001-0782. Citado na pág. 6, 7

Davies Jr (1973) Charles T. Davies Jr. Recovery semantics for a db/dc system. Em *Proceedings of the ACM annual conference*, ACM '73, páginas 136–141, New York, NY, USA. ACM. Citado na pág. 22

Davies Jr (1978) Charles T. Davies Jr. Data processing spheres of control. *IBM Systems Journal*, 17(2):179–198. ISSN 0018-8670. Citado na pág. 22

Dayal (1994) Umeshwar Dayal. *Active Database Systems: Triggers and Rules for Advanced Database Processing.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1558603042. Citado na pág. 17

Dumas *et al.* **(2005)** Marlon Dumas, Wil M. van der Aalst e Arthur H. ter Hofstede. *Process-aware Information Systems: Bridging People and Software Through Process Technology.* John Wiley & Sons, Inc., New York, NY, USA. ISBN 0-47166-360-9. Citado na pág. 6, 7

Ellis *et al.* **(1995)** Clarence Ellis, Karim Keddara e Grzegorz Rozenberg. Dynamic change within workflow systems. Em *Proceedings of Conference on Organizational Computing Systems*, COCS '95, páginas 10–21, New York, NY, USA. ACM. ISBN 0-89791-706-5. Citado na pág. 2, 36, 39

Elmagarmid (1992) Ahmed K. Elmagarmid, editor. *Database Transaction Models for Advanced Applications.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1-55860-214-3. Citado na pág. 21

Elmasri e Navathe (1999) Ramez A. Elmasri e Shankrant B. Navathe. *Fundamentals of Database Systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd ed. ISBN 0805317554. Citado na pág. 62

Ferreira *et al.* **(2010a)** João Eduardo Ferreira, Wu Qinyi, Simon Malkowski e Calton Pu. Towards flexible event-handling in workflows through data states. Em *Proceedings of 6th World Congress on Services (SERVICES-1)*, páginas 344 –351. Citado na pág. 25

Ferreira *et al.* **(2010b)** João Eduardo Ferreira, Osvaldo Kotaro Takai, Simon Malkowski e Calton Pu. Reducing exception handling complexity in business process modeling and implementation: the wed-flow approach. Em *Proceedings of the International Conference on the Move to Meaningful Internet Systems - Volume Part I*, OTM'10, páginas 150–167, Berlin, Heidelberg. Springer-Verlag. ISBN 3-642-16933-3, 978-3-642-16933-5. Citado na pág. 4, 9, 25

Ferreira *et al.* **(2012)** João Eduardo Ferreira, Kelly Rosa Braghetto, Osvaldo Kotaro Takai e Calton Pu. Transactional recovery support for robust exception handling in business process services. Em *Proceedings of the 19th International Conference on Web Services (ICWS)*, páginas 303–310. Citado na pág. 4, 25, 28, 29

Fokkink (2000) Wan Fokkink. *Introduction to Process Algebra.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2st ed. ISBN 354066579X. Citado na pág. 13

Garcia (2013) Marcela Ortega Garcia. Implementação do arcabouço wed-flow para controle de processos transacionais. Dissertação de Mestrado, Universidade de São Paulo, São Paulo, SP, Brasil. Citado na pág. 32

Garcia *et al.* **(2012a)** Marcela Ortega Garcia, Kelly Rosa Braghetto, Calton Pu e João Eduardo Ferreira. An implementation of a transaction model for business process systems. *Journal of Information and Data Management*, 3(3):271–286. Citado na pág. 25, 32

Garcia *et al.* **(2012b)** Marcela Ortega Garcia, Pedro Paulo de S. B. da Silva, Kelly Rosa Braghetto e João Eduardo Ferreira. Wed-tool: uma ferramenta para o controle de execução de processos de negócio transacionais. Em *Proceedings of the 27th Brazillian Symposium on Databases - Demos and Applications Session.* Citado na pág. 34

García-Molina e Salem (1987) Héctor García-Molina e Kenneth Salem. Sagas. *ACM SIGMOD Record*, 16(3):249–259. ISSN 0163-5808. Citado na pág. 21, 23, 62

Georgakopoulos *et al.* **(1995)** Diimitrios Georgakopoulos, Mark Hornick e Amit Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153. Citado na pág. 7

Gupta *et al.* **(2011a)** Nitin Gupta, Lucja Kot, Sudip Roy, Gabriel Bender, Johannes Gehrke e Christoph Koch. Entangled queries: Enabling declarative data-driven coordination. Em *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD'11, páginas 673–684, New York, NY, USA. ACM. ISBN 978-1-4503-0661-4. Citado na pág. 23

Gupta *et al.* **(2011b)** Nitin Gupta, Milos Nikolic, Sudip Roy, Gabriel Bender, Lucja Kot, Johannes Gehrke e Christoph Koch. Entangled transactions. *VLDB Endowment*, 4(11):887–898. Citado na pág. 23

Hensinger *et al.* **(2000)** Clemens Hensinger, Manfred Reichert, Thomas Bauer, Thomas Strzeletz e Peter Dadam. $\text{Adept}_{\text{workflow}}$ - advanced workflow technology for the efficient support of adaptive, enterprise-wide processes. Em *Proc. Software Demonstration Track, Held in conjunction with EDBT '00 conference*, páginas 29–30. Citado na pág. 47

Jablonski e Bussler (1996) Stefan Jablonski e Christoph Bussler. *Workflow management - modeling concepts, architecture and implementation.* International Thomson Computer Press. ISBN 978-1-85032-222-1. Citado na pág. 7

Kamath e Ramamritham (1996) Mohan Kamath e Krithi Ramamritham. Correctness issues in workflow management. *Distributed Systems Engineering*, 3(4):213. Citado na pág. 35

Kot *et al.* **(2010)** Lucja Kot, Nitin Gupta, Sudip Roy, Johannes Gehrke e Christoph Koch. Beyond isolation: Research opportunities in declarative data-driven coordination. *ACM SIGMOD Record*, 39(1):27–32. ISSN 0163-5808. Citado na pág. 21

Kradolfer e Geppert (1999) Markus Kradolfer e Andreas Geppert. Dynamic workflow schema evolution based on workflow type versioning and workflow migration. Em *Proceedings of the IFCIS International Conference on Cooperative Information Systems*, CoopIS'99, páginas 104–114. IEEE Computer Society. ISBN 0-7695-0384-5. Citado na pág. vi, 41, 43, 44, 45, 47, 58

Kradolfer *et al.* (1999) Markus Kradolfer, Andreas Geppert e Klaus R. Dittrich. Workflow specification in trams. Em *Proceedings of the 18th International Conference on Conceptual Modeling*, ER'99, páginas 263–277, London, UK, UK. Springer-Verlag. ISBN 3-540-66686-9. Citado na pág. 41, 43, 58

Liberato *et al.* (2014) Rafael Liberato, André Luis Schwerz, Osvaldo Kotaro Takai e João Eduardo Ferreira. Gerenciador de dados compartilhados entre processos de negócio: Data-control process para wed-tool. Em *VIII Workshop Brasileiro em Gestão de Processos de Negócios*, volume 02 of *WBPM 2014*, páginas 26–33, Londrina, PR, Brasil. X Simpósio Brasileiro de Sistemas de Informação. Citado na pág. 4

Liberato *et al.* (2015) Rafael Liberato, André Luis Schwerz, Calton Pu e João Eduardo Ferreira. Abordagem transacional para padrões de controle de fluxo de processos científicos e de negócios. Em *XXX Simpósio Brasileiro de Banco de Dados*, SBBD 2015, páginas 15–20, Petrópolis, RJ, Brasil. XXX Simpósio Brasileiro de Banco de Dados. Citado na pág. 5, 84

Liberato *et al.* (2016) Rafael Liberato, André Luis Schwerz, Calton Pu e João Eduardo Ferreira. Increasing the migration of non-compliant instances in adaptive pais. Manuscrito submetido para publicação, 2016. Citado na pág. 5

Liu *et al.* (2000) Ling Liu, Calton Pu e Wei Tang. Continual queries for internet scale event-driven information delivery. *IEEE Transactions on Knowledge and Data Engineering*, 12(5): 861–861. ISSN 1041-4347. Citado na pág. 33

Lu e Sadiq (2007) Ruopeng Lu e Shazia Sadiq. A survey of comparative business process modeling approaches. Em *Proceedings of the 10th International Conference on Business Information Systems*, BIS'07, páginas 82–94, Berlin, Heidelberg. Springer-Verlag. ISBN 978-3-540-72034-8. Citado na pág. 17

Milner (1982) R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA. ISBN 0387102353. Citado na pág. 16

Montali (2010) Marco Montali. *Specification and Verification of Declarative Open Interaction Models - A Logic-Based Approach*, volume 56 of *Lecture Notes in Business Information Processing*. Springer. ISBN 978-3-642-14537-7. Citado na pág. vi, 18

Moss (1985) J. E. B. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing*. Massachusetts Institute of Technology, Cambridge, MA, USA. ISBN 0-262-13200-1. Citado na pág. 22, 24

Muehlen (2004) M. zur Muehlen. *Workflow-based Process Controlling: Foundation, Design, and Implementation of Workflow-driven Process Information Systems*, volume 6 of *Advances in Information Systems and Management Science*. Logos Verlag Berlin, Berlin, Germany. Citado na pág. 7

Murata (1989) Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580. ISSN 0018-9219. Citado na pág. 9, 10

OASIS (2007) OASIS. Business process execution language for web services - version 2.0, 2007. URL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf. Citado na pág. 8

OMG (2011) OMG. Business process model and notation, 2011. URL http://www.omg.org/spec/BPMN/2.0. Citado na pág. 8

Reichert e Dadam (1997) Manfred Reichert e Peter Dadam. A framework for dynamic changes in workflow management systems. Em *Proceedings of Eighth International Workshop on Database and Expert Systems Applications*, páginas 42–48. Citado na pág. 2, 12, 41, 47, 58

Reichert e Dadam (1998) Manfred Reichert e Peter Dadam. $\text{Adept}_{\text{flex}}$ - supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems - Special issue on workflow management systems*, 10(2):93–129. ISSN 0925-9902. Citado na pág. vi, 41, 47, 48, 49, 50, 51, 52

Reichert e Weber (2012) Manfred Reichert e Barbara Weber. *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies.* Springer-Verlag Berlin Heidelberg. ISBN 978-3-642-30408-8. Citado na pág. 1, 18, 20

Reuter *et al.* (1997) Andreas Reuter, Kerstin Schneider e Friedemann Schwenkreis. Contracts revisited. Em Sushil Jajodia e Larry Kerschberg, editors, *Advanced Transaction Models and Architectures*, páginas 127–151. Springer US, Boston, MA. ISBN 978-1-4613-7851-8. Citado na pág. 24

Rinderle (2004) Stefanie Rinderle. *Schema Evolution in Process Management Systems.* Tese de Doutorado, Uni Ulm. Citado na pág. 1

Rinderle *et al.* (2003) Stefanie Rinderle, Manfred Reichert e Peter Dadam. Evaluation of correctness criteria for dynamic workflow changes. Em *Proceedings of the International Conference on Business Process Management*, páginas 41–57, Berlin, Heidelberg. Springer-Verlag. ISBN 978-3-540-40318-0. Citado na pág. 3, 36, 60

Rinderle *et al.* (2004a) Stefanie Rinderle, Manfred eichert e Peter Dadam. Flexible support of team processes by adaptive workflow systems. *Distributed and Parallel Databases*, 16(1):91–116. ISSN 0926-8782. Citado na pág. vi, 41, 47, 52, 53, 54, 55, 58

Rinderle *et al.* (2004b) Stefanie Rinderle, Manfred Reichert e Peter Dadam. Correctness criteria for dynamic changes in workflow systems: A survey. *Data & Knowledge Engineering - Special issue: Advances in business process management*, 50(1):9–34. ISSN 0169-023X. Citado na pág. 36

Rinderle-Ma e Reichert (2010) Stefanie Rinderle-Ma e Manfred Reichert. Advanced migration strategies for adaptive process management systems. Em *12th IEEE Conference on Commerce and Enterprise Computing (CEC 2010).* URL http://eprints.cs.univie.ac.at/86/. Citado na pág. 84

Rinderle-Ma *et al.* (2008) Stefanie Rinderle-Ma, Manfred Reichert e Barbara Weber. Relaxed compliance notions in adaptive process management systems. Em *Proceedings 27th International Conference on Conceptual Modeling*, number 5231 in ER'08, páginas 232–247. Citado na pág. 67

Russell (2007) Nicholas Charles Russell. *Foundations of Process-Aware Information Systems.* Tese de Doutorado, Queensland University of Technology. Citado na pág. 8

Sadiq *et al.* (2000) Shazia W. Sadiq, Olivera Marjanovic e Maria E. Orlowska. Managing change and time in dynamic workflow processes. *International Journal of Cooperative Information Systems*, 9(1-2):93–116. Citado na pág. 60, 62

Salem *et al.* (1994) Kenneth Salem, Héctor García-Molina e Jeannie Shands. Altruistic locking. *ACM Transactions on Database Systems (TODS)*, 19(1):117–165. ISSN 0362-5915. Citado na pág. 22

Silva (2013) Pedro Paulo Souza Bento da Silva. Uma abordagem transacional para o tratamento de exceções em processos de negócio. Dissertação de Mestrado, Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, SP, Brasil. Citado na pág. 25, 34

Silva *et al.* **(2012)** Pedro Paulo Souza Bento da Silva, Kelly Rosa Braghetto e João Eduardo Ferreira. Uma abordagem ad-hoc para o tratamento de exceções em processos transacionais. Em *Proceedings of the 27th Brazilian Symposium on Databases.* Citado na pág. 63

Song e Jacobsen (2016) W. Song e H. A. Jacobsen. Static and dynamic process change. *IEEE Transactions on Services Computing*, PP(99):1–1. ISSN 1939-1374. doi: 10.1109/TSC.2016.2536025. Citado na pág. 60

Song *et al.* **(2013)** Wei Song, Xiaoxing Ma, Hao Hu, Yang Zou e Gongxuan Zhang. Migration validity of ws-bpel instances revisited. Em *Proceedings of the IEEE 16th International Conference on Computational Science and Engineering*, CSE '13, páginas 1013–1020. IEEE Computer Society. ISBN 978-0-7695-5096-1. doi: 10.1109/CSE.2013.148. Citado na pág. 67

Wächter e Reuter (1992) Helmut Wächter e Andreas Reuter. The contract model. Em Ahmed K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, páginas 219–263. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1-55860-214-3. Citado na pág. 24

WED-flow (2016) WED-flow. Wed-flow official website, 2016. URL http://data.ime.usp.br/wedflow. Citado na pág. 25

WED-tool (2016) WED-tool. Wed-tool official website, 2016. URL http://data.ime.usp.br/wedflow/wed-tool.html. Citado na pág. 73

Weikum e Schek (1992) Gerhard Weikum e Hans Jörg Schek. Concepts and applications of multilevel transactions and open nested transactions. Em Ahmed K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, páginas 515–553. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1-55860-214-3. Citado na pág. 24

Weske (2007) Mathias Weske. *Business Process Management: Concepts, Languages, Architectures.* Springer-Verlag New York, Inc., Secaucus, NJ, USA. ISBN 3540735216. Citado na pág. 12

Wohed *et al.* **(2003)** Petia Wohed, Wil M. P. van der Aalst, Marlon Dumas e Arthur H. M. ter Hofstede. Analysis of web services composition languages: The case of bpel4ws. Em *Proceedings 22nd International Conference on Conceptual Modeling*, páginas 200–215, Chicago, IL, USA. Citado na pág. 8