Uma abordagem transacional para o tratamento de exceções em processos de negócio

Pedro Paulo de Souza Bento da Silva

Dissertação de Mestrado

Instituto de Matemática e Estatística
da
Universidade de São Paulo

Programa: Programa de Mestrado em Ciência da Computação
Orientador: Prof. Dr. João Eduardo Ferreira
Coorientador: Prof[a]. Dr[a]. Kelly Rosa Braghetto

Durante parte do desenvolvimento deste trabalho, o autor recebeu auxílio financeiro do CNPq

São Paulo, Maio de 2013

Uma abordagem transacional para o tratamento de exceções em processos de negócio

Esta é a versão original da dissertação elaborada pelo candidato Pedro Paulo de Souza Bento da Silva, tal como submetida à Comissão Julgadora.

"Maître est celui qui enferme une intelligence dans
le cercle arbitraire d'où elle ne sortira qu'à se rendre
à elle-même nécessaire. Pour émanciper un ignorant
il faut et il suffit d'être soi-même émancipé, c'est-à-dire
conscient du véritable pouvoir de l'esprit humain."
Jacques Rancière

Agradecimentos

Agradeço ao meu orientador, o Prof. João E. Ferreira, pelos conselhos, orientações, esclarecimentos e atenção dedicada a este trabalho. Agradeço, especialmente, a amizade que compartilhamos e que foi uma das motivações para os numerosos trabalhos que desenvolvemos desde minha época de iniciação científica até hoje. Agradeço por ele sempre ter sabido lidar pacientemente com minha impaciência.

Agradeço à Profª Kelly R. Braghetto, minha co-orientadora, que desde o início sempre contribuiu com sugestões e conselhos valiosos. Sobretudo, agradeço pela dedicação e disponibilidade, especialmente durante o processo de escrita de artigos e da dissertação, literalmente, vírgula a vírgula.

Agradeço ao Prof. Marco A. Gerosa, membro de minha banca de qualificação juntamente com os professores João E. Ferreira e Kelly R. Braghetto, pela revisão do texto e pelas sugestões que foram importantes para o desenvolvimento deste trabalho.

Não poderia deixar de agradecer novamente à Profª Kelly R. Braghetto e aos demais amigos que compartilharam comigo os laboratórios do IME-USP, sobretudo, Pedro "The Boss" Takecian, Bruno "O Culpado" Padilha, Daniel "Le Mouton" Cordeiro, Eduardo Cotrin e Marcela Garcia, pela amizade e pelos bons momentos que passamos juntos. Cabe aqui um agradecimento especial à Marcela Garcia pelas inúmeras discussões, reuniões e esclarecimentos que foram essenciais para o desenvolvimento deste trabalho.

Agradeço ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) por ter financiado parte deste trabalho.

Agradeço aos sempre presentes amigos do CEFET-SP (2001), que entremeavam os vários momentos de estresse decorrentes do mestrado com momentos de camaradagem, alegria e diversão; a Daniel "Bubs-nu" Uilgurson, Fábio "Bigodinho" Menino, Felipe "Bigode" Menino e Aline Sodré, pelos finais de semana dedicados à música, uma de minhas paixões; e aos amigos que me acompanharam durante a graduação e que ainda hoje continuam presentes, em especial, Rafael Lopes, Lucas Ikeda, Rafael Sato e Fábio Matsumoto.

Agradeço aos meus pais Marlene e Pedro Paulo, que sempre me apoiaram e sempre me incentivaram nos estudos. Agradeço também às minhas irmãs Luana e Mayra, à minha vovó Natália e também aos meus demais familiares, que sempre me deram força para não desistir de meus objetivos.

Finalmente, faço um agradecimento especial à Huana, companheira e amiga, que, além de ter sempre me apoiado e incentivado, também me aturou, pacientemente e com muito carinho, nos períodos mais estressantes do mestrado. Sou também muito grato a ela por sempre me convidar para fora dos muros das ciências exatas, através de textos e discussões que certamente ajudam a moldar meu posicionamento político, e, consequentemente, meu posicionamento acadêmico.

Resumo

Silva, P. P. S. B. **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, 2013.

Com o intuito de tornarem-se mais eficientes, muitas organizações – empresas, órgãos governamentais, centros de pesquisa, etc. – optam pela utilização de ferramentas de *software* para apoiar a realização de seus processos. Uma opção que vem se tornando cada vez mais popular é a utilização de sistemas de Gestão de Processos de Negócio (GPN), que são ferramentas genéricas, ou seja, não são específicas a nenhuma organização, altamente configuráveis e ajustáveis às necessidades dos objetos de atuação de cada organização. Uma das principais responsabilidades de um sistema de GPN é prover mecanismos de tratamento de exceções à execução de instâncias de processos de negócio. Exceções, se forem ignoradas ou se não forem corretamente tratadas, podem causar o aborto da execução de instâncias e, dependendo da gravidade da situação, podem causar falhas em sistemas de GPN ou até mesmo em sistemas subjacentes (sistema operacional, sistema gerenciador de banco de dados, etc.). Sendo assim, mecanismos de tratamento de exceções têm por objetivo resolver a situação excepcional ou conter seus efeitos colaterais garantindo, ao menos, uma degradação controlada (*graceful degradation*) do sistema. Neste trabalho, estudamos algumas das principais deficiências de modelos atuais de tratamento de exceções, no contexto de sistemas de GPN, e apresentamos soluções baseadas em Modelos Transacionais Avançados para contorná-las. Isso é feito por meio do aprimoramento dos mecanismos de tratamento de exceções da abordagem de modelagem e gerenciamento de execução de processos de negócio *WED-flow*. Por fim, estendemos a ferramenta *WED-tool*, uma implementação da abordagem WED-flow, através do desenvolvimento de seu gerenciador de recuperação de falhas.

Palavras-chave: Bancos de dados, tratamento de exceções, processos transacionais, WED-flow

Abstract

Silva, P. P. S. B. **A transactional approach to exception handling in business process.** Master's Dissertation - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2013.

With the aim of becoming more efficient, many organizations – companies, governmental entities, research centers, etc – choose to use software tools for supporting the accomplishment of its processes. An option that becomes more popular is the usage of Business Process Management Systems (BPM), which are generic tools, that is, not specific to any organization and highly configurable to the domain needs of any organization. One of the main responsibilities of BPM Systems is to provide exception handling mechanisms for the execution of business process instances. Exceptions, if ignored or incorrectly handled, may induce the abortion of instance executions and, depending on the gravity of the situation, induce failures on BPM Systems or even on subjacent systems (operational system, database management systems, etc.). Thus, exception handling mechanisms aim to solve the exceptional situation or stopping its collateral effects by ensuring, at least, a graceful degradation to the system. In this work, we study some of the main deficiencies of present exception handling models – in the context of BPM Systems – and present solutions based on Advanced Transaction Models to bypass them. We do this through the improvement of exception handling mechanisms from *WED-flow*, a business process modelling and instance execution managing approach. Lastly, we extend the *WED-tool*, an implementation of WED-flow approach, through the development of its failure recovery manager.

Keywords: Databases, exception handling, transactional processes, WED-flow

Sumário

Capítulo 1

Introdução

Com o intuito de tornarem-se mais eficientes, muitas organizações como empresas, órgãos governamentais, centros de pesquisa, etc., optam pela utilização de ferramentas de *software* para apoiar a realização de seus processos. Os maiores controle e precisão decorrentes da automatização, supervisão e execução de atividades que essas ferramentas proporcionam são as principais características que levam essas organizações a utilizarem esse tipo de abordagem.

Algumas organizações optam por desenvolver suas próprias ferramentas, construídas especialmente para seus domínios de atuação. Esse tipo de solução envolve, usualmente, altos custos financeiros, de pessoal e de tempo para ser implantado, além de ter custo elevado de manutenção, o que acaba limitando sua utilização. Uma alternativa que vem se tornando cada vez mais popular é a utilização de **Sistemas de Gestão de Processos de Negócio** (GPN), que são ferramentas genéricas, ou seja, que não são específicas a nenhuma organização, altamente configuráveis e que são ajustáveis às necessidades dos domínios de atuação de cada organização.

Processos de negócio são conjuntos de atividades interligadas que coletivamente realizam um objetivo em comum [BN97]. Eles são comumente utilizados para modelar atividades complexas em um conjunto de atividades mais simples, que podem ser manuais ou passíveis de automatização. **Sistemas de Gestão de Processos de Negócio** são sistemas de informação responsáveis por (i) auxiliar projetistas na tarefa de modelar processos de negócio; (ii) interpretar modelos e instanciar processos de negócio a partir deles; (iii) gerenciar a execução de instâncias de processos de negócios, e (iv) analisar instâncias de processos de negócio.

Exceções são situações não modeladas ou desvios entre o que foi planejado e o que acontece de fato durante a execução de uma instância de um processo de negócio [LSKM00]. O processo de identificação de uma exceção e a escolha e posterior execução de uma ação pertinente a ela é chamado de **tratamento de exceções** [Saa93]. Como parte da responsabilidade de se gerenciar execuções de instâncias de processos de negócio, o tratamento de exceções, que também é feito por sistemas de GPN, provê mecanismos automatizados ou baseados na interação com um usuário (*intervenção ad hoc*) para resolver a situação excepcional ou ao menos prover uma degradação controlada (*graceful degradation*) do sistema.

1.1 Definição do problema

Exceções, se forem ignoradas ou se não forem corretamente tratadas, podem causar o aborto da execução de instâncias de processos de negócio e, dependendo da gravidade da situação, podem

causar falhas nos Sistemas de GPN ou até mesmo em sistemas subjacentes (sistema operacional, sistema gerenciador de banco de dados, etc.). Problemas como esses afetam a organização que faz uso do sistema de GPN uma vez que processos de negócio normalmente são de longa duração, ou seja, demoram minutos, horas, dias, etc. para terminar, e o prejuízo associado ao aborto da execução da instância e ao custo de se reiniciá-la pode ser alto. Veja que os termos prejuízo e custo não estão associados, obrigatoriamente, a perdas financeiras visto que, apesar da palavra "negócio", processos de negócio não se restringem a usos relacionados à negócios (finanças, administração, etc.), podendo ser aplicados em qualquer domínio.

Não obstante as possíveis graves consequências de exceções não tratadas, ainda é bastante usual encontrar sistemas de GPN que não implementam nenhuma forma de tratamento de exceções ou que o fazem de maneira deficiente. À vista disso, pode-se dizer que, à parte de problemas de mal funcionamento do sistema de GPN (como corrompimento do disco, vírus, etc.), falhas relativas a exceções não tratadas são causadas, **unicamente**, pela ausência de mecanismos capazes de identificar exceções não descritas, no momento da modelagem, pelo projetista de processos de negócio. Além disso, é importante observar que, apesar do grande número de sistemas de GPN disponíveis atualmente, a maioria deles não é capaz de identificar esse tipo de exceção [Ant11].

Tão importante quanto prover o tratamento de exceções é provê-lo de forma clara e precisa. Reichert et al. [RDB03] afirmam que o tratamento de uma exceção não pode ser mais complicado ou tomar mais tempo do que fazer uma ligação telefônica para que a pessoa certa resolva diretamente o problema. Mesmo que os autores tenham exagerado na comparação, de fato, existem sistemas de GPN que dificultam o trabalho dos projetistas e de seus usuários em geral.

Processos de negócio evoluem e é necessário que isso seja refletido em seus modelos. Entretanto, são raros os casos em que essa necessidade é satisfeita de forma completa, ou seja, quando é permitido ao projetista alterar os modelos dos processos de negócio em outras etapas da execução de sistemas de GPN além da etapa de modelagem, como, por exemplo, durante a etapa de execução de instâncias ou durante o tratamento de exceções. Existem também linguagens de modelagem cuja manutenção é difícil de ser feita devido a uma série de motivos, que variam de sintaxe obscura à falta de expressividade da linguagem (dificuldade em se representar condições, laços, variáveis, etc.).

Os fatores que discutimos nesses últimos parágrafos descrevem algumas das principais deficiências relacionadas ao tratamento de exceções encontradas, atualmente, em sistemas de GPN. Existem vários grupos de pesquisa estudando formas de se enfrentar esses problemas, mas, apesar das importantes contribuições propostas por eles, ainda há vários desafios a serem superados.

1.2 Proposta de solução

Conforme discutimos anteriormente, os sistemas de gestão de processos de negócio atuais apresentam diversas deficiências no que tange às suas formas de tratamento de exceções. Entre as mais graves, podemos citar a ausência de mecanismos para o tratamento de exceções, a dificuldade de se evoluir os processos de negócio, falta de expressividade das linguagens de modelagem e dificuldade de manutenção dos processos.

A **WED-flow** [FTMP10, FBTP12, Mar12, GSBF12] é uma abordagem de modelagem e controle de execução de processos de negócio que vem sendo elaborada pelo grupo DATA IME-USP [DIa]. Ela se apoia sobre conceitos de Modelos Transacionais Avançados (Capítulo 2), modelagem evolutiva

e intervenção *ad hoc* e mostrou-se terreno fértil para a solução de algumas dessas deficiências, como discutiremos a seguir.

- **Exceções não tratadas**
  A abordagem WED-flow é concebida de forma a detectar vários tipos de exceções. Além das exceções cujos tratamentos podem ser definidos previamente no modelo dos processos de negócio, conhecidas como exceções esperadas, a abordagem permite a detecção de exceções que não constam no modelo, chamadas de exceções não esperadas ou exceções verdadeiras (ver Seção 2.3.5). Assim, com a possibilidade de detecção desses dois tipos de exceção, é possível, por meio do desenvolvimento de mecanismos de tratamento de exceções, resolver a situação excepcional ou, ao menos, permitir uma degradação controlada do sistema.

- **Precisão no tratamento de exceções**
  Exceções que não constam no modelo de processos de negócio não podem, na maioria das vezes, ser tratadas de forma automática. Para tratar esse tipo de exceção, a abordagem WED-flow faz uso de uma estratégia *ad hoc*, que interrompe a execução da instância em que foi detectada a exceção e, então, permite que um administrador intervenha para resolver diretamente a situação excepcional. Nesse momento, o administrador de sistema tem a sua disposição informações detalhadas sobre a execução da instância, que servirão de base para que ele tome a melhor decisão e escolha o mecanismo de tratamento de exceções mais adequado.

- **Evolução do modelo**
  Na abordagem WED-flow, a linguagem de modelagem utilizada para descrever os processos de negócio é, por definição, evolutiva, ou seja, é permitido ao projetista alterar o modelo dos processos de negócio tanto durante a modelagem inicial quanto durante a execução de instâncias ou de mecanismos de tratamento de exceções. A forma como a linguagem foi concebida também permite o reaproveitamento de definições do modelo, o que facilita a manutenção dele por projetistas e administradores de sistema.

- **Expressividade da linguagem de modelagem**
  A ferramenta WED-tool [GSBF12] é um protótipo de sistema de gestão de processos de negócio que implementa a abordagem WED-flow. A especificação da linguagem de modelagem de processos de negócio utilizada por ela permite que o projetista defina as atividades de um WED-flow por meio da linguagem de programação *Ruby*, o que garante maior liberdade e maior poder de expressão.

As características listadas acima são suficientes para indicar que a abordagem WED-flow é terreno fértil para o desenvolvimento de mecanismos que contribuam para a solução das diversas deficiências que atingem o tratamento de exceções no âmbito de sistemas de GPN, conforme discutimos previamente. Dessa forma, nossa proposta é estudar os principais Modelos Transacionais Avançados e ferramentas consagradas de GPN e Workflow, e aplicar esse conhecimento no aprimoramento dos mecanismos de tratamento de exceções da abordagem WED-flow. Para tal, refinamos os mecanismos de tratamento de exceções já existentes e desenvolvemos novos, de modo a aumentar a abrangência e variedade de opções para o tratamento de exceções. Além disso, para analisar a via-

bilidade prática de nossas propostas, implementamos o Gerenciador de Recuperação da ferramenta WED-tool, um protótipo de sistema de GPN que implementa a abordagem WED-flow.

Consideramos que com o auxílio da WED-flow, uma abordagem transacional de tratamento de exceções que se apoia sobre conceitos de consistência transacional, modelagem evolutiva e intervenção *ad-hoc*, é possível contornar várias das deficiências que atingem o tratamento de exceções de processos de negócio e contribuir com avanços importantes nesse sentido.

1.3 Objetivos

Neste trabalho, nosso principal objetivo é o aprimoramento e implementação do modelo de tratamento de exceções para a modelagem e controle de execução de processos de negócio, tendo como referência a abordagem **WED-flow**. Nosso principal desafio é sanar ou contornar algumas das deficiências que discutimos ao longo deste capítulo. De forma mais sistemática, temos o seguinte:

- **Aprimoramento do modelo de tratamento de exceções da abordagem WED-flow:** Conceitos relacionados ao tratamento de exceções da abordagem WED-flow foram definidos de forma sucinta e inicial em trabalhos anteriores [FTMP10, FBTP12]. Neste trabalho, nosso objetivo é revisar os conceitos já existentes, aprimorá-los e, conforme a necessidade, contribuir com novos.

- **Implementação do modelo de tratamento de exceções da abordagem WED-flow:** A ferramenta **WED-tool** [GSBF12] é um protótipo de um sistema de gestão de processos de negócio que implementa a abordagem WED-flow e que está sendo desenvolvido por pesquisadores do grupo DATA IME-USP. Por considerarmos que a implementação de nossa proposta de tratamento de exceções indicaria a viabilidade prática de nossa solução, traçamos como objetivo para este trabalho o desenvolvimento do **Gerenciador de Recuperação** dessa ferramenta, que é o módulo responsável por prover o tratamento de exceções às execuções de instâncias de processos de negócio gerenciadas pela WED-tool.

1.4 Organização do trabalho

No Capítulo 2 introduziremos diversos conceitos necessários para o embasamento e compreensão de nossa proposta, como transações longas, Modelos Transacionais Avançados, processos de negócio, tratamento de exceções e a abordagem WED-flow. No Capítulo 3 contextualizamos os principais trabalhos relacionados e suas contribuições à gestão de processos de negócio e ao tratamento de exceções. No Capítulo 4, apresentamos nossa proposta de modelo de tratamento de exceções assim como as definições que a fundamentam. No Capítulo 5, descrevemos os detalhes do módulo de gerenciamento de recuperação da ferramenta WED-tool que implementa nossa abordagem de tratamento de exceções. No Capítulo 6 discutimos os resultados deste trabalho e suas possíveis extensões e trabalhos futuros.

Capítulo 2

Fundamentos

Neste capítulo apresentaremos alguns conceitos essenciais para a melhor compreensão dos próximos capítulos e, em especial, de nossa proposta (Capítulo 4).

2.1 Transações

Para definirmos o conceito de transação, é necessário especificar, antes de tudo, qual modelo transacional será utilizado. Existem vários modelos transacionais diferentes, mas, para os fins deste trabalho, o que melhor se adequa e que também é o modelo mais utilizado e conhecido é o modelo transacional usado em bancos de dados [Alo05].

No modelo transacional de bancos de dados, uma transação é, em resumo, uma função que leva o banco de dados de um **estado** para outro **estado**, preservando quatro propriedades importantes, conhecidas como propriedades **ACID** [GR93, BN97], responsáveis por garantir a correção desses estados. Por estado, entende-se os valores de todos os dados e parâmetros de sistema relevantes para o banco de dados e é garantido que ele só possa ser alterado por meio de uma transação. A seguir aprofundaremos um pouco mais esses conceitos.

2.1.1 Propriedades ACID

O acrônimo ACID origina-se das iniciais das propriedades de **A**tomicidade, **C**onsistência, **I**solamento e **D**urabilidade, as quais serão apresentadas a seguir.

Atomicidade

Uma transação é uma unidade atômica de processamento, ou seja, que é realizada integralmente ou não é realizada de modo algum [EN05]. Não é possível que haja execução parcial da transação e, consequentemente, não é possível que haja resultados parciais. Essa propriedade caracteriza o funcionamento de um sistema de recuperação que, na impossibilidade da conclusão com sucesso de uma transação, seja capaz de desfazer as possíveis modificações que a transação tenha feito. Chamamos de **commit** ou **confirmação** à conclusão com sucesso de uma transação e chamamos de **rollback** ou **aborto** o desfazimento das operações feitas durante a execução de uma transação que apresentou falhas.

Consistência

Uma transação é consistente se a sua plena execução levar o banco de dados de um estado consistente para outro estado também consistente [EN05]. Por consistente, entende-se a satisfação das restrições de integridade do banco de dados, como unicidade de chaves primárias, correção das referências de chaves estrangeiras, satisfação dos tipos das tabelas, etc.

Isolamento

A propriedade de isolamento garante que uma transação será executada **como** se estivesse sendo executada sozinha, ou seja, sem outras transações concorrentes. Em outras palavras, o isolamento permite que diversas transações sejam executadas paralelamente de modo a preservar a propriedade de consistência [BN97]. O escalonamento das transações é feito pelos mecanismos de controle de concorrência do sistema gerenciador de bancos de dados.

Durabilidade

A propriedade de durabilidade garante que as alterações resultantes da execução de uma transação devem ser persistidas em alguma mídia de armazenamento que seja estável, em outras palavras, algum tipo de armazenamento que sobreviva à falha do sistema operacional. A durabilidade pode ser vista como uma propriedade que garante que o sistema vai se manter consistente após a alteração do estado do banco de dados pela transação.

Podemos concluir que o objetivo das propriedades ACID é garantir a consistência do banco de dados e, consequentemente, a definição de transação pode ser entendida como a execução de um programa que leva o banco de dados de um estado **consistente** para outro estado **consistente**.

2.2 Transações longas

Durante muito tempo, algumas hipóteses implícitas à utilização do modelo transacional de bancos de dados eram tidas como garantidas nas aplicações que faziam uso desse modelo [FF00]. Podemos citar as seguintes como exemplo:

(1) **Transações curtas:** O modelo transacional de banco de dados foi concebido para tratar transações com duração da ordem de milissegundos ou segundos, que são chamadas de transações simples ou transações curtas. Dizemos que uma transação é longa quando ela tem duração da ordem de minutos, horas, dias, etc.

(2) **Sequencialidade:** Uma transação sempre submete operações aos bancos de dados usando, como **única** fonte de dados, os parâmetros passados à transação no início do processamento ou o próprio banco de dados. Assim, o processamento da transação não depende da intervenção humana para prosseguir, diferentemente do que ocorre nas **transações interativas**, em que há algum tipo de interação com um usuário.

(3) **Homogeneidade:** Bancos de dados que fazem parte de um mesmo sistema distribuído devem compartilhar os protocolos utilizados para se comunicar e para executar operações distribuídas.

(4) **Disponibilidade:** Todos os participantes de uma transação, em um sistema de bancos de dados distribuídos, estão disponíveis durante toda a transação.

Com o passar dos anos, as aplicações tornaram-se cada vez mais complexas e passaram a não mais satisfazer essas hipóteses, criando assim novos desafios na área de banco de dados. Nosso interesse está ligado, principalmente, às violações das duas primeiras hipóteses citadas acima, que se relacionam diretamente com as consequências da utilização de transações longas.

Na prática, as consequências da utilização de uma transação longa podem ser inviáveis para uma aplicação. Por exemplo, os recursos utilizados por uma transação longa ficam bloqueados até o fim de sua execução, podendo causar a paralisação de várias outras transações durante muito tempo. Outro exemplo de inviabilidade é o possível custo computacional de um aborto ou reexecução de uma transação longa que estava em execução havia horas, dias, semanas, etc.

Para contornar essas consequências, é necessário encontrar formas de se flexibilizar as propriedades ACID sem, no entanto, perder as garantias de consistência transacional providas por esse modelo. Esse é o objetivo dos modelos conhecidos como **Modelos Transacionais Avançados** que veremos a seguir: transações com *savepoints*, transações aninhadas, transações multiníveis, transações encadeadas, modelo SAGA e a semiatomicidade.

2.2.1 Transações com savepoints

O aborto de uma transação longa tem um efeito negativo não só porque o desfazimento de operações pode ser custoso, mas principalmente porque pode ser que a transação seja reexecutada futuramente e boa parte da execução tenha que ser refeita. Suponha, por exemplo, que uma transação que está sendo executada há 2 dias é interrompida por uma queda de energia. Quando a energia voltar e a transação for reexecutada, ela demorará mais 2 dias para chegar ao mesmo ponto em que estava no momento da falha. Ou seja, a transação terá levado 4 dias para executar o trecho em questão. Essa situação pode ser inviável.

Para solucionar problemas como esse, foi proposto o modelo de **transações com *savepoints*** [GR93] que consiste, basicamente, em salvar em alguma mídia permanente "pontos" intermediários da execução que, em caso de falha, poderão servir de ponto de partida para reexecução.

A definição da quantidade de *savepoints* e suas posições relativas à execução é feita pelos projetistas da aplicação. Se alguma falha for detectada durante a execução de uma transação longa, cabe à aplicação decidir se um desfazimento parcial será executado, no caso em que algum *savepoint* tenha sido definido anteriormente, ou se um aborto faz-se necessário. No primeiro caso, tudo o que a execução fez entre o momento do aborto e o *savepoint* será desfeito e, a partir desse último, a transação poderá continuar sua execução. No segundo caso, ocorrerá um aborto normal, como o aborto de uma transação curta comum. Caso haja uma queda brusca do sistema, no momento de sua reinicialização, o sistema transacional pode decidir por continuar a execução da transação longa a partir do último *savepoint* ou, como explicado acima, reiniciar a transação longa do zero.

É importante salientar que os *savepoints* e, consequentemente, o estado intermediário de uma transação longa não podem ser acessados por nenhuma outra transação, longa ou curta. O modelo de transações com *savepoints* respeita as propriedades ACID, logo, as alterações realizadas por uma transação, longa ou curta, só são externalizadas após o seu *commit*. Assim, percebe-se que a utilização desse modelo por si só resolve apenas parte do problema das transações longas uma vez

que os recursos utilizados pela transação continuam bloqueados até o fim de sua execução, embora, no caso de interrupção da execução da transação, seja possível reaproveitar parte do trabalho já realizado pela transação por meio dos *savepoints*.

2.2.2 Transações aninhadas

Transações aninhadas podem ser entendidas como uma generalização do modelo de *savepoints*: *savepoints* permitem a organização de uma transação longa em uma sequência de ações que podem ser desfeitas individualmente e transações aninhadas formam uma hierarquia de **peças** de trabalho [GR93]. Existem diversas definições de transações aninhadas mas, neste texto, vamos nos ater à mais utilizada delas, descrita por Moss et al. [Mos81].

O modelo de transações aninhadas é uma forma de se organizar uma transação longa em uma árvore de subtransações. Cada nó dessa árvore, com exceção à sua raiz, que é chamada de **transação primordial**, é uma subtransação. Chamamos de subtransação **filha** à subtransação que sucede imediatamente uma outra subtransação na estrutura da árvore e, analogamente, chamamos de subtransação **mãe** à subtransação que precede imediatamente uma outra. Duas ou mais subtransações são **irmãs** se elas compartilham a mesma mãe. As únicas subtransações que executam de fato algum programa são as transações de **acesso** que se localizam nas folhas da árvore. As demais, que são chamadas de transações **internas**, têm por objetivo somente a inicialização de outras subtransações e o gerenciamento de *commits* e abortos de sua subárvore. Os *commits* efetuados pelas subtransações são condicionais à execução do *commit* de suas subtransações mãe. No caso de aborto de uma subtransação, todas as suas filhas também são abortadas e cabe à subtransação mãe decidir se ela também se abortará ou se ela tentará executar alguma outra ação, como reexecutar o nó que falhou, criar uma nova transação, ou ignorar a falha, dependendo da implementação do modelo. De forma geral, uma transação aninhada e suas subtransações só efetuam de fato um *commit* quando a transação primordial efetua *commit*. A Figura 2.1 ilustra o funcionamento de uma transação aninhada.

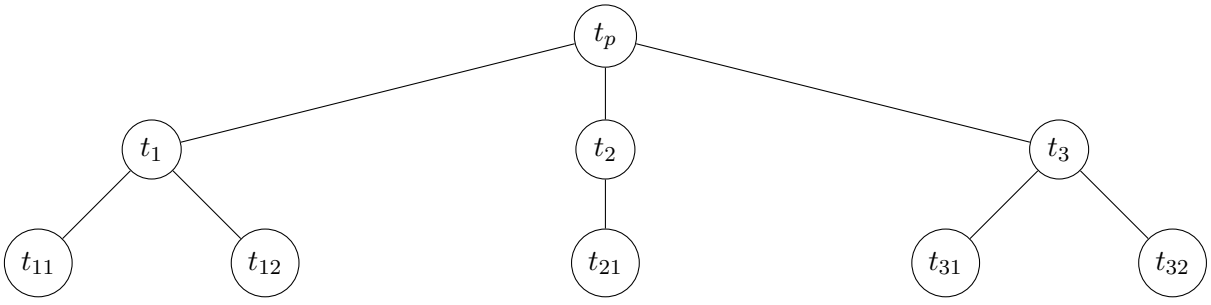


Figura 2.1: *Exemplo de uma transação aninhada onde t_p é a transação primordial e as únicas subtransações sendo executadas de fato são as subtransações t_{11}, t_{12}, t_{21}, t_{31} e t_{32}*

É interessante observar que, embora toda transação aninhada satisfaça as propriedades ACID, esse não é o caso de suas subtransações. É possível que uma subtransação que já tenha executado um *commit* seja abortada por sua subtransação mãe. Se os efeitos da subtransação filha fossem externalizados, isso caracterizaria uma violação das propriedades de atomicidade e durabilidade, entretanto, o gerenciamento das subtransações é tratado internamente pela transação aninhada e, consequentemente, não há reflexo de subtransações abortadas no estado do banco de dados.

As transações aninhadas e suas extensões tornaram a modelagem de transações longas mais

intuitiva e organizada, entretanto, ainda não resolvem o problema do bloqueio de recursos por longos períodos de tempo, o que, como já foi discutido anteriormente, pode resultar em consequências indesejáveis.

2.2.3 Transações encadeadas

Transações encadeadas, assim como transações aninhadas, são concebidas como uma generalização de *savepoints* [GR93]. O modelo baseia-se na subdivisão de uma transação longa em várias subtransações ligadas entre si de forma encadeada. A execução de uma subtransação s_1 é condicionada à confirmação de sua subtransação precedente s_0. As ações de uma subtransação são refletidas no estado do banco de dados no momento do *commit*. O preço pago por isso é a **impossibilidade** de se abortar uma subtransação já confirmada. Uma característica importante desse modelo é que o *commit* de uma subtransação e a inicialização da subtransação que a sucede é feita atomicamente. Dessa forma, parte do contexto de uma subtransação pode ser repassado à próxima sem que haja interferência de alguma outra transação do sistema. A Figura 2.2 ilustra um exemplo de uma sequência de transações encadeadas.

O modelo de transações encadeadas permite a concepção de transações longas de forma menos burocrática do que no modelo de transações aninhadas e ainda oferece uma solução para o problema do bloqueio de recursos. No entanto, sua viabilidade, devido a ausência de uma maneira de se abortar subtransações, é controversa.

Figura 2.2: *Exemplo de uma sequência de transações encadeadas.*

2.2.4 Modelo SAGA

O modelo SAGA, proposto por Garcia-Molina et al. [GMS87], utiliza conceitos de transações encadeadas como base de sustentação. Uma **SAGA** é uma sequência de transações curtas, chamadas de **passos SAGA**, cuja execução é condicionada à confirmação do passo SAGA anterior. O *commit* de um passo SAGA é refletido no estado de dados independentemente dos outros passos SAGA e, por isso, em caso de falha, é impossível efetuar o desfazimento (*rollback*) de toda a SAGA (veja Seção 2.2.5). Para contornar esse problema, cada passo SAGA tem uma transação compensatória associada, que tem por objetivo revertê-lo semanticamente em caso de falha em sua execução. A Figura 2.3 ilustra o exemplo de uma SAGA.

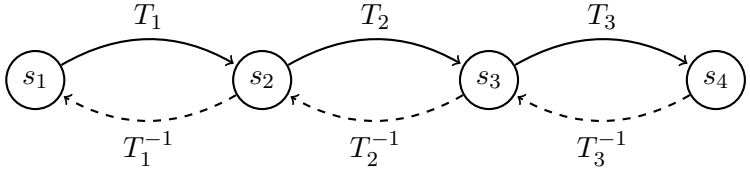


Figura 2.3: *Exemplo de uma SAGA. Os nós s_i são os estados do banco de dados, as transações T_i são os passos SAGA e as transações T_i^{-1} são suas respectivas compensações.*

O modelo SAGA é uma solução simples e completa que, embora tenha sido publicada em

1987, ainda é um conceito muito utilizado, pois oferece uma solução para o problema dos recursos bloqueados e, consequentemente, das transações bloqueadas por muito tempo.

2.2.5 Transações multiníveis

O modelo de transações multiníveis, assim como o modelo de transações aninhadas, concebe a organização e divisão de uma transação longa em um hierarquia de subtransações, que é representada por uma árvore. As diferenças entre as duas modelagens são que subtransações do modelo multiníveis podem executar programas mesmo não estando nos nós folhas e, quando uma subtransação efetua um *commit*, os efeitos são imediatamente refletidos no estado do banco de dados. Em outras palavras, o *commit* de uma subtransação não é mais condicionado ao *commit* de sua subtransação mãe. Dessa forma, a propriedade de durabilidade é respeitada e, consequentemente, podemos dizer que cada subtransação de uma transação multiníveis é ACID.

Quanto ao aborto de uma subtransação, como as suas subtransações filhas já foram confirmadas no estado do banco de dados, o desfazimento não é mais possível, visto que o modelo define o objetivo de manter a execução das subtransações atômicas [WH93]. Para contornar essa situação, o modelo faz uso de **transações compensatórias**, as quais, como vimos anteriormente, revertem semanticamente as transações em vez de desfazê-las. O modelo supõe a associação de uma transação compensatória a cada subtransação. Assim, sempre que houver o cancelamento da execução de uma subtransação t_i, as subtransações filhas de t_i já confirmadas executarão suas transações compensatórias associadas uma vez que elas não podem ser desfeitas. As subtransações filhas de t_i que não confirmaram serão desfeitas.

Percebe-se então que transações multiníveis podem ser entendidas como generalizações de transações aninhadas. A utilização de compensações solucionam o problema dos recursos bloqueados e, consequentemente, das transações bloqueadas por muito tempo, entretanto, a complexidade inerente à organização de transações em árvore se mantém.

2.2.6 Semiatomicidade

A semiatomicidade [ZNBB94] estende o modelo SAGA visando sanar uma de suas principais deficiências, que é a dificuldade de, em algumas situações, definir-se a compensação de um passo SAGA. Especialmente em ambientes concorrentes, os efeitos colaterais de uma compensação podem afetar a execução de outras SAGAs. Para enfrentar esse problema, a semiatomicidade define uma transação longa, comparável a uma SAGA, que é dividida em várias subtransações, também chamadas de passos, comparáveis aos passos SAGA, que são executadas de maneira encadeada, até o último passo da transação longa. Os passos podem ser dos seguintes tipos:

- **Compensável:** Se o passo tem uma compensação associada, como no modelo SAGA, dizemos que ele é compensável.
- **Reexecutável:** Se é garantido que um passo será executado com sucesso após um número finito de tentativas, dizemos que ele é reexecutável.
- **Pivô:** Quando um passo não é compensável e nem reexecutável ele é chamado de pivô. Por não ser possível efetuar um *rollback* de toda a transação longa e, também, por pivôs não possuírem compensações, em caso de falha, esse tipo de passo é sempre o ponto de partida para caminhos

alternativos de execução. Ao menos um desses caminhos alternativos de execução precisa ter terminação garantida.

Assim, além dos passos compensáveis, o projetista pode definir passos reexecutáveis e, em especial, passos pivôs, que definem caminhos alternativos de execução. Dessa forma, em caso de falha durante a execução de algum passo, ou a transação é compensada (completamente ou parcialmente), ou a transação será concluída pela execução de um caminho alternativo com a possível reexecução de algum passo nesse processo.

É inegável que a semiatomicidade flexibiliza a forma de se tratar transações longas, entretanto, a carga extra de complexidade que recai sobre os projetistas durante a modelagem é grande e, em casos mais complexos, pode inviabilizar a concepção de um modelo correto, mesmo com o auxílio de verificadores automáticos de correção [Alo05].

2.3 Processos de negócio

Processos de negócio são conjuntos de atividades interligadas que coletivamente realizam um objetivo em particular [BN97]. Eles são comumente utilizados para modelar atividades complexas como um conjunto de atividades mais simples que podem ser manuais ou passíveis de automatização.

Uma **instância** de um processo de negócio é um caso específico de execução de um processo de negócio. Cada instância representa uma *thread* separada de execução do processo que pode ser controlada independemente e que tem seu próprio estado interno e identidade externa [Spe].

A **definição** de um processo de negócio, também chamada de **modelo** de um processo de negócio, descrita por meio de uma **linguagem de modelagem de processos**, contém toda a informação necessária para se criar e executar instâncias desse processo. Uma possível maneira de agrupar essa informação é descrita na tese de doutorado de N. Russell [RoTFoIT07]:

- **Fluxo de controle:** descreve a estrutura do modelo do processo em termos das atividades que o consistituem, a maneira como essas atividades são implementadas e as interconexões entre elas.
- **Dados:** descreve como os dados são definidos e utilizados durante a execução de instâncias do processo de negócio.
- **Recursos:** descreve todo o contexto organizacional de execução de uma instância do processo e a forma pela qual itens de trabalho individuais podem ser atribuídos a pessoas ou mecanismos para execuções subsequentes.
- **Tratamento de exceções:** aborda a especificação de estratégias de tratamento de exceções em resposta a eventos esperados ou inesperados que podem ocorrer durante a execução de uma instância.

Existem dois formalismos predominantes nos quais linguagens de modelagem de processos são baseadas: **formalismo baseado em grafos** e **formalismo baseado em regras**. Uma **linguagem de modelagem baseada em grafos** tem suas raízes na teoria dos grafos e variantes, enquanto que uma **linguagem de modelagem baseada em regras** apoia-se na lógica formal [LS07]. Veremos a seguir cada um desses tipos de linguagem de modelagem em mais detalhes:

- Em **linguagens de modelagem baseada em grafos** as definições do modelo do processo de negócio são especificadas de forma visual e explícita: atividades são representadas por nós e o fluxo de controle e as dependências entre atividades, por arcos. Além disso, essas linguagens normalmente se baseiam em modelos formais, como Redes de Petri [Pet62], os quais permitem que a correção da definição seja verificada antes mesmo da criação das instâncias (é possível detectar travamentos, por exemplo). No entanto, algumas vezes é difícil ou muito trabalhoso definir a ordem das atividades e suas interconexões especialmente em casos em que o processo de negócio é muito complexo. Em outros casos, o término da execução de uma atividade pode não ser uma condição suficiente para iniciar outra atividade. Em resumo, apesar de o processo de modelagem ser mais intuitivo e apesar do forte apelo visual, o modelo resultante é, usualmente, complexo, pois a quantidade de detalhes que precisa ser descrita pelo projetista tende a ser grande. A Figura 2.4 ilustra um exemplo de um processo de negócio modelado com o auxílio da linguagem de modelagem baseada em grafos BPMN [RoTFoIT07].

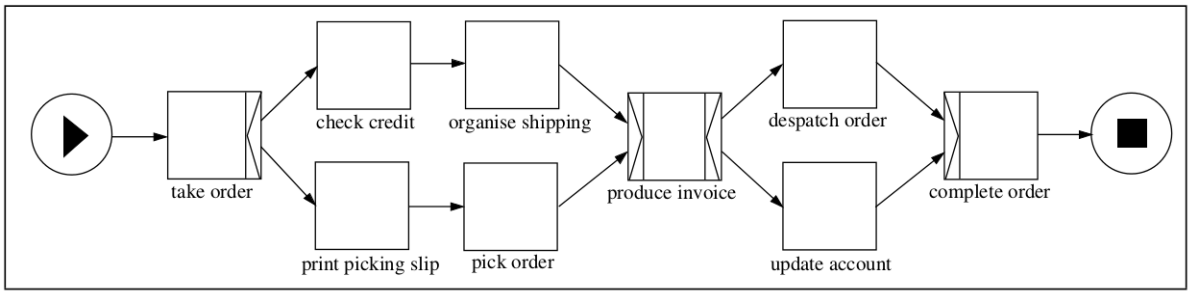


Figura 2.4: *Exemplo de um processo de negócio representado por uma linguagem de modelagem baseada em grafos. (Retirado de Russel et al. [RvdAtH])*

- **Linguagens de modelagem baseadas em regras** abstraem a lógica do processo em um conjunto de regras, as quais definem precondições e poscondições de execução às atividades. Dessa forma, a ordem de execução das atividades é definida de acordo com as regras associadas a cada uma delas, como no ECA (*Event-Condition-Action* [DGG95]), por exemplo. Os efeitos colaterais de alterações na definição do processo (remoção de uma atividade, por exemplo) são menores se comparados às linguagens de modelagem baseadas em grafos, pois não há dependência explícita entre as atividades. Isso garante maior versatilidade ao processo de modelagem, entretanto, essa mesma característica acaba por impedir que verificações prévias da correção da definição do processo de negócio sejam realizadas, por só se saber quais atividades serão executadas e em qual ordem no momento da execução. Outro problema é a maior facilidade de o projetista perder o controle do modelo dos processos de negócio [vdAPS09] uma vez que os relacionamentos entre as regras não são explicitamente definidos.

O Trecho de código 2.1 ilustra parte de um processo de negócio modelado pela linguagem de modelagem baseada em regras Chimera Exc [CCPP99].

```
define trigger lateCarReturn

  events modify(carRental.returnTime)

  condition carRental(C), ocurred(modify(carRental.returnTime), C),
            C.returnTime > old(C.returnTime)

  actions notify(C.responsible, ''Late return for car'' + C.bookedCarPlate)

end
```

Trecho 2.1: *Exemplo da linguagem de modelagem baseada em regras para tratamento de exceções Chimera Exc (Retirado de Casati et al. [CCPP99]).*

2.3.1 Workflows

Um conceito importante associado a processos de negócio é o de **Workflow**. Apesar de alguns autores o definirem como um sinônimo de processo de negócio [BN97], a *Workflow Management Coalition* define workflow como a automatização total ou parcial de um processo de negócio [Spe].

Essa automatização é descrita na definição do processo de negócio. É importante salientar que tarefas não automatizáveis presentes na definição do processo de negócio não podem fazer parte de um workflow e que no caso em que um processo de negócio tenha somente atividades automatizáveis, o workflow resultante será equivalente a esse processo. Analogamente às instâncias de processos de negócio, chamamos de instância de workflow o caso específico de execução de um determinado workflow.

2.3.2 Sistemas de gerenciamento de workflows

Um **sistema de gerenciamento de workflows** é um sistema que define, cria e gerencia a execução de workflows através do uso de programas de computador, executado por um ou mais mecanismos de workflow (*workflow engines*), que é capaz de interpretar a definição de processos de negócio, interagir com atores do workflow e, quando requisitado, utilizar ferramentas computacionais e aplicações [Spe]. Em resumo, é o sistema de gerenciamento de workflows o responsável por dar apoio à criação e execução de instâncias de workflows.

2.3.3 Sistemas de gestão de processos de negócio

Sistemas de gestão de processos de negócio são sistemas que apoiam processos de negócio por meio de procedimentos, técnicas e programas para à modelagem, execução, controle e análise de processos operacionais envolvendo pessoas, organizações, aplicações, documentos e outras fontes de informação [vdAHW03]. É importante salientar que sistemas de gerenciamento de workflows fazem parte de sistemas de gestão de processos de negócio.

2.3.4 Processos de negócio transacionais e workflows transacionais

A consistência transacional é uma característica interessante a ser adicionada à execução de instâncias de processos de negócio e de workflows. No entanto, apesar de, em algumas ocasiões, ser possível a execução de todos os passos de uma instância de um processo de negócio ou de um workflow dentro de uma única transação de banco de dados, os possíveis efeitos colaterais (ver Seção 2.2) de tal execução poderiam inviabilizá-la [BN97]. Uma solução é utilizar Modelos Transacionais Avançados para flexibilizar algumas propriedades transacionais de forma a diminuir os efeitos colaterais resultantes das restrições impostas pelas propriedades ACID, sem, entretanto, perder a garantia de consistência que elas proporcionam [BN97, Alo05]. Processos de negócio concebidos conforme essa solução são conhecidos como processos transacionais. De forma análoga, workflows que são concebidos com o auxílio das técnicas de Modelos Transacionais Avançados são chamados de **workflows transacionais** [SR93].

2.3.5 Exceções

Uma **exceção** é uma situação não modelada por um sistema de informação ou um desvio entre o que foi planejado e o que acontece de fato [LSKM00].

Exceções em processos de negócios se diferenciam de exceções em linguagens de programação no sentido de que essas últimas têm como principal função detectar condições de erros e evitar a terminação anormal de programas, visando a recuperação de uma situação problemática ou ao menos uma degradação controlada (*graceful degradation*). No caso de processos de negócio, as exceções operam em um nível mais alto de abstração dado que elas não têm como função proteger os sistemas de GPN de falhas sistêmicas ou de programação. Por outro lado, espera-se que exceções nesse contexto permitam a especificação de um comportamento complexo que é anômalo em relação a semântica "normal" dos processos de negócios, mas ainda assim orientado pela semântica da aplicação [CCPP99].

Visando conter ou amenizar as possíveis consequências de exceções, que incluem abortos de instâncias de processos de negócio executadas por longos períodos (horas, dias, etc.), geração de resultados ou dados incorretos, etc., muitos sistemas de GPN oferecem mecanismos de **tratamento de exceções** em suas implementações. O tratamento de exceções é o processo de identificação de uma situação de exceção e a escolha e posterior execução de uma ação pertinente a ela [Saa93].

Para entender melhor como o tratamento de exceções funciona, é interessante esclarecer a natureza das exceções e, com isso, analisar seus possíveis tipos. Uma forma de se caracterizar exceções é por meio da análise do espaço de conhecimento de uma exceção [LSKM00]. Esse espaço é formado por três dimensões chamadas de "Conhecido", "Detectável" e "Solucionável" e cada ponto nesse espaço é um ponto de exceção (Fig. 2.5).

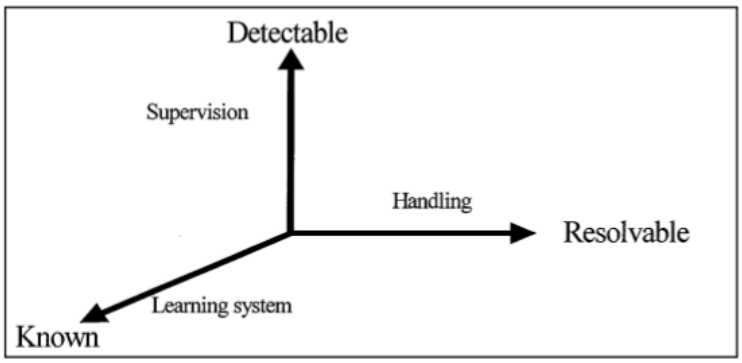


Figura 2.5: *Representação do espaço de conhecimento de uma exceção (retirada de [LSKM00]).*

- **Dimensão "Conhecido"**: Essa dimensão indica se uma exceção é conhecida ou não. Em resumo, uma exceção é conhecida se os projetistas do processo de negócio sabem da possibilidade de sua ocorrência. Exceções que são desconhecidas também são chamadas de **exceções verdadeiras** [Saa93] ou **exceções não esperadas** [EL95]. Perceba que o fato de uma exceção ser conhecida ou não independe do fato de ela ser detectável ou solucionável, como veremos a seguir.

- **Dimensão "Detectável"**: Essa dimensão indica se o sistema de GPN tem os mecanismos necessários, descritos na definição do processo de negócio, para se detectar uma exceção no momento em que ela ocorre. Para que sistemas de GPN possam realizar o tratamento de exceções, é imprescindível que elas possam ser detectadas. Se uma exceção não é detectável, então, mesmo que ela seja conhecida, ela não pode ser tratada e isso pode resultar em efeitos colaterais indesejados, como discutimos acima.

- **Dimensão "Solucionável"**: Essa dimensão indica se uma situação de exceção pode ser solucionada ou não pelo sistema de GPN no momento de sua ocorrência. Exceções não detectáveis não são solucionáveis visto que o sistema não tem meios de identificar suas ocorrências.

Através da análise das dimensões que caracterizam uma exceção, de acordo com o espaço de conhecimento discutido acima, podemos descrever o tratamento de exceções em sistemas de GPN de maneira mais precisa.

Como vimos, um pré-requisito indispensável para que uma exceção seja tratada é que ela seja detectável, ou seja, que tenha a dimensão "Detectável". Fixada essa dimensão, podemos analisar as características impressas pela combinação dela com as outras duas restantes, ou seja, as dimensões "Conhecido" e "Solucionável". Assim, identificamos três grupos de exceções que um sistema de GPN pode detectar: exceções conhecidas e solucionáveis, que têm as dimensões "Detectável", "Conhecido" e "Solucionável"; exceções desconhecidas e solucionáveis, que têm as dimensões "Detectável" e "Solucionável"; e exceções desconhecidas e não solucionáveis, que só têm a dimensão "Detectável".

Exceções **conhecidas e solucionáveis**, também chamadas de **exceções esperadas** [EL95] representam o tipo de exceção que a grande maioria dos sistemas de GPN prevê. O projetista descreve na definição do processo de negócio todos os mecanismos de tratamento de exceções esperadas antes da fase de criação e execução de instâncias. Caso alguma exceção desse tipo ocorra, ela é detectada e solucionada no momento de sua ocorrência pelo sistema de GPN.

Ao primeiro olhar, a detecção de **exceções desconhecidas** pode causar estranheza, entretanto, muitas vezes isso é possível por meio da detecção de uma exceção mais geral que seja conhecida e que "englobe" a exceção desconhecida. Dependendo da situação de exceção, com o auxílio dessa exceção mais geral, é possível até mesmo que ela seja solucionada no momento de sua ocorrência, como ocorre no grupo das **exceções desconhecidas e solucionáveis**. Em outros casos, a utilização de uma exceção mais geral apenas auxilia na detecção da exceção desconhecida mas não é suficiente para permitir que o sistema de GPN possa solucioná-la no momento de sua ocorrência. Isso caracteriza as **exceções desconhecidas e não solucionáveis**. Embora existam formas de se tratar esse tipo de exceção, o que envolve, normalmente, a intervenção de um especialista, são raros os sistemas de GPN que permitem isso [Ant11].

2.4 Cenário: Processo de aluguel de carros

Para facilitar a compreensão de alguns conceitos relacionados a abordagem WED-flow (ver Seção 2.5), que explicaremos adiante, utilizaremos, no decorrer deste trabalho, exemplos baseados no cenário de uma empresa de aluguel de carros com ênfase no acompanhamento do pedido de locação feito por um cliente.

(1) Um cliente solicita o início do processo de locação;

(2) Em seguida o cliente escolhe o carro de sua preferência e envia documentos pessoais (CNH, RG);

(3) O pedido do cliente é analisado por um gerente que decide se a reserva pode ser autorizada ou não;

(4) Caso o pedido de reserva seja negado, o processe de locação termina;

(5) Caso o pedido de reserva seja autorizado, o cliente retira o carro;

(6) Devolvido o carro, ele é então vistoriado;

(7) Caso alguma avaria seja verificada, a multa por ela é computada e acrescentada ao valor da locação;

(8) O cliente paga pela reserva e o processo é encerrado.

2.5 Abordagem WED-flow

A abordagem **WED-flow** (Work, Event and Data-flow) [FBTP12, FTMP10] foi proposta como uma forma de se modelar processos de negócios, de se instanciá-los, de se executar suas instâncias garantindo propriedades transacionais e de se tratar exceções de uma maneira mais simples. Nessa abordagem os passos de negócio, chamados de **WED-transitions**, e suas precondições de execução, chamadas **WED-conditions**, operam sempre sobre os estados de dados das instâncias. Tais estados, chamados **WED-states**, são valores para um conjunto de atributos de interesse da aplicação, chamados **WED-attributes**. Dessa forma, a execução de uma WED-transition em uma

dada instância de processo só é iniciada se o estado atual da instância satisfizer a WED-condition associada à WED-transition. Essa última, quando executada, alterará o estado da instância.

Um WED-trigger é um par (WED-condition, WED-transition) que associa uma precondição a um passo de negócio. Um processo de negócio é modelado por meio de um WED-flow, que é composto por um conjunto de WED-triggers e duas WED-conditions, as quais especificam a condição de início e término da execução de uma instância do processo em questão. Como uma dada aplicação pode ser constituída por diversos processos de negócio, podemos ter diversos WED-flows definidos sobre um mesmo conjunto de WED-attributes.

Alterações nos WED-states geram eventos que são capturados pelas WED-triggers que, por sua vez, verificam suas WED-conditions associadas e, caso elas sejam satisfeitas, disparam suas WED-transitions associadas, de forma similar ao modelo ECA (*Event*, *Condition*, *Action*). A execução de uma WED-transition, a qual é modelada como um passo SAGA [GMS87], gera um novo WED-state que, por conseguinte, gera um novo evento e assim por diante, até que um WED-state gerado satisfaça a condição de término do WED-flow. Caso alguma inconsistência seja detectada, os mecanismos de recuperação interrompem a execução da instância para tentar retorná-la a um estado consistente e dar prosseguimento à sua execução. A definição de inconsistência de instância será abordada na Seção 4.1.

Durante a modelagem, o projetista define de forma declarativa as WED-transitions, WED-conditions, WED-triggers e WED-flows, além das restrições de integridade da aplicação, chamadas de **AWIC** (Application-Wide Integrity Constraints), que são expressas na forma de WED-conditions e avaliadas sobre os WED-states. Todas as componentes acima são descritas pelos projetistas no modelo do WED-flow.

Nas próximas seções, apresentaremos mais formalmente as definições brevemente discutidas acima.

2.5.1 Exemplos

A seguir, na Figura 2.6, ilustramos uma possível modelagem do processo descrito na Seção 2.4 segundo a abordagem WED-flow. A notação utilizada foi exposta pela primeira vez no Simpósio Brasileiro de Banco de Dados 2012 durante a apresentação do artigo "WED-tool: uma ferramenta para o controle de execução de processos de negócio transacionais" [GSBF12]. Nessa notação, as elipses representam WED-attributes do WED-state da instância que serão alterados e os retângulos representam as WED-transitions a serem executadas. Quanto ao fluxo, as flechas contínuas indicam o disparo de uma WED-transition, as flechas tracejadas a alteração de um WED-attribute do WED-state da instância e as flechas pontilhadas representam a possibilidade de uma atividade alterar o WED-state de diferentes maneiras, mas de tal forma que somente uma delas pode ocorrer por execução. Finalmente, as flechas tracejadas e pontilhadas com um círculo contendo o termo AND em suas extremidades representam pontos de sincronismo do WED-state da instância.

O estado inicial do processo é aquele que tem definido o WED-attribute "**Reserva requisitada**". Ele é responsável por disparar a execução da WED-transition "**Inicializar reserva**", a qual define o WED-attribute "**Reserva iniciada**" no estado da instância. Esse estado é responsável pelo disparo da execução paralela de duas WED-transitions: a de escolha do carro e a de envio de documentos. Quando os WED-attributes "**Carro escolhido**" *E* "**Documentos recebidos**" estiverem definidos, a WED-transition "**Solicitar verificação do gerente**" é executada. Dependendo da situação, essa

WED-transition pode definir o WED-attribute "**Reserva rejeitada**" ou "**Reserva confirmada**". No primeiro caso, a execução da instância é concluída ("**Reserva rejeitada**") e no segundo a execução continua seguida do disparo da WED-transition "**Registrar retirada**", a qual altera o estado da instância definindo o WED-attribute "**Carro retirado**". Assim, a WED-transition "**Registrar devolução e vistoriar**" é disparada e ao fim de sua execução, dependendo das condições do carro devolvido, definirá os WED-attributes "**Carro vistoriado**" *OU* "**Carro com avarias**". No segundo caso, a WED-transition "**Computar multas**" é disparada para só então o WED-attribute "**Carro vistoriado**" ser definido. O estado de dados com essas características dispara a execução da WED-transition que registra o pagamento do cliente, resultando na definição do WED-attribute "**Reserva paga**", o que caracteriza, nesse caso, um estado final.

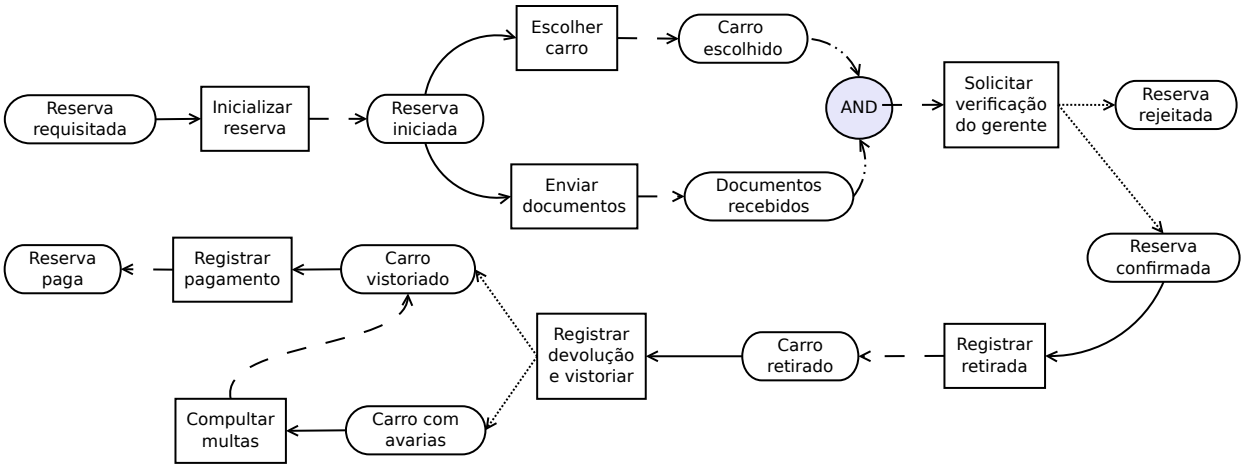


Figura 2.6: *Possível modelagem do processo descrito na Seção 2.4.*

Ao longo deste trabalho, apresentaremos exemplos baseados em trechos da execução de instâncias do modelo do WED-flow de aluguel de carros descrito acima. Optamos por representar trechos da execução de instâncias e não a execução completa para facilitar a compreensão do próprio exemplo e do conceito que faz uso dele. Exemplos completos de execução de instâncias podem ser encontrados no Apêndice A.2.

No exemplo da Figura 2.7, ilustramos parte da execução de uma instância do WED-flow de aluguel de carros. Entretanto, antes de apresentá-lo, consideramos válido explicar rapidamente a notação que foi utilizada. A execução de uma instância é representada por uma tabela que descreve a evolução dos WED-states dessa instância e dos WED-attributes associados a eles conforme são executadas WED-transitions. Cada linha dessa tabela representa um WED-state da instância e cada coluna contém os valores dos WED-attributes que fazem parte dos WED-states. A execução de WED-transitions é ilustrada por meio de flechas laterais. Elas indicam qual foi o WED-state que causou o disparo da execução de uma WED-transition e qual WED-state foi gerado por ela.

Voltando ao exemplo da Figura 2.7, representamos o trecho de execução da instância que trata a retirada de um carro por um cliente e sua devolução.

O WED-state 0 contém os valores "xyz", "confirmada", "null" e "null" para os WED-attributes "id. do usuário", "Status reserva", "Status carro" e "Placa do carro", respectivamente. Esse WED-state satisfaz a WED-condition de um determinado WED-trigger, o qual dispara sua WED-transition

associada, no caso, “WT: Registrar retirada”. Essa WED-transition gera o WED-state 1, que tem os valores “xyz”, “confirmada”, “retirado” e “ABC-1234” para os WED-attributes “id. do usuário”, “Status reserva”, “Status carro” e “Placa do carro”, respectivamente. Note que os valores “retirado” e “ABC-1234” estão em negrito e sublinhados para indicar que esses foram os WED-attributes alterados pela execução da WED-transition “WT: Registrar retirada”. O WED-state 1 satisfaz a WED-condition de outro WED-trigger, o qual dispara sua WED-transition associada “WT: Registrar devolução e vistoriar”, resultando na geração do WED-state 2. Esse WED-state, por sua vez, tem os valores “xyz”, “confirmada”, “vistoriado” e “ABC-1234” para os WED-attributes “id. do usuário”, “Status reserva”, “Status carro” e “Placa do carro”.

	id. do usuário	Status reserva	Status carro	Placa do carro
WED-state 0	xyz	confirmada	*null*	*null*
WED-state 1	xyz	confirmada	**retirado**	**ABC-1234**
WED-state 2	xyz	confirmada	**vistoriado**	ABC-1234

WT: Registrar retirada

WT: Registrar devolução e vistoriar

Figura 2.7: *Exemplo de parte da execução de uma instância de processo de negócio baseado no cenário de Aluguel de carros, disponível na Seção 2.4.*

As definições a seguir foram extraídas de Ferreira et al. (2012) [FBTP12] e Ferreira et al. (2010) [FTMP10].

2.5.2 WED-attributes

WED-attributes é um subconjunto dos atributos de um ou mais bancos de dados utilizados pela aplicação. Definimos a tupla $A = \langle a_1, a_2, ..., a_n \rangle$, onde cada a_i (com $1 \leq i \leq n$) é um atributo de interesse da aplicação.

2.5.3 WED-state

Um WED-state representa o estado de todos os WED-attributes de uma instância do WED-flow em um dado momento. Seja S o conjunto de todos os possíveis WED-states em uma aplicação. S é definido como $S = \{\langle v_1, v_2, ..., v_n \rangle \mid \forall i \in [1; n],\ v_i \in \mathit{domínio}(a_i)\}$ onde a_i é o i-ésimo atributo de A e $\mathit{domínio}(a_i)$ é o conjunto de valores que esse atributo pode assumir. Dizemos que um WED-state s é o WED-state atual de uma instância do WED-flow em execução quando s é o WED-state mais recente da instância.

2.5.4 WED-condition

Uma WED-condition é uma condição lógica sobre os WED-attributes da aplicação. Denotaremos por C o conjunto de todas as WED-conditions da aplicação.

2.5.5 WED-transition

Uma WED-transition t é uma função $t : S \rightarrow S$. Em outras palavras é uma função que a partir de um WED-state $s_1 \in S$ gera um WED-state $s_2 \in S$. Cada WED-transition tem uma compensação associada chamada de **WED-compensation**. As noções de compensação e WED-compensation serão mais aprofundadas no Capítulo 4. Denotaremos por U_t o conjunto de atributos atualizados por t e T o conjunto de todas as WED-transitions da aplicação.

2.5.6 WED-trigger

Um WED-trigger é um par $g = \langle c, t \rangle$ onde c é uma WED-condition e t é uma WED-transition. Denotaremos por G o conjunto de todos os WED-triggers da aplicação. Supondo que s_{atual} é o WED-state atual de uma instância i do WED-flow, se um WED-state s de i satisfaz c, então g vai disparar a transição $t(s_{atual})$. Dizemos que um WED-state s' é **oferecido** a um WED-trigger g quando a WED-condition de g é verificada sobre s'.

2.5.7 WED-flow

Um WED-flow é uma tripla $\langle G`, c_i, c_f \rangle$ onde $G` \subseteq G$ e $c_i, c_f \in C$ são as condições iniciais e finais respectivamente. Em outras palavras, um WED-flow é um conjunto de WED-triggers.

2.5.8 AWIC-consistent WED-state

Restrições de integridade da aplicação (**A**pplication-**W**ide **I**ntegrity **C**onstraints) são condições lógicas definidas pelo projetista com o objetivo de expressar regras de negócio. AWICs são definidas sobre os WED-attributes de uma aplicação. Chamaremos de W o conjunto de todas as AWICs da aplicação e, além disso, $W \subseteq C$. Um WED-state que respeite a todas as AWICs é chamado **AWIC-consistent WED-state**.

2.5.9 Transaction-consistent WED-state

Dizemos que um WED-state s é Transaction-consistent se ao menos uma das seguintes propriedades é assegurada:

- s satisfaz a WED-condition de ao menos um WED-trigger g resultando no disparo da WED-transition associada a g.
- existe ao menos uma outra WED-transition t sendo executada na instância do WED-flow em que s foi gerado, ou seja, mesmo que s não satisfaça nenhuma WED-condition, s não será o último WED-state da instância, pois t, ao final de sua execução, gerará um novo WED-state.

2.5.10 WED-state inconsistente

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

2.5.11 Histórico de execução

Um registro histórico guarda os dados da execução de uma WED-transition e pode ser representado por uma 4-tupla $\langle s_c, t, s_e, s_s \rangle$, onde:

- $s_c \in S$ indica o WED-state que satisfez a condição que disparou t;
- $t \in T$ é a WED-transition executada;
- $s_e \in S$ indica o WED-state utilizado como entrada para t;
- $s_s \in S$ indica o WED-state gerado como saída de t.

A história de uma instância do WED-flow ou, como utilizaremos frequentemente neste trabalho, o **histórico de execução** de uma instância do WED-flow é uma sequência $\langle e_1, e_2, ..., e_k \rangle$ onde cada e_i (com $1 \leq i \leq k$) é um registro histórico.

2.5.12 Execução paralela

Conforme explicado na introdução da Seção 2.5, quando um WED-state é gerado na instância de um WED-flow, os WED-triggers dessa instância testam suas WED-conditions sobre esse WED-state e, caso elas sejam satisfeitas, os WED-triggers disparam a execução de suas WED-transitions. Quando um WED-state satisfaz a WED-condition de mais de um WED-trigger e esses disparam suas WED-transitions associadas, temos uma **execução paralela** de WED-transitions dentro de uma mesma instância de WED-flow. A principal peculiaridade da concepção de execuções paralelas é que nem sempre o WED-state que habilita a execução de uma WED-transition é o mesmo WED-state que será usado como entrada para a sua execução. Isso ocorre porque entre o momento em que o mecanismo de controle de execução detecta que uma WED-transition pode ser executada e o momento da sua execução de fato, pode ser que o WED-state que disparou a WED-transition em questão não seja mais o WED-state atual da instância. Então, ao invés de efetuar alterações sobre o WED-state que a disparou, a WED-transition efetua alterações sobre o WED-state atual, garantindo que, apesar das transições paralelas, a instâcia possua um único estado atual, o qual acumula todas as mudanças de dados causadas na instância pela execução das transições ao longo do tempo.

A Figura 2.8 ilustra um exemplo de execução paralela dentro de uma instância de um WED-flow baseado no cenário de aluguel de carros (ver Seção 2.4). O trecho da execução ilustra o momento em que um cliente inicia o processo de reserva de um carro e envia seus documentos e as características desejadas do veículo. As WED-transitions "Enviar documentos" e "Escolher carro" foram projetadas para serem executadas paralelamente.

O WED-state 0 satisfaz as condições de um dado WED-trigger, que dispara a execução da WED-transition t_1 ("Inicializar reserva"). O WED-state 1 gerado pela WED-transition 1 passa a ser o WED-state atual da instância em execução e é oferecido a todos os WED-triggers do WED-flow em questão. O WED-state 1 satisfaz as WED-conditions de dois WED-triggers, os quais disparam a execução das WED-transitions t_2 ("Enviar documentos") e t_3 ("Escolher carro"), que são executadas paralelamente. O controle de execução se encarrega de fazer com que uma WED-transition em execução sempre faça alterações sobre o WED-state atual. Assim, a WED-transition 3 faz alterações sobre o WED-state 2, gerado pela WED-transition 2 (que foi executada mais rapidamente que a

WED-transition 3), e não sobre o WED-state 1, o qual disparou sua execução.

	id. do usuário	Status reserva	Modelo do carro	Categoria do carro	Período reserva	Doc. do usuário
WED-state 0	*null*	requisitada	*null*	*null*	*null*	*null*
WED-state 1	xyz	**iniciada**	*null*	*null*	*null*	*null*
WED-state 2	xyz	iniciada	*null*	*null*	*null*	**xyz.pdf**
WED-state 3	xyz	iniciada	**W3**	**B-1**	**1-1-2013 a 1-2-2013**	xyz.pdf

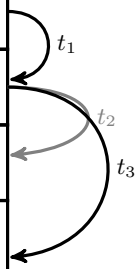


Figura 2.8: *Exemplo da execução paralela de duas WED-transitions em uma instância de um WED-flow modelado a partir do cenário de aluguel de carros, disponível na Seção 2.4. t_1, t_2 e t_3 representam as WED-transitions "Inicializar reserva", "Enviar documentos" e "Escolher carro", respectivamente.*

Esse exemplo pode ser representado de maneira mais sucinta, como mostra a Figura 2.9. Os WED-states são representados pelos nós do grafo, as WED-transitions pelas arestas sólidas e a ordem de geração dos WED-states, conforme o histórico de execução da instância, pelas arestas pontilhadas.

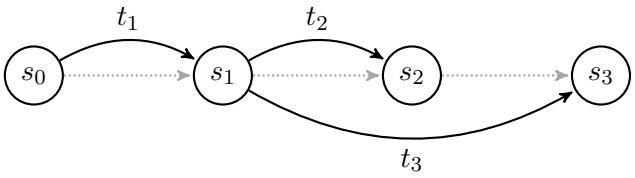


Figura 2.9: *Representação mais sucinta do exemplo da Figura 2.8.*

2.5.13 Mecanismos de recuperação

Ferreira et al. [FBTP12, FTMP10] definiram de forma sucinta e inicial dois mecanismos de recuperação: a recuperação *backward* e a recuperação *forward*. Quando é detectada alguma exceção durante a execução de uma instância do WED-flow, essa é interrompida e um desses mecanismos deve ser iniciado.

Recuperação backward

A recuperação backward consiste, na verdade, de dois métodos de recuperação. O primeiro, que só pode ser utilizado quando o WED-state atual da instância interrompida é consistente, realiza a execução encadeada de compensações de WED-transitions até que uma condição de parada seja atingida. O segundo, chamado de WED-S^{-1}, é executado sobre um WED-state inconsistente e pode ser entendido como um tipo especial de WED-transition que tem como função gerar um WED-state consistente. Esse WED-state gerado será, então, usado como ponto de partida para a execução encadeada de compensações.

Recuperação forward

O método de recuperação WED-S^{+a} é, em resumo, uma forma de se executar caminhos alternativos de execução durante o tratamento de exceções da abordagem WED-flow. Ele só pode ser usado quando o WED-state atual da instância for inconsistente.

2.6 Sumário

Neste capítulo estudamos os principais conceitos relacionados a processos de negócio e transações longas que fundamentam a abordagem WED-flow e, também, nossa proposta de aprimoramento do tratamento de exceções dessa abordagem, que apresentaremos em detalhes nos Capítulos 4 e 5. A organização deste capítulo de forma histórica auxilia na compreensão das evoluções promovidas pelos Modelos Transacionais Avançados, que viabilizaram a utilização de transações longas e, também, a aplicação desses conceitos em workflows e processos de negócio.

A discussão detalhada sobre exceções e linguagens de modelagem de processos de negócios nos ajudam a compreender melhor os problemas que apresentamos na Introdução deste texto (Capítulo 1) e também nos ajudam a justificar alguma das características da abordagem WED-flow e de nossa proposta.

Finalmente, além da definição das principais componentes da abordagem WED-flow, apresentamos o cenário de uso que servirá de plano de fundo para todos os exemplos utilizados no decorrer deste texto.

Capítulo 3

Trabalhos relacionados

O principal objetivo do tratamento de exceções em um sistema de gestão de processos de negócio é evitar ao máximo que uma instância em execução seja abortada e que, por consequência, tempo ou outros recursos sejam desperdiçados.

No decorrer dos últimos capítulos discutimos vários fundamentos relativos à gestão de processos de negócio e ao tratamento de exceções. Neste capítulo contextualizaremos os principais trabalhos relacionados e suas contribuições com o objetivo de ter uma visão geral do campo de estudos de processos de negócio, das principais abordagens de tratamento de exceções e de algumas das principais ferramentas e implementações de sistema de gestão de processos de negócio disponíveis.

3.1 Flexibilidade e Apoio

A **flexibilidade** de sistemas de gestão de processos de negócio é um assunto amplamente discutido em trabalhos acadêmicos [RD98, RDB03, LSKM00, MS02, SSO01, LS07, vdAPS09, WdLB03]. Apesar de cada autor definir esse conceito de maneira ligeiramente diferente, todas as definições acabam por convergir para a capacidade de um sistema de gestão de processos de negócio ou de um sistema de gerenciamento de workflows de alterar seus processos em algumas das etapas de seu ciclos de vida, a dizer nos momentos de modelagem e de execução de instâncias. Um termo bastante utilizado para sistemas de gerenciamento de workflows que têm essa característica é **Workflows Adaptáveis** [KBTB98]. Vários trabalhos apontam a falta de flexibilidade ou a insuficiência dela como principal causa do fracasso de alguns projetos de sistemas gerenciadores de workflow [RD98].

No contexto de tratamento de exceções em sistemas de gestão de processos de negócio e sistemas de gerenciamento de worflows, a flexibilidade tem um papel central uma vez que exceções desconhecidas, falhas de modelagem e falhas específicas de determinada instância são situações que não são previstas no modelo e que, por isso, exigem mudanças na instância ou no próprio modelo para serem tratadas. Podemos citar como exemplos de mecanismos que aumentam a flexibilidade de sistemas de gestão de processos de negócio, de sistemas de gerenciamento de workflows e, consequentemente, de suas formas de tratamento de exceções: (i) a possibilidade de intervenção *ad-hoc*, que permite a alteração da execução de uma determinada instância sem interferir nas possíveis demais; (ii) a modelagem evolutiva, que permite que o modelo de um processo de negócio seja modificado mesmo quando instâncias desse modelo estejam sendo executadas, e (iii) a reutilização de componentes ou partes do modelo de um processo de negócio em modelos de outros processos. Uma lista bem mais detalhada pode ser encontrada em outros trabalhos [KBTB98, vdAPS09, SSO01].

O **apoio**[1] oferecido por sistemas de gestão de processo de negócio e sistemas de gerenciamento de workflows é, em linhas gerais, o conjunto de mecanismos oferecidos por esses sistemas para viabilizar a realização com sucesso das etapas de seus ciclos de vida. Assim como o conceito de flexibilidade, o conceito de apoio é abordado em vários trabalhos [LS07, vdAPS09, EL95, RD98, CCPP99] e, apesar das diferentes nomenclaturas, converge para a definição descrita acima. Aalst et al. fazem uma sistematização interessante sobre a relação entre flexibilidade e apoio [vdAPS09] e, assim como outros autores [CCPP99, LS07, MS02], concluem que quanto mais flexibilidade um sistema de GPN provê, menos **apoio** ele oferece em contrapartida.

Estamos especialmente interessados no apoio em duas etapas do ciclo de vida de um processo de negócio ou workflow: na etapa de modelagem e na etapa de execução de uma instância. O **apoio à modelagem** é caracterizado sobretudo pela possibilidade de verificação de correção do modelo e pela análise de desempenho, e o **apoio à execução de uma instância** pela garantia de que a execução de instâncias respeite o que está definido no modelo, pela verificação da correção de alterações feitas no modelo durante a execução de instâncias e pelo monitoramente delas. Podemos observar que o apoio é uma qualidade tão essencial quanto a flexibilidade, entretanto, como visto anteriormente, quanto maior a flexibilidade, menor o apoio. Logo, assim como representado na Figura 3.1 concluímos que há uma busca pelo "ponto" ótimo entre flexibilidade e apoio, de acordo com as restrições da forma de modelagem utilizada e a finalidade dos processos de negócio a serem gestionados pelo sistema de GPN ou gerenciados pelo sistema de gerenciamento de workflows.

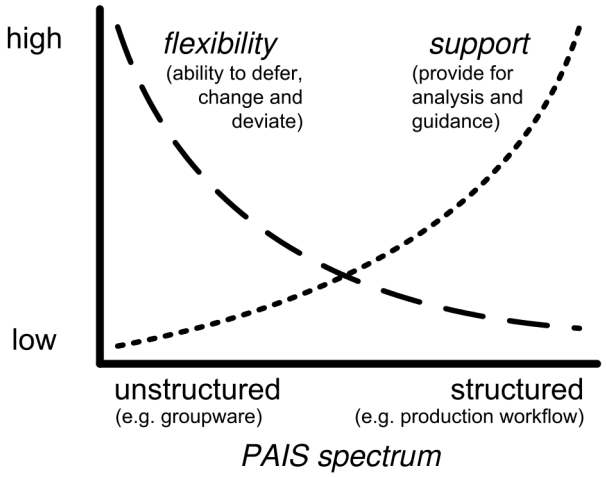


Figura 3.1: *Figura que retrata a relação entre flexibilidade e apoio em PAIS (Process Aware Information Systems) que são sistemas que, assim como sistemas de gestão de processos de negócio e sistemas de gerenciamento de workflow, baseiam-se em processos de negócio para realizarem algum trabalho. Esta figura foi retirada de [vdAPS09]).*

3.2 Influência das linguagens de modelagem de processos de negócio

Vários trabalhos [LS07, vdAPS09, RD98, MS02] apontam a linguagem de modelagem de processos de negócio como a principal causadora da relação paradoxal entre flexibilidade e apoio. Na Seção

[1] O termo originalmente utilizado por Aalst et al. é *support*. Na ausência de uma tradução para o português que reflita o sentido original dessa palavra, optamos pela utilização do termo apoio, que, em nossa concepção, é o que mais se aproxima.

2.3 discutimos os dois principais tipos de linguagem (linguagens baseadas em grafos e baseadas em regras) e apontamos algumas qualidades e defeitos de cada um deles. Contextualizando esses tipos de linguagem com os conceitos de flexibilidade e apoio, percebemos que linguagens baseadas em regras tendem a ser mais flexíveis e prover menos apoio enquanto que linguagens baseadas em grafos tendem a prover mais apoio e ser menos flexíveis [LS07, vdAPS09, RD98, WdLB03]. Entretanto vale ressaltar que a escolha da linguagem de modelagem também está relacionada ao perfil dos processos de negócios a serem gestionados já que existem processos que são melhor representados por linguagens baseadas em grafos e outros que são melhor representados por linguagens baseadas em regras [vdAPS09].

3.3 Abordagens transacionais

A utilização de técnicas de workflows transacionais e processos transacionais é uma forma usual de se embasar abordagens de processos de negócio e workflows quando há a preocupação de se garantir propriedades transacionais à execução de instâncias. Trabalhos como Schuldt et al. [SABS02], Bhiri et al. [BGG+11] e Vydiangaskar & Vossen [VV11] fazem uso da semi-atomicidade [ZNBB94], um modelo transacional avançado, que estende o modelo SAGA (Seção 2.2.4) e permite, além da compensação de atividades, a reexecução de atividades e a execução de um caminho alternativo. Por visarem principalmente o apoio à modelagem e à execução, esses trabalhos utilizam linguagens de modelagem baseadas em grafos. Os sistemas de gerenciamento de workflows dessas abordagens permitem, em especial, a validação das definições dos modelos dos processos de negócio no momento de modelagem e, especificamente no trabalho de Schuldt et al., a geração de escalonamentos de atividades, os quais permitem a execução concorrente de instâncias que partilham os mesmos recursos.

A principal desvantagem do modelo semi-atômico é a necessidade de se definir no momento de modelagem todos os possíveis caminhos de execução, todos os tipos de passo de negócio e todas as compensações dos modelos dos processos de negócio. Esse tipo de necessidade leva a uma modelagem passível de erros uma vez que no momento da modelagem, normalmente, o projetista não tem muita clareza sobre o processo de negócio. Além disso, para cada pivô, é necessário definir uma alternativa de execução que termine com sucesso, o que em sistemas complexos pode ser uma tarefa complicada.

Reichert & Dadam [RD98] apresentam um sistema de gerenciamento de workflows transacionais chamado "ADEPT" que utiliza uma linguagem baseada em grafos. Por meio da elaboração e fundamentação de diversos mecanismos, que permitem, por exemplo, alterações do modelo do processo e das instâncias durante a execução, os autores visaram aumentar a flexibilidade do sistema de gestão resultante da abordagem. Em um trabalho posterior [RDB03], Reichert et al. incrementaram o sistema de gerenciamento de workflows apresentado anteriormente e desenvolveram mecanismos de pulos *forward* e *backward*, que permitem a alteração da execução de uma determinada instância sem que outras sejam afetadas. De fato, o sistema torna-se mais flexível mantendo o apoio característico das linguagens de modelagem baseadas em grafos, no entanto, a curva de complexidade do modelo acaba por crescer mais rapidamente do que a do aumento da flexibilidade, o que faz com que o modelo de processos um pouco mais elaborados seja de difícil compreensão.

Eder & Liebhart apresentaram "WAMO" [EL95], um Workflow Transacional que utiliza uma linguagem de modelagem baseada em regras para modelar processos de negócios orientados a fluxo,

algo que não é muito usual (Seção 2.3). Com isso os autores visavam facilitar o processo de modelagem e o tratamento de exceções esperadas de modo que o projetista pudesse explicitar detalhes sobre a forma de compensação de uma atividade. Todavia, apesar da maior facilidade de aproveitamento de componentes do modelo, o projetista do sistema não tem acesso a uma das principais qualidades de uma linguagem de modelagem baseada em grafos: a forma visual de representação. Como o sistema não oferece mecanismos como intervenção *ad hoc*, alteração do modelo durante a execução, etc., as principais qualidades de linguagens baseadas em regras também não são aproveitadas. Entretanto, a principal desvantagem delas, que é a maior dificuldade de se compreender as relações entre atividades no modelo do processo, está presente.

Casati et al. [CCPP99] desenvolveram em seu trabalho a "Chimera Exc" uma linguagem de modelagem baseada em regras, bastante semelhante ao ECA, específica para o tratamento de exceções. Essa linguagem é implementada pelo sistema de gerenciamento de workflows *FAR* e tem componentes que visam garantir propriedades transacionais à execução de instâncias. O trabalho, no entanto, concentra-se em apoiar a modelagem de workflows e deixa de lado, além do apoio à execução, a flexibilidade. Esse não é o caso do trabalho de Wainer & Silva [WdLB03], que concentra esforços para garantir flexibilidade ao arcabouço "Tucupi Server", que implementa uma linguagem de modelagem baseada em regras, mais especificamente, baseada em restrições. Entretanto, o tratamento de exceções da abordagem é feito de forma bastante preliminar e as restrições definidas por meio de uma linguagem lógica acaba por dificultar a compreensão do modelo, especialmente para usuários não familiarizados com esse tipo de representação.

3.4 Abordagens não transacionais

Dependendo da finalidade dos processos de negócio, a garantia de propriedades transacionais pode não ser uma prioridade para o sistema de gestão de processos de negócio ou para o sistema de gerenciamento de workflows. Mesmo que fexibilizadas, propriedades transacionais impõem certas restrições que podem ser um empecilho especialmente em processos de negócio cujo controle do fluxo depende totalmente ou em grande parte da intervenção de usuários. Como exemplos desse tipo de abordagem, podemos citar os trabalhos de Mourão & Antunes e Antunes et al. [Ant11, MaA07].

Em outros casos, a necessidade de se garantir flexibilidade e de se facilitar o processo de modelagem que é o caso do arcabouço *Declare* desenvolvido por Aalst et at. [vdAPS09] que utiliza uma linguagem de modelagem baseada em restrições. No caso desse trabalho, os autores não se preocuparam em definir formas de tratamento de exceções, mas sim de estruturar a linguagem de modelagem de tal forma que mecanismos de apoio à modelagem, como pré-verificação de correção do modelo, e de apoio à execução, como a garantia de correção da execução de instâncias e o monitoramento delas, fossem possíveis de ser desenvolvidos.

3.5 Ferramentas comerciais e implementações

Nesta seção, nosso objetivo é apresentar algumas das principais iniciativas de implementação, comerciais e acadêmicas, de conceitos e abordagens de processos de negócio e de workflows discutidas neste capítulo e nos capítulos anteriores. É importante ressaltar que está fora do nosso escopo fazer uma análise aprofundada ou comparar os trabalhos e ferramentas enunciados a seguir.

Dentre os trabalhos que discutimos neste capítulo e que contribuem com modelos, abordagens ou formas de tratamento de exceções, o único que tem, atualmente, uma implementação disponível publicamente é o *Declare* de Aalst et al. Os demais ou tiveram seus projetos descontinuados, ou estagnaram em protótipos que não se tornaram um produto final, ou ainda não foram implementados. No entanto, fora do âmbito da academia, existem vários sistemas de gestão de processos de negócio e de gerenciamento de workflows, tanto gratuitos quanto pagos, que se utilizam de conceitos concebidos em trabalhos acadêmicos em suas implementações.

Garcês et al. [GJCV09] fizeram um levantamento das principais ferramentas *open source* de gestão de processos de negócio e de gerenciamento de workflows disponíveis na atualidade. É interessante observar que, de acordo com esse trabalho, dos dez sistemas analisados, dois não possuem tratamento de exceções e nenhum dos listados faz uso, nativamente, de linguagens de modelagem baseadas em regras.

Entre os trabalhos descritos por Garcês et al. estão os sistemas de gestão de processos de negócio *YAWL System* [vdAADtH04, Fon] e *Bonita* [Bon] que são conhecidos e que dispõem de uma grande quantidade de documentação e de usuários.

O *Bonita* é um sistema de gestão de processos de negócio que utiliza uma linguagem baseada em grafos para descrever suas definições. Apesar de ser conhecida, essa ferramenta peca no quesito de tratamento de exceções. Apenas exceções esperadas são detectadas e o tratamento delas resume-se à execução de atividades previamente definidas pelo projetista, mas que não são necessariamente compensações uma vez que compensações não são explicitamente modeladas nesse sistema. Mais informações sobre esse sistema de gestão de processos de negócio podem ser encontradas no sítio e na documentação da ferramenta [Bon].

O *YAWL System* é um sistema de gestão de processos de negócio modular que utiliza principalmente a linguagem de modelagem *YAWL* (*Yet Another Workflow Language*), que é baseada em grafos, mas que prevê a utilização de regras e restrições para a descrição de determinados procedimentos. Essa mistura de tipos de linguagem de modelagem permite que sejam desenvolvidos mecanismos de apoio, como a verificação prévia do modelo, e, também, mecanismos que garantam flexibilidade, como, por exemplo, a definição explícita de exceções, por meio de regras, prevista pelo módulo padrão de tratamento de exceções. Ainda no contexto de tratamento de exceções, o *YAWL System* é capaz de identificar exceções esperadas e não esperadas assim como ele permite o acréscimo ou remoção de regras ao módulo de tratamento de exceções. A característica modular desse sistema de gestão de processos de negócio ainda permite que um outro módulo de tratamento de exceções, definido pelo usuário, seja utilizado.

O *YAWL System* também prevê a utilização de um componente que implementa o arcabouço *Declare*, o qual permite a utilização de uma linguagem baseada em restrições, conforme discutimos anteriormente neste capítulo, para a definição do modelo dos processos de negócio substituindo a linguagem de modelagem YAWL. Esse componente permite que o Declare seja integrado com as demais funcionalidades do *YAWL System*, como o tratamento de exceções, gerenciamento de dados, de recursos, etc., garantindo assim mais flexibilidade ao sistema. Mais informações sobre esse sistema de gestão de processos de negócio podem ser encontradas em outros trabalhos [vdAADtH04, vdAPS09] e na documentação da ferramenta [Fon].

Obter uma visão geral de ferramentas pagas não é uma tarefa trivial. As dificuldades vão desde a ausência de versões de demonstração das ferramentas, passando por documentação vaga ou ine-

xistente para não clientes, até pré-requisitos de instalação difíceis de se alcançar. As soluções que encontramos para superar essas dificuldades foram:

- Procurar artigos acadêmicos que se proponham a avaliar ferramentas de workflow e de processos de negócio comerciais;
- Utilizar a pouca informação disponível nos sítios e documentação das ferramentas;
- Procurar mais informações nos fóruns e espaços de discussão da *Internet*.

Eswaran et al. [EVWH06] fazem uma análise detalhada das ferramentas Microsoft BizTalk Server [Mica], Microsoft Workflow Foundation [Micb] e Oracle BPEL Process Manager [Ora]. Apesar de o principal interesse desse artigo ser a avaliação dessas ferramentas no contexto de computação em grade, nos momentos em que os autores abordam as formas de tratamento de exceções disponibilizadas por elas, é possível observar a tendência de limitação do tratamento de exceções à execução de compensações e re-execução de atividades. Encontramos essas mesmas características nas ferramentas Appian BPM [App], Savvion BPM [Aur] e IBM BPM [IBM]. A intervenção *ad hoc* de um administrador de sistema no momento do tratamento de exceções é atendida somente na Oracle BPEL Process Manager. A possibilidade de utilização de linguagens de modelagem baseadas em regras, ao menos durante o tratamento de exceções, aparece somente na ferramenta da Oracle e na Appian BPM.

Nosso objetivo nesta seção foi apresentar uma visão geral das ferramentas atuais que implementam alguns dos vários conceitos de tratamento de exceções citados ao longo deste trabalho. De fato, um estudo mais aprofundado sobre essas ferramentas que envolva, por exemplo, a avaliação e a comparação delas poderia ser feito, porém, não é o escopo deste trabalho.

3.6 Sumário

Neste capítulo discutimos e contextualizamos os principais conceitos relacionados ao tratamento de exceções em sistemas de gestão de processos de negócio e sistemas de gerenciamento de workflow. Em especial, discutimos a flexibilidade e o apoio, e a relação desses dois conceitos com o tipo de linguagem de modelagem escolhida para representar os processos de negócio. Estudamos também as principais abordagens de controle de execução apontando as principais deficiências e as principais contribuições de cada uma delas. Por fim, este capítulo também nos ajudou a dar uma visão geral de algumas das principais ferramentas comerciais (gratuitas, pagas ou open-source) assim como de protótipos acadêmicos.

Todos esses conceitos auxiliarão na compreensão e contextualização da nossa proposta de tratamento de exceções que apresentaremos nos capítulos a seguir.

Capítulo 4

Aprimoramento de um modelo de tratamento de exceções

O tratamento de exceções é uma frente importante da abordagem WED-flow e tem sido alvo de estudos desde suas primeiras definições. De fato, alguns dos métodos de tratamento de exceções da abordagem WED-flow já foram apresentados em outros trabalhos [FBTP12, FTMP10], entretanto, o foco principal deles estava relacionado principalmente ao controle de execução e à sustentação teórica da abordagem. Consequentemente, conceitos e métodos de tratamento de exceções foram definidos de forma sucinta e inicial.

Em nossa proposta para o tratamento de exceções na abordagem WED-flow, (i) explicamos em detalhes e embasamos teoricamente os mecanismos de interrupção e de detecção de exceções, previamente apresentados [FBTP12]; (ii) formalizamos e fundamentamos os métodos de tratamento de exceções apresentados anteriormente [FBTP12, FTMP10], mais explicitamente as formas de compensação, a alteração da definição de um WED-flow durante uma interrupção, os pulos *backward* (WED-S^{-1}) e os pulos *forward* (WED-S^{+a}) e, finalmente, (iii) criamos um novo método chamado **oferecimento adaptativo**.

4.1 Exceções na abordagem WED-flow

Exceções, as quais são capturadas durante a execução de instâncias de um WED-flow, são englobadas pelos seguintes **casos gerais**:

- **Falha durante a execução de uma WED-transition:** Uma WED-transition pode falhar devido a problemas de implementação, *time-out* da execução, conflitos de escrita causados por WED-transitions sendo executadas paralelamente, cancelamento explícito da execução por um administrador do sistema ou devido a problemas estruturais que resultem na interrupção inesperada da execução, como no caso de um defeito em algum componente do servidor.
- **Produção de um WED-state inconsistente:** Um WED-state é dito inconsistente quando ele não é AWIC e nem *Transaction consistent* (ver Seção 2.5.10). WED-states inconsistentes são, normalmente, consequência de falhas de projeto, manipulação direta do banco de dados ou de problemas estruturais, como, por exemplo, o corrompimento do disco rígido.

Dizemos que uma **instância** de um WED-flow está **inconsistente** quando uma exceção é detectada durante sua execução [Mar12]. Isso resulta na ativação do **gerenciador de recuperação**, o qual **interrompe** imediatamente a instância em questão para permitir que, por meio de mecanismos de tratamento de exceções, ela volte a ser consistente novamente, e a execução da instância possa continuar.

4.1.1 Detecção de exceções

Na abordagem proposta neste trabalho, uma exceção é detectada face a uma falha de uma execução de uma WED-transition ou pela produção de um WED-state inconsistente. Essa forma geral de se capturar exceções não traz informações específicas sobre a causa da exceção, mas sim sobre o caso geral em que ela se enquadra.

Na Seção 2.3.5, apresentamos uma maneira de se analisar exceções sob a ótica do espaço de conhecimento de uma exceção. Resumidamente, esse espaço tem três dimensões, "Conhecido", "Detectável" e "Solucionável", que, observadas em conjunto, ajudam a compreender melhor algumas características de exceções, que são representadas por pontos nesse espaço. A forma de se detectar exceções da abordagem WED-flow é capaz de identificar exceções desconhecidas, ou seja, que não têm a dimensão "Conhecido", e exceções não solucionáveis, em outras palavras, que não têm a dimensão "Solucionável".

Por exemplo, ao detectar uma exceção consequente da geração de um WED-state inconsistente, não é possível saber exatamente qual foi a falha ou exceção específica que levou à geração da exceção em questão. Pode acontecer, inclusive, que no momento da ocorrência não se saiba ainda como tratá-la. Estamos, então, sempre associando uma falha específica a uma falha mais geral. Assim, podemos afirmar que essa forma de detecção acaba por identificar uma quantidade maior de exceções do que outras abordagens de GPN uma vez que a maioria delas é capaz de detectar apenas exceções esperadas, ou seja, que têm as dimensões "Conhecido" e "Detectável".

Na abordagem WED-flow, exceções esperadas são tratadas de maneira implícita. Devido a grande semelhança entre exceções esperadas e atividades "normais" do processo de negócio, especialmente em se tratando de linguagens de modelagem baseada em regras, optou-se por passar a responsabilidade de se tratar esse tipo de situação excepcional ao projetista. Ele deve definir no modelo do processo de negócio WED-triggers que tenham WED-conditions que identifiquem a exceção esperada e que tenham WED-transitions que, quando disparadas, resolvam a situação excepcional. Dessa forma, os mecanismos de tratamento de exceções descritos na abordagem WED-flow só são acionados de fato quando exceções não esperadas são detectadas, o que, como veremos a frente, concentra o trabalho do administrador de sistema em exceções que não podem ser previstas no momento de projeto.

4.2 Interrupção

A interrupção da execução de uma instância inconsistente é o ponto de partida para a inicialização do gerenciador de recuperação. Como explicado anteriormente, uma instância só é interrompida se ela estiver inconsistente e isso só acontece quando uma exceção não esperada é detectada durante sua execução. A interrupção de uma instância implica em:

- paralização dos WED-triggers da instância;
- suspensão de novas execuções de WED-transitions na instância;
- aborto de WED-transitions que estavam sendo executadas na instância, no momento da interrupção.

Vale ressaltar que outras instâncias sendo executadas concomitantemente não são afetadas pela interrupção de uma dada instância.

A interrupção de uma instância do WED-flow é alvo de estudo de outros trabalhos [Mar12, Gar13] desenvolvidos no grupo DATA (*Database Modeling, Transactions and Analysis*) do IME-USP.

4.2.1 Tipos de interrupção

Para facilitar o processo de recuperação, algumas informações relativas à interrupção são salvas. Como a abordagem WED-flow provê o **histórico de execução** [FBTP12] de todas as instâncias, só é necessário manter a causa da interrupção e a data e horário em que ela ocorreu. Tais informações e, em especial, o último WED-state gerado na instância antes da interrupção, que é obtido a partir do histórico de execução da instância, influenciam diretamente na escolha dos métodos de recuperação que poderão ser utilizados no momento do tratamento de exceções. Esses métodos serão explicados na Seção 4.3.

Definição 4.1. Chamamos de **WED-state atual da interrupção** o último WED-state gerado na instância que foi interrompida.

O WED-state atual de uma interrupção pode ser **consistente** ou **inconsistente** e isso depende do tipo de exceção que causou a interrupção.

A conjunção da causa da exceção com o WED-state atual da interrupção resulta no **tipo da interrupção**, o qual será responsável pela seleção dos métodos de recuperação à disposição do administrador do sistema.

Definição 4.2. Interrupções causadas pela produção de WED-state inconsistente sempre têm o WED-state atual de interrupção inconsistente.

Conforme a definição de inconsistência de WED-states disponível na Seção 2.5.10, um WED-state inconsistente é necessariamente o último WED-state de uma instância. Logo, o WED-state atual da interrupção de uma instância inconsistente é necessariamente inconsistente.

Definição 4.3. Interrupções causadas por falha na execução de WED-transitions têm sempre o WED-state atual consistente.

Também pela definição de inconsistência, sempre que um novo WED-state é criado em uma instância que possui WED-transitions em execução, esse WED-state é considerado consistente. Portanto, o WED-state atual de uma interrupção causada por uma falha na execução de uma WED-transition é sempre um estado consistente.

Uma vez escolhido(s) e executado(s) o(s) método(s) de recuperação, a instância volta a ser consistente e, com isso, a interrupção pode ser finalizada e a execução dessa instância pode então prosseguir. É importante salientar que a única e imprescindível condição para a finalização de uma interrupção é que a instância interrompida volte a ser consistente. A única forma prevista na abordagem WED-flow para tal é por meio da execução de um ou mais métodos de recuperação.

4.3 Tratamento de exceções

Na abordagem WED-flow, quando uma exceção é detectada, o gerenciador de recuperação é ativado, a instância inconsistente em execução é interrompida e, só então, o tratamento de exceções pode ser iniciado. Seguindo o padrão de modelagem evolutiva defendido pela abordagem, optamos por uma forma de tratamento de exceções baseada na intervenção *ad hoc* de um administrador do sistema. Esse administrador será responsável, principalmente, por escolher um ou mais métodos de tratamento de exceções a serem utilizados pelo gerenciador de recuperação.

Neste trabalho, temos dois tipos básicos de recuperação: **Recuperação *Backward*** [1] e **Recuperação *Forward*** [1]:

- **Recuperação Backward:** Na literatura, uma recuperação *backward* normalmente se refere a métodos de recuperação que "voltam" no caminho de execução, ou seja, retornam o estado do banco de dados, ou, de forma geral, da aplicação, a um estado que já existiu ou equivalente a ele. Os métodos desse tipo que propomos são o **oferecimento adaptativo**, as **compensações** (encadeadas ou não) e o **WED-S^{-1}**.
- **Recuperação Forward:** Uma recuperação *forward* normalmente relaciona-se com métodos de recuperação que objetivam tratar uma exceção sem "voltar" no caminho de execução, seja por meio de retentativas, caminhos alternativos de execução ou por pulos [RDB03]. Neste trabalho, nós propomos os métodos de **oferecimento adaptativo** e **WED-S^{+a}**, que são classificados como recuperação *forward*.

Note que o método de oferecimento adaptativo ora se comporta como método de recuperação *backward*, ora como método de recuperação *forward*, dependendo do contexto de sua execução. Além disso, a **alteração da definição de um WED-flow**, que não é formalmente um método de recuperação tem um papel importante no processo de tratamento de exceções assim como os métodos recém citados. A seguir, faremos algumas definições necessárias à compreensão desses métodos e, logo em seguida, os enunciaremos em detalhes.

[1] Devido à consagração do uso dos termos recuperação *Backward* e recuperação *Forward*, optamos por não traduzí-los.

4.3.1 Equivalência de WED-states

Definição 4.4. Dois WED-states são ditos equivalentes quando eles habilitam a execução das mesmas WED-transitions.

Sejam s_1, s_2 WED-states, G_1 o conjunto de WED-triggers cujas WED-conditions são satisfeitas por s_1 e G_2 o conjunto de WED-triggers cujas WED-conditions são satisfeitas por s_2. Dizemos que s_1 é **equivalente** a s_2 se e somente se $G_1 = G_2$.

4.3.2 Histórico de execução

Como discutiremos em mais detalhes nas próximas seções deste capítulo, existem mecanismos de recuperação que precisam verificar a existência ou as características de execuções anteriores de WED-transitions, WED-compensations, WED-S^{-1}s e WED-S^{+a}s em uma determinada instância de WED-flow. Na Seção 2.5.11, apresentamos o histórico de execução de uma instância, que registra detalhes da execução de WED-transitions de determinada instância. É importante salientar que o histórico de execução também serve para guardar informações sobre WED-compensations, WED-S^{-1}s e WED-S^{+a}s.

Definição 4.5. WED-compensations, WED-S^{-1}s e WED-S^{+a}s são tipos especiais de WED-transitions.

WED-compensations, WED-S^{-1}s e WED-S^{+a}s geram um novo WED-state a partir de um WED-state inicial assim como WED-transitions. Elas apenas diferenciam-se entre si quanto ao tipo de WED-state inicial, que, em alguns casos, só pode ser consistente e em outros, inconsistente, e quanto às características do WED-state gerado, como por exemplo, as características especiais de WED-states gerados por pulos e compensações que respeitam algumas restrições específicas, como discutiremos nas próximas seções.

4.3.3 WED-compensation

No modelo SAGA (Seção 2.2.4), cada passo SAGA p_s tem uma compensação p_c associada que tem por objetivo desfazer, **de um ponto de vista semântico**, ações efetuadas por p_s sem, entretanto, retornar, obrigatoriamente, o banco de dados ao estado anterior à execução de p_s. Guardadas as diferenças quanto ao tipo de atividade, que, no caso do modelo SAGA são transações de bancos de dados e no caso da abordagem WED-flow são WED-transitions, podemos dizer que WED-compensations têm a mesma função que compensações de passos SAGA.

Uma WED-compensation é um mecanismo de tratamento de exceções responsável por desfazer, **semanticamente**, uma WED-transition.

Definição 4.6. Uma **WED-compensation** wc, assim como uma WED-transition, pode ser entendida como uma função $wc : S \to S$, tal que S é o conjunto de todos os possíveis WED-states da aplicação. Seja t uma WED-transition e wc uma WED-compensation. Se wc é a WED-compensation associada a t, então a execução de wc sobre um WED-state gerado pela execução de t deve resultar em um WED-state equivalente ao usado como entrada para a execução de t. Ou seja, se $t(s_i) = s_f$, então $wc(s_f) = s'_i$ sendo que s'_i é equivalente a s_i e s_i, s_f e s'_i são WED-states.

De forma análoga a WED-transitions, WED-compensations também registram suas execuções no histórico de execução da instância e, com isso, atualizam o WED-state atual da interrupção (ver Definição 4.1).

Observe que, diferentemente do modelo SAGA, que não impõe nenhum tipo de restrição ao termo "desfazer" e, assim, ao conceito de compensação, impomos que uma WED-compensation precisa gerar WED-states equivalentes aos que serviram de ponto de partida para a WED-transition associada. Isso garante mais controle ao tratamento de exceções.

A Figura 4.1 ilustra a execução de uma compensação. Nesse exemplo, s_3 é consistente e c_3 indica a execução da WED-compensation associada a WED-transition t_3. Além disso, utilizamos o sinal $\equiv$ para indicar a equivalência entre s_2 e s_4.

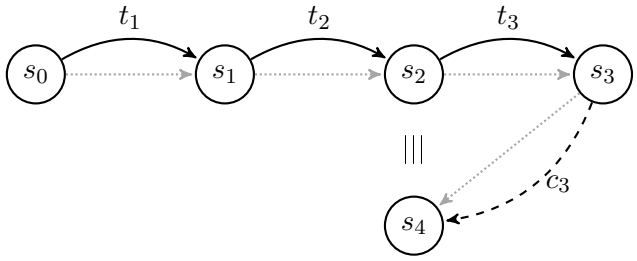


Figura 4.1: *Exemplo de execução de uma WED-compensation*

A Figura 4.2 ilustra o exemplo da execução de parte da instância de um processo de negócio baseado no cenário de aluguel de carros (Seção 2.4). Estamos supondo que um funcionário registrou a devolução de um carro errôneamente. Assim, a WED-transition "WT: Registrar devolução e vistoriar" é compensada por meio da execução da WED-compensation "WC: Registrar devolução e vistoriar" que gera o WED-state 3 que é equivalente ao WED-state 1.

	id. do usuário	Status reserva	Status carro	Placa do carro
WED-state 0	xyz	confirmada	*null*	*null*
WED-state 1	xyz	confirmada	**retirado**	**ABC-1234**
WED-state 2	xyz	confirmada	**vistoriado**	ABC-1234
WED-state 3	xyz	confirmada	**retirado**	ABC-1234

WT: Registrar retirada

WT: Registrar devolução e vistoriar

WC: Registrar devolução e vistoriar

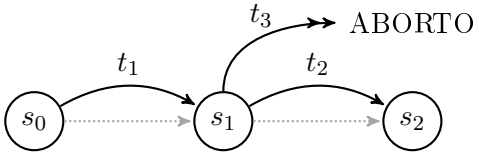
Figura 4.2: *Exemplo de parte da execução de uma instância de processo de negócio baseado no cenário de aluguel de carros (Seção 2.4). Cada WT:nome é uma WED-transition e cada WC:nome é uma WED-compensation.*

Uma primeira interpretação do conceito de WED-compensation foi apresentada em [FBTP12] sob o nome de WED-transition^{-1}. Nossa principal contribuição para o aprimoramento desse conceito foi o de estabelecer que uma WED-compensation precisa necessariamente gerar um WED-state equivalente ao WED-state sobre o qual a WED-transition sendo compensada foi executada. Desse modo, há maior controle sobre os efeitos de uma WED-compensation e é garantido que o WED-state gerado por ela será consistente.

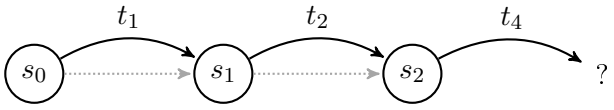
4.3.4 Oferecimento adaptativo

O **oferecimento adaptativo** é um método de recuperação que só pode ser executado quando o WED-state atual da interrupção é consistente. Como explicado na introdução da Seção 4.1, a exceção causadora desse tipo de interrupção é resultante de falha na execução de uma WED-transition. Em alguns casos, a execução de uma WED-transition que falhou pode terminar corretamente caso seja executada novamente. Entretanto, é necessário observar que a interrupção da instância inconsistente pode ter causado o aborto de WED-transitions que estavam sendo executadas paralelamente à WED-transition problemática. Por essa razão, a recuperação deve, além de garantir a chance de redisparo da WED-transition que sofreu falha, garantir também a chance de redisparo de todas as possíveis WED-transitions que foram abortadas em virtude da interrupção da instância.

Conforme explicado na introdução da Seção 2.5.12, WED-transitions executadas paralelamente em uma instância podem ter sido disparadas por diferentes WED-states gerados ao longo da execução dessa instância. Simplesmente reoferecer o WED-state atual da interrupção a todos os WED-triggers de um WED-flow poderia resultar na reexecução de somente um subconjunto das WED-transitions que foram interrompidas, como representado na Figura 4.3.



a- Uma exceção é detectada causando a interrupção e o consequente aborto de t_3.



b- Se o WED-state atual (s_2) fosse oferecido aos WED-triggers do WED-flow sem a utilização do oferecimento adapatativo e se s_2 não fosse equivalente a s_1, então t_3 não seria reexecutada, o que poderia resultar em falhas.

Figura 4.3: *Exemplo de situação que pode ocorrer caso haja o oferecimento de um WED-state sem a utilização do oferecimento adaptativo.*

Para sanar esse problema, o gerenciador de recuperação executa um pequeno algoritmo que, a partir de um WED-state consistente passado como parâmetro, faz as *adaptações* necessárias na instância interrompida para que haja a possibilidade da WED-transition que falhou ser disparada novamente.

Definição 4.7. O **oferecimento adaptativo** é um mecanismo de recuperação que tem como objetivo reiniciar uma instância interrompida por meio do oferecimento de um WED-state consistente aos WED-triggers dessa instância. Antes desse oferecimento, o mecanismo é responsável pela correção de eventuais inconsistências, causadas pelo aborto de execuções paralelas no momento da interrupção, por meio da seleção e execução automática de WED-compensations.

A seguir especificamos o algoritmo de oferecimento adaptativo. Ele recebe o WED-state atual da interrupção como parâmetro e faz o seguinte:

(1) O gerenciador de recuperação identifica a WED-transition t que gerou o WED-state s consistente, passado como parâmetro;

(2) Por meio do histórico de execução da instância, é verificado se s é o WED-state inicial da instância interrompida;

   (a) Caso s seja o WED-state inicial, a interrupção é finalizada e o WED-state atual da interrupção é oferecido a todos os WED-triggers do WED-flow.

(3) Caso t seja um WED-S^{-1} ou WED-compensation, o histórico de execução é verificado para encontrar s''' equivalente a s, e t passa ser a WED-transition que gerou s''';

(4) Caso t, em algum momento de sua execução, teve outras WED-transitions sendo executadas paralelamente que tenham sido interrompidas (abortadas), compensadas ou puladas (por um pulo backward WED-S^{-1}):

   (a) O gerenciador identifica o conjunto R que contém todas as WED-transitions que foram, em algum momento, executadas paralelamente a t e que tenham sido interrompidas (abortadas), compensadas ou puladas;

   (b) A partir de R e do histórico de execução, é possível identificar o WED-state s' que é o WED-state mais antigo a ter disparado alguma(s) das transições de R;

   (c) É efetuada uma **compensação encadeada** (ver Seção 4.3.5) tendo como condição de parada a geração de um WED-state s'' equivalente a s'. Note que WED-compensations atualizam o histórico de recuperação e, consequentemente, o WED-state atual da interrupção;

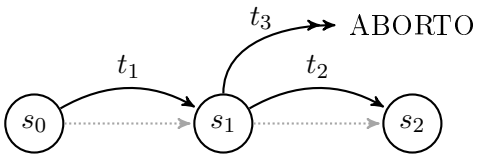
   (d) O passo 1 é então reexecutado utilizando s' como parâmetro.

(5) Caso contrário a interrupção é finalizada e o WED-state atual da interrupção é oferecido a todos os WED-triggers do WED-flow.

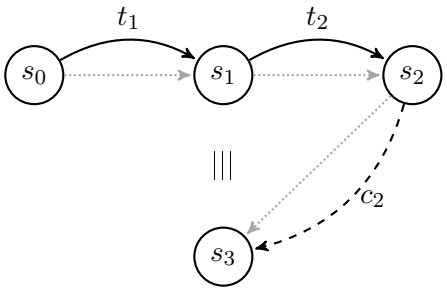
Caso haja WED-transitions sem WED-compensations definidas, o administrador deve adicioná-las às definições do WED-flow antes de iniciar o processo de recuperação ou somente será possível o uso do método de recuperação *forward* (ver Seção 4.3.7).

As Figuras 4.4 e 4.5 ilustram exemplos de oferecimento adaptativo. A Figura 4.5, em especial, ilustra um exemplo de oferecimento adaptativo que é baseado no cenário de aluguel de carros (Seção 2.4). Estamos supondo que houve um problema durante a execução da WED-transition t_3 ("Escolher carro"), que estava sendo executada em paralelo com a WED-transition t_2 ("Enviar documentos"). A

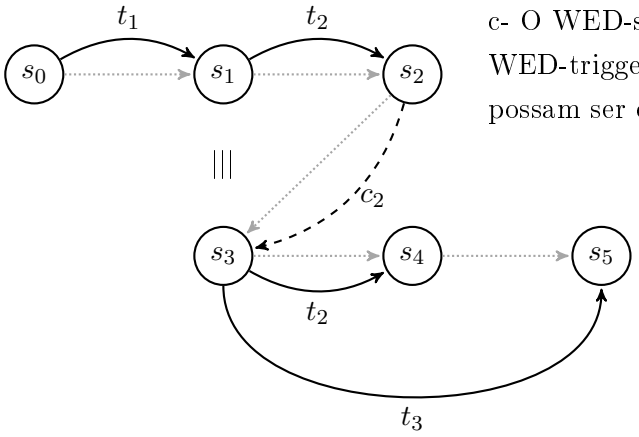
exceção é detectada e opta-se por fazer o oferecimento adaptativo do WED-state atual (WED-state 2). Com isso, A WED-transition t_2 que havia sido executada com sucesso é compensada pela WED-compensation c_2 ("Compensação escolher carro") e o WED-state 3 que é equivalente ao WED-state 1 é gerado. Assim, as WED-transitions t_2 e t_3 são disparadas (de forma paralela) novamente e os WED-states 4 e 5 são gerados gerados.



a- Uma exceção é detectada causando a interrupção e o consequente aborto de t_3.



b- O oferecimento adaptativo detecta o paralelismo pelo histórico de execução da instância e, assim, compensa t_2.



c- O WED-state s_3 é oferecido novamente aos WED-triggers possibilitando assim que t_1 e t_2 possam ser executadas novamente

Figura 4.4: *Exemplo de um oferecimento adaptativo.*

	id. do usuário	Status reserva	Modelo do carro	Categoria do carro	Período reserva	Doc. do usuário
WED-state 0	*null*	requisitada	*null*	*null*	*null*	*null*
WED-state 1	xyz	**iniciada**	*null*	*null*	*null*	*null*
WED-state 2	xyz	iniciada	*null*	*null*	*null*	**xyz.pdf**
WED-state 3	xyz	iniciada	*null*	*null*	*null*	***null***
WED-state 4	xyz	iniciada	*null*	*null*	*null*	**xyz.pdf**
WED-state 5	xyz	iniciada	**W3**	**B-1**	**1-1-13 a 1-2-13**	xyz.pdf

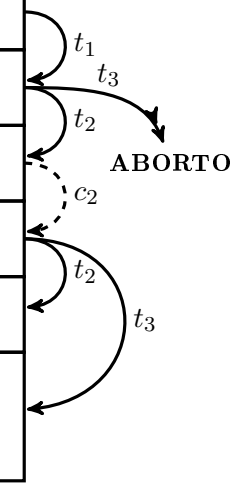


Figura 4.5: *Exemplo da execução de duas WED-transitions em uma instância de um WED-flow modelado a partir do cenário de aluguel de carros, disponível na Seção 2.4. t_1, t_2 e t_3 representam as WED-transitions "Inicializar reserva", "Enviar documentos" e "Escolher carro", respectivamente e c_2 representa a WED-compensation "Compensação escolher carro".*

4.3.5 Encadeamento de compensações

O administrador do sistema pode optar pela recuperação por meio do encadeamento de uma ou mais WED-compensations (ver Definição 4.3.3) sempre que o WED-state atual da interrupção é consistente.

O mecanismo de recuperação recebe como entrada o número de WED-compensations a serem executadas ou uma condição de parada. A partir do WED-state atual da interrupção e com o auxílio do histórico de execução, são executadas as WED-compensations de cada uma das WED-transitions que foram executadas com sucesso e os WED-states gerados pelas WED-compensations vão sendo usados como WED-states de entrada para as próximas WED-compensations. Essa sequência de WED-compensations continua a ser executada até que uma das seguintes situações seja verificada:

- O número de compensações passado como entrada para o encadeamento de compensações seja atingido;
- A condição de parada passada como entrada para o encadeamento de compensações seja satisfeita;
- Um WED-state equivalente ao WED-state inicial da instância seja gerado.

Uma vez concluída a execução das compensações e outros mecanismos de recuperação que utilizam WED-states consistentes como entrada podem ser executados. É possível, então, executar um oferecimento adaptativo (ver Seção 4.3.4) e reiniciar a execução, ou, ainda, caso seja necessário, executar um WED-S^{+a} (ver Seção 4.3.7).

WED-compensations são preferivelmente definidas no momento da modelagem do WED-flow, entretanto, assim como as demais definições de um WED-flow, podem ser acrescentadas a qualquer momento do ciclo de vida dele. Caso haja WED-transitions sem WED-compensations definidas, o

administrador poderá acrescentá-las ao modelo também durante a interrupção e, em seguida, usar a recuperação por WED-compensations encadeadas. Se o administrador não puder ou não quiser acrescentá-las, a única opção para retornar a instância a um estado consistente é por meio de uma recuperação *forward*, que será explicada a seguir.

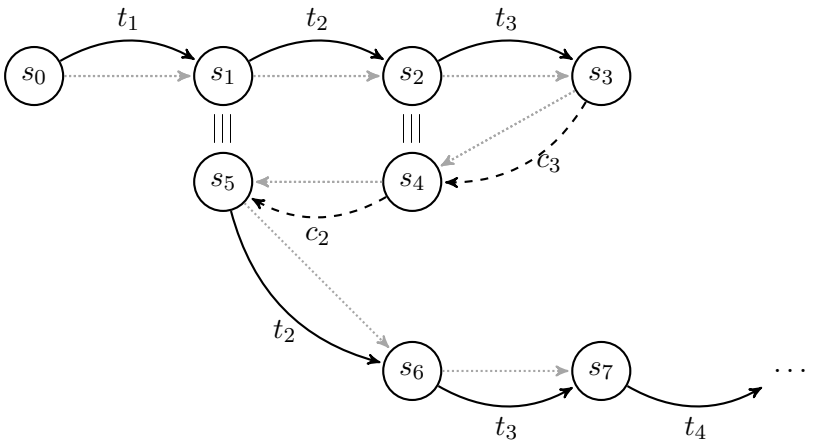


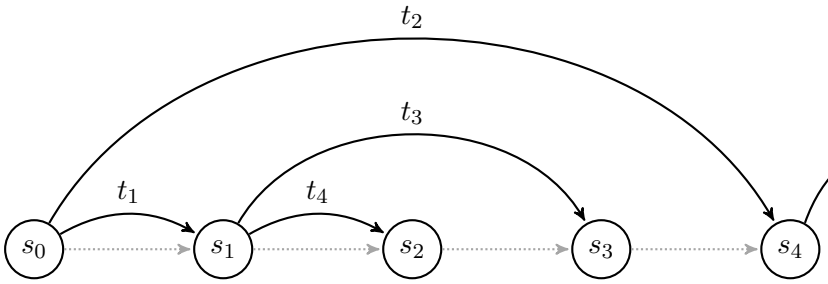
Figura 4.6: *Exemplo da execução de um encadeamento de compensações.*

A Figura 4.6 ilustra um encadeamento de compensações executado a partir de um WED-state consistente s_3 e que tem como condição de parada a geração de um WED-state equivalente a s_1. Quando s_5, o qual satisfaz a condição de parada, é gerado, é feito um oferecimento adaptativo dele que resulta no reinício da execução da instância com o disparo de t_2.

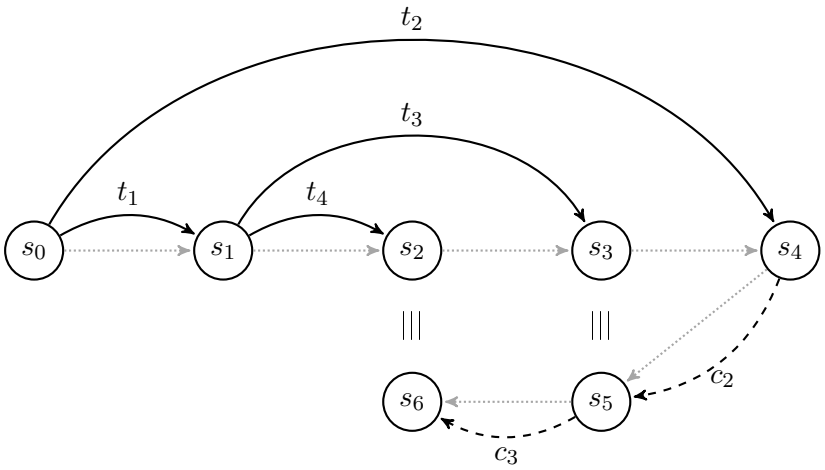
A Figura 4.7 ilustra um encadeamento de compensações executado a partir de um WED-state s_4, com condição de parada a geração de um WED-state equivalente a s_2. Quando esse fim é alcançado, o WED-state $s_6 \equiv s_2$ sofre um oferecimento adaptativo que resulta em mais duas compensações gerando o WED-state $s_8 \equiv s_0$. s_8 é então oferecido aos WED-triggers da instância.

A Figura 4.8 mostra a execução de uma instância de um WED-flow modelado a partir do cenário de aluguel de carros (Seção 2.4). Uma falha de execução na WED-transition t_6 ("Registrar devolução e vistoriar") resulta na geração de uma exceção, a qual o administrador de sistemas decide tratar com a execução de um encadeamento de compensações que tem a geração de um WED-state equivalente a S0 como condição de parada.

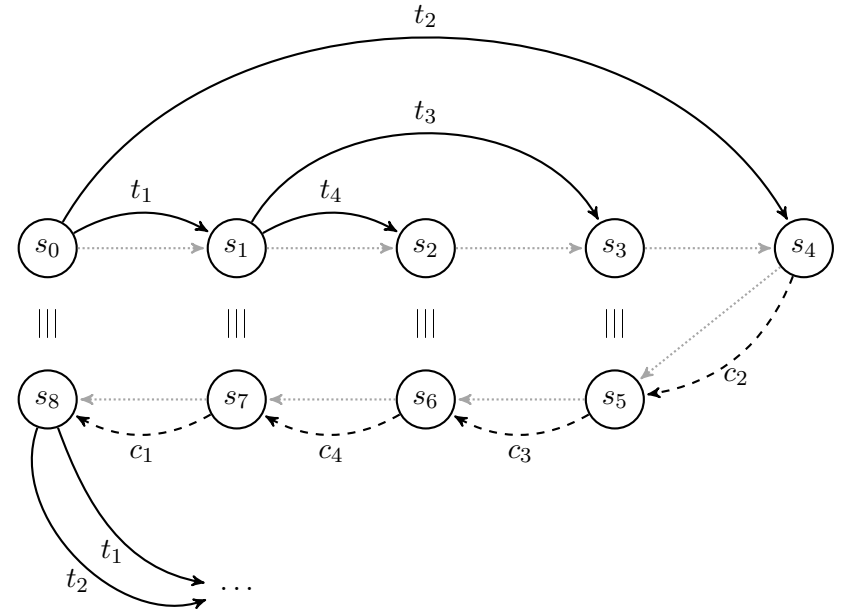
O mecanismo do encadeamento de compensações foi apresentado de maneira sucinta e inicial em Ferreira et al. [FBTP12]. As contribuições mais relevantes de nosso trabalho, no sentido de aprimorar esse mecanismo, são a definição da condição de parada baseada no número de WED-transitions compensadas e, principalmente, a definição de um mecanismo – o oferecimento adaptativo – capaz de dar continuação, de maneira **consistente**, à execução da instância interrompida após o término com sucesso de um encadeamento de compensações.



a- A interrupção da instância causou o aborto de t_5 que estava em execução.

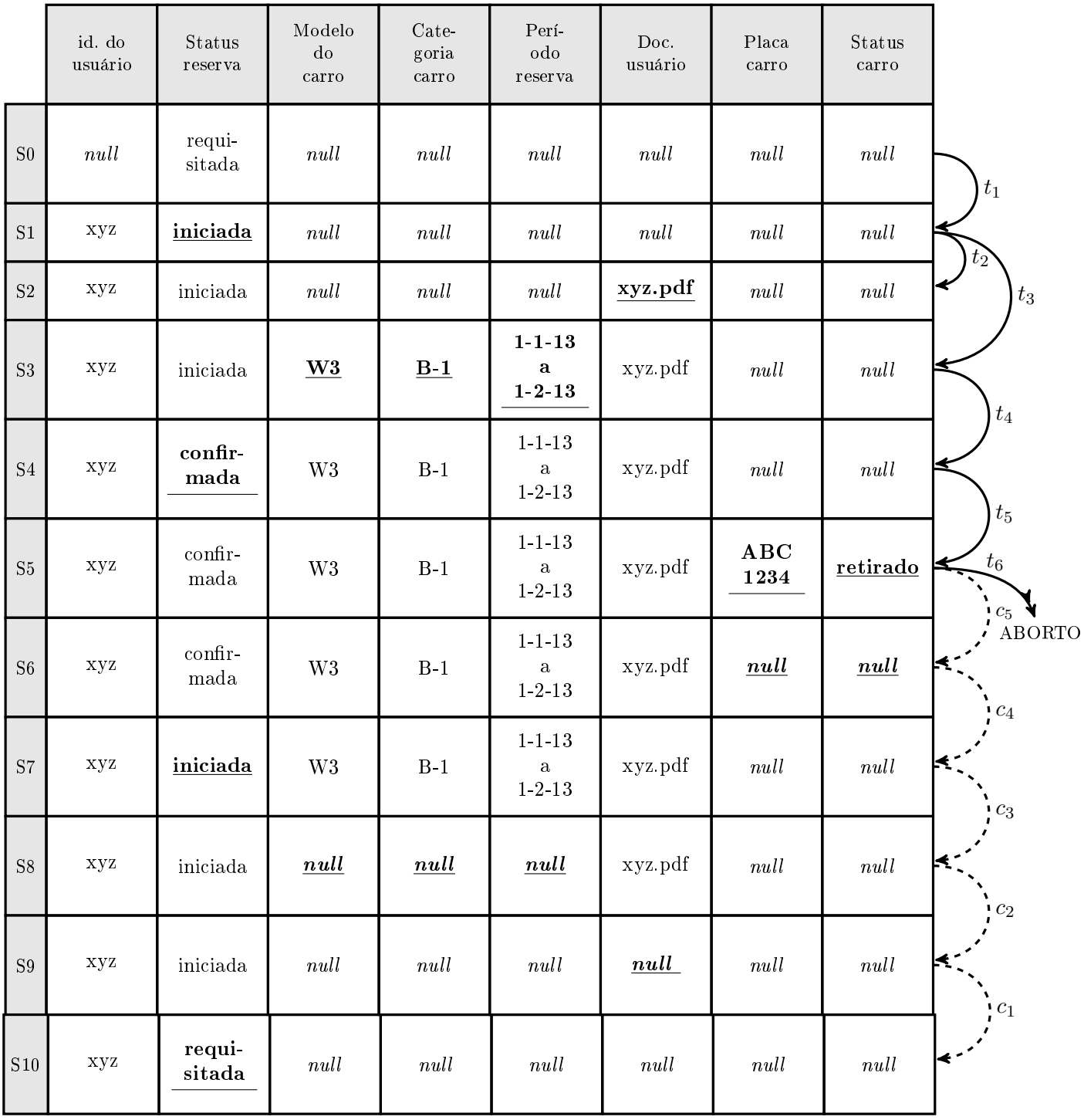


b- É executada uma compensação encadeada com a geração de um WED-state equivalente a s_2 como condição de parada.



c- É feito o oferecimento adaptativo do WED-state s_6 que resulta numa sequência de compensações a qual acaba por gerar $s_8 \equiv s_0$ que é oferecido aos WED-triggers da instância.

Figura 4.7: *Exemplo mais complexo de encadeamento de compensações.*



	id. do usuário	Status reserva	Modelo do carro	Categoria carro	Período reserva	Doc. usuário	Placa carro	Status carro
S0	*null*	requisitada	*null*	*null*	*null*	*null*	*null*	*null*
S1	xyz	**iniciada**	*null*	*null*	*null*	*null*	*null*	*null*
S2	xyz	iniciada	*null*	*null*	*null*	**xyz.pdf**	*null*	*null*
S3	xyz	iniciada	**W3**	**B-1**	**1-1-13 a 1-2-13**	xyz.pdf	*null*	*null*
S4	xyz	**confirmada**	W3	B-1	1-1-13 a 1-2-13	xyz.pdf	*null*	*null*
S5	xyz	confirmada	W3	B-1	1-1-13 a 1-2-13	xyz.pdf	**ABC 1234**	**retirado**
S6	xyz	confirmada	W3	B-1	1-1-13 a 1-2-13	xyz.pdf	***null***	***null***
S7	xyz	**iniciada**	W3	B-1	1-1-13 a 1-2-13	xyz.pdf	*null*	*null*
S8	xyz	iniciada	***null***	***null***	***null***	xyz.pdf	*null*	*null*
S9	xyz	iniciada	*null*	*null*	*null*	***null***	*null*	*null*
S10	xyz	**requisitada**	*null*	*null*	*null*	*null*	*null*	*null*

Figura 4.8: *Exemplo da execução de uma instância de um WED-flow modelado a partir do cenário de aluguel de carros (Seção 2.4). A execução flui normalmente até que uma exceção é detectada durante a execução da WED-transition t_6, motivo pelo qual um encadeamento de compensações é executado. t_1, t_2, t_3, t_4, t_5 e t_6 representam as WED-transitions "Inicializar reserva", "Enviar documentos", "Escolher carro", "Encaminhar para validação do gerente", "Registrar retirada" e "Registrar devolução e vistoriar", respectivamente. Cada c_i representa a WED-compensation associada a cada WED-transition t_i.*

4.3.6 Alteração da definição de um WED-flow

Durante o momento de tratamento de exceções, o administrador do sistema pode identificar uma situação que não é exatamente uma exceção, mas, na verdade, uma regra de negócio que deveria fazer parte da definição do WED-flow, porém não foi pensada no momento da modelagem. Apesar de ser possível alterar as definições de um WED-flow a qualquer momento de seu ciclo de vida, é principalmente durante o tratamento de exceções que as falhas de modelagem poderão ser mais facilmente detectadas.

4.3.7 Pulos *backward* e *forward*

Pulos são mecanismos de Modelos Transacionais Avançados que, por meio de alterações no estado atual de instâncias de processos de negócio ou de workflows, permitem, de maneira resumida, gerar um determinado estado da instância que normalmente não seria gerado pela execução de uma única atividade. No caso de pulos *backward*, o estado atual da instância é alterado de forma a coincidir com um estado anterior da instância. É como se a execução "pulasse" do estado atual diretamente para um dado estado anterior sem passar por nenhum outro estado intermediário. Pulos *forward* alteram o estado atual com o objetivo de gerar um estado inédito da instância. São comuns em casos em que é necessário ignorar atividades ou gerar caminhos alternativos.

A abordagem WED-flow prevê os dois pulos descritos acima: O pulo *forward* é chamado de WED-S^{+a} e o pulo *backward*, WED-S^{-1}. Eles foram apresentados de maneira sucinta e inicial em [FBTP12] e têm em comum a necessidade de que o administrador do sistema defina um tipo especial de WED-transition, no momento do tratamento da interrupção, que, a partir do WED-state atual da interrupção, gere um novo WED-state consistente. Como veremos a seguir, WED-S^{-1}s só operam sobre WED-states atuais da interrupção inconsistentes, enquanto que WED-S^{+a}s operam sobre WED-states atuais da interrupção consistentes e inconsistentes.

WED-S^{-1}

Quando o WED-state atual da interrupção é inconsistente, os métodos de oferecimento adaptativo e de encadeamento de compensações não podem ser executados pois prevêem a utilização de WED-compensations, as quais se executadas sobre um WED-state inconsistente violam sua definição (Seção 4.3.3). No entanto, a instância que teve a execução interrompida precisa prosseguir e, para isso, precisa voltar a ser consistente.

O WED-S^{-1}, que só pode ser executado a partir de WED-states inconsistentes, tem por objetivo gerar um WED-state consistente equivalente a algum outro WED-state consistente presente no histórico de execução da instância interrompida. A partir desse WED-state, que é necessariamente consistente, o administrador do sistema pode executar métodos de recuperação que utilizam WED-states atuais de interrupção consistentes como entrada (encadeamento de compensações, oferecimento adaptativo, WED-S^{+a}).

A Figura 4.9 ilustra a execução de um WED-S^{-1} sobre s_3, um WED-state inconsistente.

A Figura 4.10 ilustra um exemplo de parte da execução de uma instância do WED-flow baseada no cenário de aluguel de carros (Seção 2.4). Para simular a geração de um WED-state inconsistente, estamos supondo que devido a algum problema na WED-transition t_5 ("Registrar retirada"), o WED-state S5, representado com uma cor mais escura, seja inconsistente por não satisfazer a WED-

condition de nenhum WED-trigger do WED-flow. Ao se deparar com essa situação, o administrador decide então executar um WED-S^{-1} que gera o WED-state S6 que é equivalente ao WED-state S0.

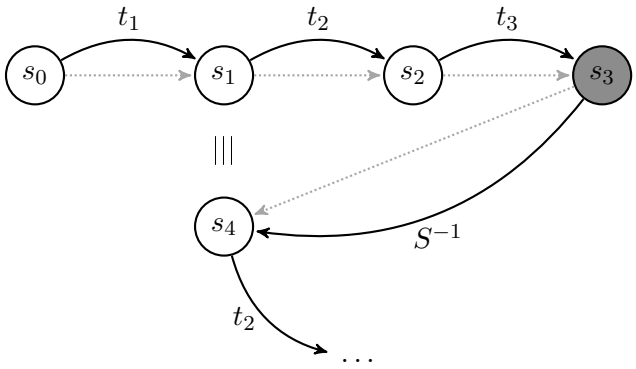


Figura 4.9: *Exemplo da execução de um WED-S^{-1} seguido de um oferecimento adaptativo.*

	id. do usuário	Status reserva	Modelo do carro	Categoria carro	Período reserva	Doc. usuário	Placa carro	Status carro
S0	*null*	requisitada	*null*	*null*	*null*	*null*	*null*	*null*
S1	xyz	**iniciada**	*null*	*null*	*null*	*null*	*null*	*null*
S2	xyz	iniciada	*null*	*null*	*null*	**xyz.pdf**	*null*	*null*
S3	xyz	iniciada	**W3**	**B-1**	**1-1-13 a 1-2-13**	xyz.pdf	*null*	*null*
S4	xyz	**confirmada**	W3	B-1	1-1-13 a 1-2-13	xyz.pdf	*null*	*null*
S5	xyz	confirmada	W3	B-1	1-1-13 a 1-2-13	xyz.pdf	**ABC 1234**	**inválido**
S6	xyz	**requisitada**	***null***	***null***	***null***	***null***	***null***	***null***

Figura 4.10: *Exemplo de parte da execução de uma instância de um WED-flow modelado a partir do cenário de aluguel de carros (Seção 2.4). A execução flui normalmente até que um WED-state inconsistente (S5) é gerado. Para resolver essa situação, o administrador de sistemas opta pela execução de um WED-S^{-1}. t_1, t_2, t_3, t_4 e t_5 representam as WED-transitions "Inicializar reserva", "Enviar documentos", "Escolher carro", "Encaminhar para validação do gerente" e "Registrar retirada", respectivamente.*

O WED-S^{-1} foi apresentado de forma sucinta e inicial em Ferreira et al. [FBTP12]. A seguir citamos as principais contribuições de nosso trabalho para o aprimoramento desse mecanismo:

- **Limite operacional:** Definimos que um WED-S^{-1} deve gerar, obrigatoriamente, um WED-state **consistente** e **equivalente** a algum outro WED-state consistente existente no histórico de execução da instância interrompida.
- **Mecanismos em sequência:** Ampliamos as opções de mecanismos disponíveis ao administrador de sistema que podem ser executados após o término com sucesso de um WED-S^{-1}. É possível, a partir do WED-state consistente gerado por esse mecanismo, executar um encadeamento de compensações ou, ainda, executar um oferecimento adaptativo e reiniciar a execução da instância interrompida.

WED-S^{+a}

O WED-S^{+a} pode ser executado a partir de WED-states consistentes e inconsistentes. Ele tem como objetivo gerar um WED-state consistente que não seja equivalente a nenhum dos WED-states presentes no histórico de execução da instância que teve a execução interrompida. Uma vez gerado o novo WED-state, a interrupção é finalizada e ele é então oferecido a todos os WED-triggers do WED-flow.

A figura 4.11 ilustra a execução de um S^{+a}. s_2, que pode ser tanto um WED-state consistente quanto inconsistente, serve de entrada para a execução de um WED-S^{+a}, que gera um WED-state S_x que não é equivalente a nenhum dos outros WED-states anteriores.

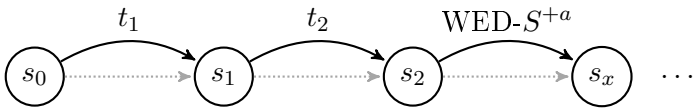


Figura 4.11: *Exemplo da execução de um S^{+a} seguido de um oferecimento adaptativo. É importante ressaltar que s_x não é equivalente a nenhum outro WED-state presente no histórico de execução.*

A Figura 4.12 ilustra parte da execução de uma instância de um WED-flow baseado no cenário de aluguel de carros (Seção 2.4). Uma exceção é levantada devido a presença de uma avaria não prevista (Cor diferente da original) pela WED-transition t_2 ("Computar multas"). O administrador de sistema decide, em vez de criar WED-triggers para essa situação bastante particular, executar um WED-S^{+a} que resolva a situação pontualmente, calculando um valor para a multa correspondente a essa avaria. O WED-state 2 gerado pelo WED-S^{+a} não é equivalente a nenhum WED-state presente na execução.

Uma primeira versão do WED-S^{+a} foi apresentado de maneira sucinta e inicial em Ferreira et al. [FBTP12]. As contribuições mais relevantes de nosso trabalho para o aprimoraramento desse mecanismo são a definição de que o WED-state gerado não deve ser equivalente a nenhum WED-state presente no histórico de execução da instância que sofre recuperação e que o WED-state atual da interrupção pode ser consistente e inconsistente.

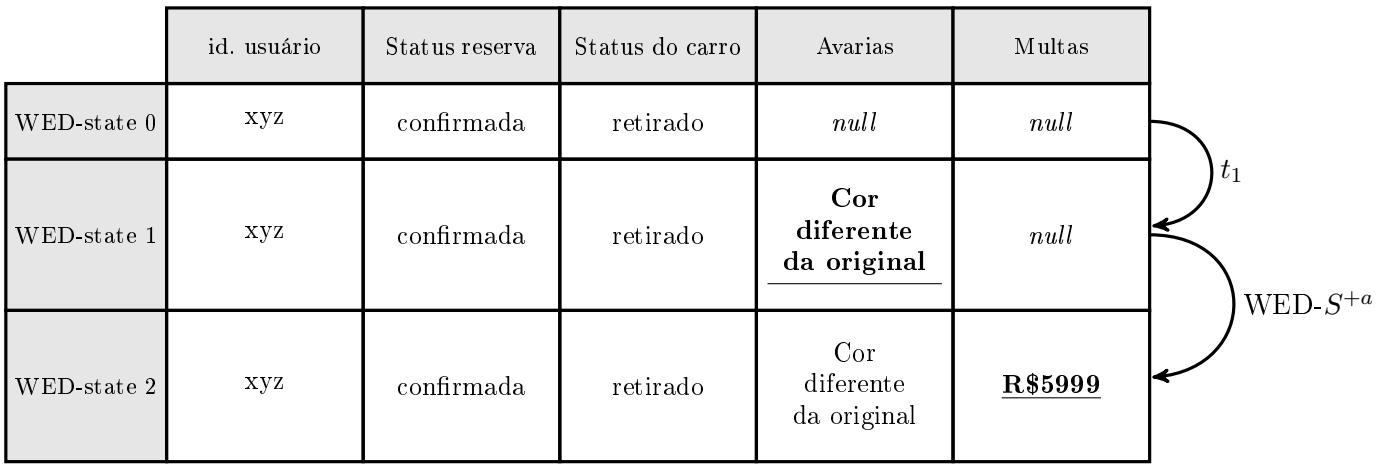


Figura 4.12: *Exemplo de parte da execução de uma instância de um WED-flow modelado a partir do cenário de aluguel de carros (Seção 2.4). t_1 e t_2 representam as WED-transitions "Registrar devolução e vistoriar" e "Computar multas", respectivamente. O WED-S^{+a} tem por objetivo resolver a situação do tipo de avaria imprevisto, ou seja, que não tem uma multa definida.*

4.4 Sumário

Neste capítulo apresentamos nossas principais contribuições para o aprimoramento do tratamento de exceções da abordagem WED-flow. As definições dos tipos de interrupção e do conceito de equivalência de WED-states nos permitiram compreender melhor o tratamento de exceções e serviram de base para o melhoramento dos mecanismos de recuperação que já haviam sido definidos. Elas foram importantes também para a criação do oferecimento adaptativo, que é um mecanismo essencial para que o reinício da execução de uma instância interrompida aconteça de forma consistente.

Consideramos que, com nossas contribuições, o tratamento de exceções da abordagem WED-flow ficou mais completo e flexível. No Capítulo 5, um protótipo do gerenciador de recuperação da ferramenta WED-tool [GSBF12], que implementa os mecanismos de tratamento de exceções aqui descritos, foi desenvolvido como forma de testar a viabilidade prática de nossas propostas.

Capítulo 5

Implementação do gerenciador de recuperação da ferramenta WED-tool

Consideramos necessário que a forma de tratamento de exceções proposta no Capítulo 4 seja avaliada na prática, por isso, nos propusemos a implementar o gerenciador de recuperação da ferramenta **WED-tool** [GSBF12, Gar13], que é um protótipo de um sistema de gestão de processos de negócio que implementa a abordagem WED-flow. Essa ferramenta, atualmente, está sendo desenvolvida de forma concomitante por diversos alunos e pesquisadores do grupo DATA IME-USP [DIa] no âmbito de seus trabalhos acadêmicos.

5.1 WED-tool - Funcionamento Básico

Atualmente a versão protótipo da WED-tool já tem algumas das principais funcionalidades de um sistema de gestão de processos de negócio:

- A ferramenta é capaz de ler e interpretar modelos de processos de negócio;
- Uma vez que os modelos de processos de negócio foram lidos e interpretados, a WED-tool é capaz de instânciá-los e controlar a execução de suas instâncias;
- Caso algum erro seja detectado durante a execução de alguma instância, a ferramenta possui um gerenciador de recuperação com a função de realizar o tratamento de exceções e dar prosseguimento a sua execução.

Na Figura 5.1 representamos as principais etapas que descrevem o funcionamento da ferramenta WED-tool e que são necessárias para instanciar um WED-flow e executar uma instância dele.

A primeira etapa é a modelagem de um WED-flow. Nela, um projetista é encarregado de definir os modelos dos WED-flows (processos de negócio) em XML, de definir algumas variáveis de configuração do sistema e, utilizando a linguagem de programação *Ruby*, implementar as WED-transitions e WED-compensations dos WED-flows.

Na etapa de configuração, a WED-tool lê e interpreta o modelo de um WED-flow para gerar a estrutura (em memória e no banco de dados) necessária para que seja possível instânciá-lo e executar suas instâncias.

A próxima etapa é a instanciação de WED-flows, que é solicitada por um administrador de sistema por meio da interface da ferramenta. Instâncias de WED-flow têm sua execução controlada pela WED-tool de forma a gerenciar a geração de WED-states, avaliar WED-conditions sobre esses estados e disparar as WED-transitions associadas a essas condições caso elas sejam satisfeitas. Se alguma falha for detectada durante a execução de alguma instância, assume a execução o gerenciador de recuperação, o qual, como apresentamos na Seção 4.3, emprega uma abordagem *ad hoc*, ou seja, que depende da intervenção de um administrador para a escolha e execução de métodos de tratamento de exceções. Uma vez executado(s) o(s) método(s) de tratamento de exceções, a execução é reiniciada.

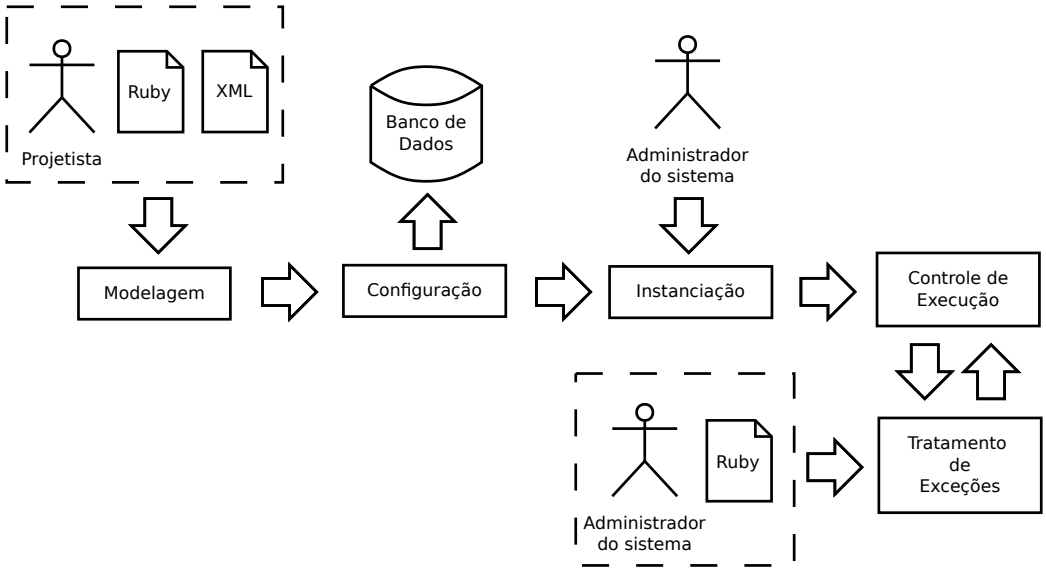


Figura 5.1: *Etapas da ferramenta WED-tool para a execução de instâncias de WED-flows.*

5.2 Arquitetura

Na Figura 5.2 temos a representação da arquitetura da versão atual da WED-tool. Nesta seção, explicaremos de forma geral cada um dos módulos nela representados e a seguir, na Seção 5.3, explicaremos em mais detalhes o Gerenciador de Recuperação.

A WED-tool foi desenvolvida de forma a beneficiar-se dos principais arcabouços e ferramentas *open source* disponíveis na atualidade. O Núcleo de Controle, a Interface com o usuário e o Gerenciador de Recuperação (GEREC) foram desenvolvidos usando a linguagem *Ruby* [Rub], a qual permitiu a fácil utilização do arcabouço *Active Record* [Rec]. Esse último, por sua vez permite a manipulação dos bancos de dados, no nosso caso, bancos de dados PostgreSQL [Pos], por meio de uma abstração de alto nível e mais natural, livrando o desenvolvedor da necessidade de escrever consultas SQL diretamente no código.

O Núcleo de Controle é o cerne da ferramenta e é responsável pela interpretação dos modelos dos WED-flows, pela instanciação deles e pelo controle de execução de suas instâncias. Esses modelos são definidos parcialmente em XML e parcialmente em *Ruby*: a estrutura das componentes de um WED-flow, assim como a forma como elas se relacionam, são definidas em XML, enquanto que o código

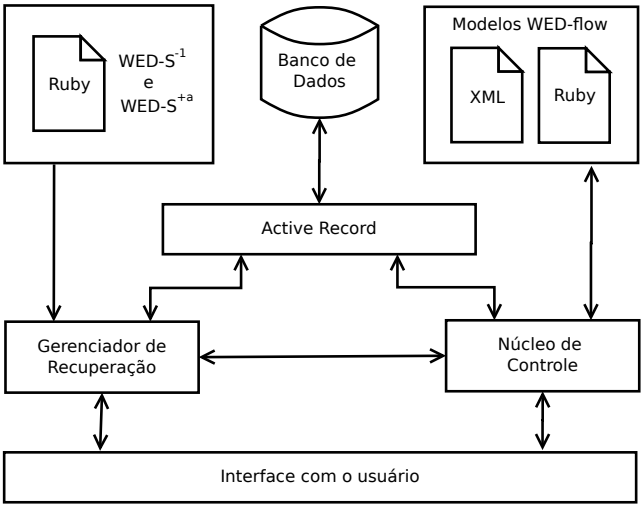


Figura 5.2: *Arquitetura da versão atual da ferramenta WED-tool*

a ser executado em cada WED-transition e WED-compensation definidas na modelagem é feito em *Ruby*. Como o Núcleo de Controle é a principal contribuição de outros trabalhos [Mar12, Gar13], nós não entraremos nos detalhes de sua implementação.

A Interface com o usuário da WED-tool atualmente é uma aplicação baseada em interação com um *prompt* de comando, o qual permite a um administrador de sistema executar tarefas de gerenciamento como instanciação de WED-flows, exibição de detalhes de instâncias em execução e tratamento de exceções.

O Gerenciador de Recuperação da WED-tool (GEREC), o qual será abordado em mais detalhes na Seção 5.3), é o módulo da ferramenta responsável por fazer o tratamento de exceções. Ele implementa vários métodos de tratamento de exceções propostos no Capítulo 4, mais explicitamente, o encadeamento de compensações, o oferecimento adaptativo e os pulos *forward* e *backward*.

5.2.1 Componentes WED-flow de tratamento de exceções

Como ilustrado na Fig. 5.2, modelos de WED-flow são definidos em XML e *Ruby*. Nesta seção, apresentaremos a estrutura das definições das componentes de um WED-flow relacionadas ao tratamento de exceções.

WED-compensations, WED-S^{-1}s e WED-S^{+a}s podem ser interpretadas como tipos especiais de WED-transitions já que elas também geram um novo WED-state para a instância a partir de um WED-state inicial. Para definir uma WED-transition, o projetista precisa realizar os seguintes passos:

(1) Especificar no modelo do WED-flow, em XML, o nome da WED-transition e os WED-attributes que serão alterados por ela para a geração de um novo WED-state.

(2) Preencher uma classe em *Ruby* gerada pela WED-tool que represente a WED-transition e que tenha por objetivo descrever as transformações a serem aplicadas aos WED-attributes do WED-state atual da instância que a executa.

Os trechos de código Trecho 5.1 e Trecho 5.2 exemplificam o processo de modelagem descrito acima. Na dissertação de Marcela Ortega [Gar13], as demais componentes do modelo não relacionadas ao tratamento de exceções são explicadas em mais detalhes. No Apêndice A.2, disponibilizamos um exemplo de modelo inicial de WED-flow em XML e *Ruby*, e uma coleção de exemplos de execuções completas de instâncias de WED-flow.

```
1 <WED-transitions>
2   <Transition Name="t_wed_transition_name" >
3     <UpdatedAttribute AttrName="attr_1" />
4     <UpdatedAttribute AttrName="attr_2" />
5   </Transition>
6   ....
7 </WED-transitions>
```

Trecho 5.1: *Trecho de código XML que define uma WED-transition.*

```
1 class TWedTransitionName
2         def self.run(initial_state)
3                 ...
4                 return {:attr_1 => var_1, :attr2 => var_2}
5         end
6 end
```

Trecho 5.2: *Classe Ruby que descreve as transformações sobre os WED-attributes de um WED-state inicial para a geração de um novo WED-state.*

WED-compensations são as únicas componentes de um WED-flow relacionadas ao tratamento de exceções que são descritas na definição do modelo inicial. De forma análoga às WED-transitions, o projetista precisa especificar na definição do WED-flow o nome da WED-compensation, os atributos a serem alterados por ela para a geração de um novo WED-state e, além disso, definir o nome da WED-transition a qual a compensação se associa (Trecho 5.3). Após essa primeira fase, o projetista deve definir uma classe em *Ruby* na qual são descritas as transformações a serem aplicadas aos WED-attributes do WED-state atual da interrupção (Trecho 5.4). Note que os arquivos *Ruby* não são gerados automaticamente como acontece no caso das WED-transitions, cabendo ao projetista a responsabilidade também de criá-los. Conforme discutido na Seção 4.3.5, WED-compensations não precisam ser obrigatoriamente definidas e WED-states gerados por WED-compensations precisam ser equivalentes aos WED-states sobre os quais suas WED-transitions associadas fizeram alterações.

```
1 <WED-compensations>
2     <Compensation Name="compensation_name" ForWedTransition="transition_name">
3       <UpdatedAttribute AttrName="attribute_1" />
4       <UpdatedAttribute AttrName="atrribute_2" />
5     </Compensation>
6     ....
7 <WED-compensations>
```

Trecho 5.3: *Trecho de código XML que define uma WED-compensation.*

```
1 class CompensationName
2         def self.run(initial_state)
3                 ...
4                 return {:attribute_1 => var_1, :atrribute_2 => var_2}
5         end
6 end
```

Trecho 5.4: *Classe Ruby que descreve as transformações sobre os WED-attributes do WED-state atual da interrução para a geração de um novo WED-state.*

A definição de **WED-S^{-1}s** (pulos *backward*) e **WED-S^{+a}s** (pulos *forward*) não é feita no modelo inicial dos WED-flows uma vez que pulos só são utilizados para o tratamento pontual de exceções inesperadas, ou seja, situações excepcionais que não foram descritas na modelagem (ver Seção 4.3.7). A definição de ambos é quase idêntica: O administrador do sistema precisa criar uma classe *Ruby* que aplique transformações a WED-attributes do WED-state atual da interrupção (Trecho 5.5) a fim de gerar um novo WED-state. A diferença entre os métodos, como explicamos na Seção 4.3.7, se resume às características desse novo WED-state gerado.

```
1 class JFwdPulaVistoriaEComputaMultas
2     def self.run(initialState)
3         return {:multas => 1000.0,
4                 :status_carro => 'devolvido', :avarias => 'avarias'}
5     end
6 end
```

Trecho 5.5: *Classe Ruby que descreve as transformações sobre os WED-attributes do WED-state atual da interrução para a geração de um novo WED-state.*

5.3 GEREC

Antes de apresentarmos o Gerenciador de Recuperação da WED-tool (GEREC) em mais detalhes, consideramos importante analisar rapidamente a forma como o Núcleo de Controle [Gar13] detecta exceções e interrompe a execução de uma instância.

Utilizando os conceitos apresentados na Seção 2.3.5, podemos dizer que o Núcleo de Controle da versão atual da ferramenta WED-tool é capaz de detectar **exceções esperadas** e **exceções não esperadas**.

Optou-se por implementar a detecção de exceções esperadas de forma implícita, por meio da definição, no modelo do WED-flow, de um WED-trigger que tem associada uma WED-condition que identifica a situação excepcional e uma WED-transition que trata a exceção caso a condição seja satisfeita. Logo, não existe diferença sintática entre um evento "normal" e uma exceção esperada e dessa forma, toda exceção esperada detectada pela WED-tool é consequentemente conhecida, detectável e solucionável. Quanto às exceções não esperadas, a implementação atual é capaz de identificar falhas na execução de WED-transitions e de detectar WED-states inconsistentes (ver Seção 2.5.10), situações essas que, ao serem identificadas, resultam na interrupção imediata da execução da instância problemática. Interrompida a instância, o administrador do sistema, por meio do *menu* do GEREC disponível na interface da WED-tool, pode escolher e iniciar os métodos de

tratamento de exceções mais adequados para a situação. Resumindo, o GEREC só pode ser ativado no caso da detecção de uma exceção não esperada e exceções esperadas são tratadas implicitamente pelo Núcleo de Controle.

5.3.1 Escolha dos mecanismos de tratamento de exceções

Conforme discutido na Seção 5.3, o GEREC só pode ser acionado após a detecção de uma exceção não esperada, durante a execução de uma instância de WED-flow, e a consequente interrupção dessa instância. Nesse momento, é necessário que um administrador do sistema acesse a interface da ferramenta WED-tool e por meio do *menu* do GEREC, disponível na interface da ferramenta, escolha o método de tratamento de exceções mais adequado para a situação. Informações adicionais sobre instâncias em execução e interrompidas podem ser acessadas por meio do *menu* que o Núcleo de Controle disponibiliza na interface da ferramenta [Gar13]. Assim, o projetista pode levantar o máximo de informação possível sobre a exceção ocorrida e sobre a execução da instância em geral garantindo mais precisão à sua escolha.

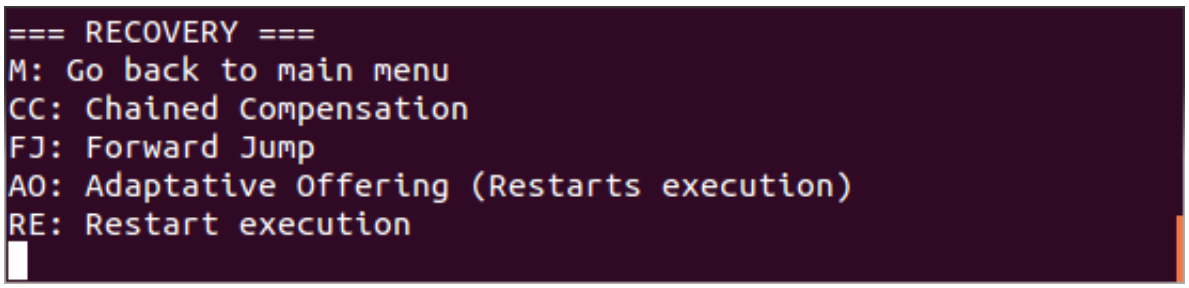


Figura 5.3: *Cópia de tela do menu da ferramenta WED-tool na qual são exibidas as opções de recuperação em caso de WED-state atual da interrupção consistente.*

Dependendo do tipo de interrupção (ver Seção 4.2), diferentes métodos de recuperação estarão à disposição do administrador do sistema. Caso o WED-state atual da interrupção seja consistente, os métodos à disposição do administrador exibidos na interface da WED-tool serão (Fig. 5.3) os seguintes:

- Compensação encadeada (ver Seção 4.3.5): Ao escolher esse método o administrador do sistema pode definir uma ou duas condições de parada que serão avaliadas de forma conjuntiva. São elas o número de WED-compensations a serem executadas e o identificador de um WED-state final que deve ser equivalente (ver Seção 4.3.1) a algum dos WED-states gerados pelas WED-compensations. Uma terceira condição de parada que é sempre considerada implicitamente é a de que só se pode compensar até a primeira WED-transition executada na instância (Fig. 5.4);

- Pulo *Forward* - WED-S^{+a} (ver Seção 4.3.7): Esse método de recuperação exige que o administrador do sistema insira o nome do arquivo que contém a classe *Ruby* (Fig. 5.5) que descreve as alterações que o WED-S^{+a} aplicará sobre o WED-state atual da interrupção para gerar um novo WED-state;

- Oferecimento adaptativo (ver Seção 4.3.4): Esse método executa um oferecimento adaptativo do WED-state atual da interrupção reiniciando a execução.

```
=== RECOVERY ===
M: Go back to main menu
CC: Chained Compensation
FJ: Forward Jump
AO: Adaptative Offering (Restarts execution)
RE: Restart execution
CC

=== Chained Compensation ===
R: Go back to recovery menu

You may provide more than one stop conditions to this method. They will be
evaluated as an 'AND' expression:

NC: Number of compensations to be executed (if you prefer not to provide it
, all executions from this instance will be compensated)

FS: You can pass a WED-state id whose object will be used as stop condition
. (if you prefer not to provide it, all executions from this instance will
be compensated)

OK: Execute chained compensation
```

Figura 5.4: *Cópia de tela do menu da ferramenta WED-tool na qual é exibido o submenu do método de encadeamento de compensações.*

```
=== RECOVERY ===
M: Go back to main menu
CC: Chained Compensation
FJ: Forward Jump
AO: Adaptative Offering (Restarts execution)
RE: Restart execution
FJ

=== Forward Jump ===
R: Go back to recovery menu
You must provide a jump transition name corresponding to the
WED-jump name you defined.
```

Figura 5.5: *Cópia de tela do menu da ferramenta WED-tool na qual é exibido o submenu do método WED-S^{+a}.*

Caso o WED-state atual da interrupção seja inconsistente, o administrador do sistema tem a sua disposição o WED-S^{-1} e o WED-S^{+a} (Fig. 5.6). O WED-S^{+a} funciona de forma idêntica à situação em que o WED-state atual da interrupção é consistente, conforme explicado acima. O WED-S^{-1} funciona de maneira similar ao WED-S^{+a} e, portanto, é necessário que o administrador do sistema insira o nome do arquivo que contém a classe *Ruby*, que implementa as transformações necessárias para que o WED-S^{-1} gere um novo WED-state. Concluída a execução do pulo *backward*, caso um WED-state consistente tenha sido gerado com sucesso durante esse processo, os métodos de tratamento de exceções para o caso de WED-state atual da interrupção consistente (Fig. 5.3)

são exibidos no *menu* da interface da ferramenta e o administrador do sistema pode, antes de reiniciar a execução da instância, executar algum deles. Por exemplo, o administrador do sistema pode executar um WED-S^{-1} seguido de um encadeamento de compensações.

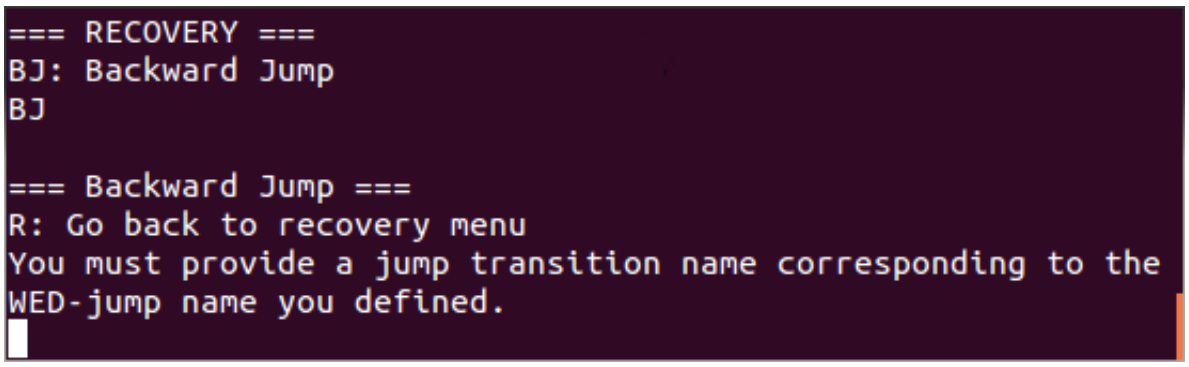


Figura 5.6: *Cópia de tela do menu da ferramenta WED-tool na qual são exibidas as opções de recuperação em caso de WED-state atual da interrupção inconsistente juntamente ao submenu do método WED-S^{-1}.*

5.3.2 Histórico de execução

Para que funcionem corretamente, os métodos de tratamento de exceções precisam de detalhes da execução das instâncias que geraram exceções. Por exemplo, uma WED-compensation precisa ter acesso aos WED-states que foram gerados na instância, às WED-transitions que os geraram e à ordem em que isso ocorreu. Da mesma forma, um WED-S^{+a} precisa ter acesso a todos os WED-states gerados na instância para assegurar que o WED-state gerado por ele não tenha nenhum equivalente. Esse tipo de informação, como discutimos na Seção 2.5.11, é disponibilizado pelo histórico de execução, que na versão atual da ferramenta foi implementado como uma tabela no banco de dados. Mais detalhes sobre a implementação do histórico de execução podem ser encontrados na dissertação de Marcela Ortega [Gar13].

5.4 Implementação dos mecanismos de tratamento de exceções

Nesta seção, discutiremos de forma mais detalhada a implementação dos mecanismos de tratamento de exceções e de alguns mecanismos auxiliares que os viabilizam. Faremos uso de pseudo-código quando necessário, entretanto, lembramos que o código fonte da ferramenta está disponível para *download* no sítio do WED-flow [DIb].

5.4.1 Equivalência de WED-states

O conceito de equivalência de WED-states (Seção 4.3.1), conforme discutimos ao longo do texto, foi crucial para a melhor compreensão e aprimoramento do tratamento de exceções da abordagem WED-flow.

Resumidamente, um WED-state é equivalente a um outro quando ambos habilitam a execução das mesmas WED-transitions. Uma WED-transition, por sua vez, é habilitada quando um WED-state satisfaz a WED-condition associada a ela. Assim, podemos dizer que um WED-state é equivalente a um outro quando ambos satisfazem as mesmas WED-conditions. A ideia geral de nossa implementação é testar todas as WED-conditions de um WED-flow sobre dois WED-states. Se eles satisfizerem as mesmas WED-conditions, então eles são equivalentes.

O Pseudocódigo 1 exemplifica a implementação do mecanismo que verifica se dois WED-states $ws1$ e $ws2$ são equivalentes. Para tal, é necessário, primeiramente, recuperar do banco de dados da ferramenta o conjunto C de todas as WED-conditions do WED-flow cuja instância contém $ws1$ e $ws2$, e, para cada WED-condition $wc \in C$, verificar se ambos os WED-states a satisfazem. Se existir alguma situação em que $wc(ws1) \neq wc(ws2)$, então $ws1$ não é equivalente a $ws2$. Caso contrário, ou seja, caso toda WED-condition testada sobre $ws1$ e $ws2$ tenha o mesmo resultado, então eles são equivalentes.

Pseudocódigo 1 Programa que, dado dois WED-states $ws1$ e $ws2$, retorna $Verdadeiro$ caso eles sejam **equivalentes** e $Falso$ caso contrário.

```
1:  function EQUIVALENTE?(ws1, ws2)
2:      wed_conditions ← recupera_todas_wed_conditions()
3:      for wc in wed_conditions do
4:          cond_ws1 ← wc.verifica_cond(ws1)
5:          cond_ws2 ← wc.verifica_cond(ws2)
6:          if ¬(cond_ws1 ∧ cond_ws2) ∧ ¬(¬cond_ws1 ∧ ¬cond_ws2) then
7:              return Falso
8:          end if
9:      end for
10:     return Verdadeiro
11: end function
```

5.4.2 Encadeamento de compensações

O encadeamento de compensações (Seção 4.3.5) é um mecanismo que tem o objetivo de, a partir de um WED-state atual da interrupção consistente, compensar WED-transitions, por meio da execução encadeada de WED-compensations, até que uma condição de parada seja alcançada. É importante ressaltar que cada WED-compensation precisa, por sua vez, gerar um WED-state equivalente ao WED-state usado como entrada para a WED-transition sendo compensada. Para explicar a implementação do encadeamento de compensações, iremos, primeiramente, apresentar a implementação do método que executa uma WED-compensation.

O Pseudocódigo 2 exemplifica o funcionamento de uma compensação de WED-transition. O método recebe como entrada a WED-transition wt que será compensada, o WED-state st que foi utilizado como entrada para a execução de wt e um WED-state si que será usado como entrada para a compensação. A partir de wt é possível recuperar a WED-compensation wc definida pelo projetista no modelo do WED-flow e executá-la sobre si resultando na geração de um novo WED-state sf. Se sf for equivalente a st, então sf é retornado. Caso contrário, temos uma WED-compensation mal projetada e o valor $nulo$ é retornado. Repare que nas linhas 4 e 5 é feita a atualização do histórico de execução da instância: na linha 4, é registrada a execução da WED-compensation wc e, na linha 5, a entrada relativa à execução da WED-transition wt é atualizada para que conste a informação de que wt foi compensada.

Pseudocódigo 2 Programa que executa a compensação de uma WED-transition *wt* que utilizou como entrada para a sua execução o WED-state *st* e devolve um novo WED-state equivalente a *st* ou o valor "nulo". *si* é o WED-state sobre o qual a WED-compensation associada a *wt* será executada.

```
1: function COMPENSA(wt, st, si)
2:     wc ← recupera_wed_compensation_associada(wt)
3:     sf ← wc.executa_compensacao(si)          ▷ Execução do método definido pelo projetista
4:     HistoricoExecucao.registra_detalhes_execucao(wc)
5:     HistoricoExecucao.atualiza_detalhes_execucao(wt)      ▷ Marca a execução de wt como
   compensada.
6:     if Equivalente(st, sf) then
7:         return sf
8:     else
9:         return nulo
10:    end if
11: end function
```

Com o mecanismo de compensação definido, podemos apresentar o encadeamento de compensações propriamente dito. O Pseudocódigo 3 resume a implementação desse método e seus parâmetros são os seguintes:

- *ws_inicial* é o WED-state cuja WED-transition geradora será a primeira a ser compensada;
- *condicoes_de_parada* é uma estrutura que contém uma ou mais das seguintes possibilidades: (i) número máximo de compensações, (ii) um WED-state para a condição de que o método deve parar caso algum WED-state equivalente a ele for gerado, (iii) um método que ao ser executado retorna "Verdadeiro" ou "Falso";
- *ws_atual* é o WED-state a partir do qual a primeira WED-compensation do encadeamento gerará um novo WED-state. Esse recurso é necessário porque nem sempre um WED-state é gerado por uma WED-transition, sendo possível que ele seja resultado da execução de uma WED-compensation ou de um WED-S^{-1}. Nesse caso, WED-S^{-1}s e WED-compensations não têm compensações associadas e consequentemente interromperiam a execução do encadeamento de compensações. Para contornar essa situação, é passado como parâmetro o WED-state *ws_atual*, que foi gerado por uma WED-transition e que é equivalente ao WED-state gerado por uma determinada WED-compensation ou WED-S^{-1}.

A linha 3, um dos pontos principais desse código, descreve o acesso ao histórico de execução da instância interrompida para se recuperar todas as WED-transitions que foram executadas com sucesso, de trás para frente, a partir da WED-transition que gerou o WED-state *ws_inicial*. Com essas WED-transitions em mãos, é possível iniciar a execução das compensações. A ideia geral do algoritmo é, para cada execução de WED-transition $wt \in wed\_transitions$, caso as condições de parada não sejam satisfeitas, recuperar do histórico de execução o WED-state sobre o qual *wt* escreveu e compensar *wt*. Isso é repetido até que uma condição de parada seja satisfeita ou até que a primeira WED-transition da instância seja compensada. Lembramos que o WED-state atual da interrupção é atualizado a cada execução de WED-compensation.

É importante ressaltar que o WED-state gerado pelo encadeamento de compensações é sempre consistente uma vez que ele é sempre equivalente a algum WED-state da execução. No entanto, como discutimos anteriormente (Seção 4.3.5), para reiniciar a execução da instância, é necessário executar um oferecimento adaptativo do WED-state gerado (sf), pois podem existir WED-transitions que estavam sendo executadas de maneira paralela e que foram abortadas pela interrupção.

Pseudocódigo 3 Programa que, dado um WED-state inicial consistente *ws_inicial*, uma estrutura de condições de parada *condicoes_de_parada* e um WED-state atual *ws_atual* retorna um WED-state consistente.

```
1:  function ENCADEAMENTO_DE_COMPENSAÇÕES(ws_inicial, condicoes_de_parada, ws_atual)
2:
3:      wed_transitions ← HistoricoExecucao.wed_transitions_a_partir_de(ws_inicial)
4:
5:      if ws_atual ≠ nulo then
6:          si ← ws_atual
7:      else
8:          si ← ws_inicial
9:      end if
10:
11:     for wt in wed_transitions do
12:         if ¬(satisfaz(condicoes_de_parada, si)) then
13:             st ← HistoricoExecucao.recupera_ws_inicial_da_exec_de(wt)
14:             sc ← Compensa(wt, st, si)
15:             if sc ≠ nulo then
16:                 si ← sc
17:             else
18:                 return si
19:             end if
20:         else
21:             return si
22:         end if
23:     end for
24:
25:     return si
26: end function
```

5.4.3 Oferecimento Adaptativo

O oferecimento adaptativo (ver Seção 4.3.4) é um mecanismo de recuperação essencial para garantir o reinício da execução de uma instância de forma consistente. De maneira sucinta, o objetivo do oferecimento adaptativo é reiniciar a execução de uma instância interrompida por meio do oferecimento de um WED-state aos WED-triggers do WED-flow. Quando o mecanismo detecta, através do histórico de execução da instância, que houve aborto da execução de WED-transitions durante a interrupção da instância, então, adaptações são feitas para que haja a possibilidade de se disparar novamente essas WED-transitions abortadas.

Pseudocódigo 4

```
function OFERECIMENTO_ADAPTATIVO(ws)

    if ¬Consistente(ws) then
        return nulo
    end if

    t ← HistoricoExecucao.transicao_geradora(ws)
    if é_WEDS^{-1}(t) ∨ é_WED-compensation(t) then
        ws ← HistoricoExecucao.encontra_wed_state_equivalente(ws)
    end if

    Oferecimento_Adaptativo_AUX(ws, ws)          ▷ Ver Pseudocódigo 5

    return HistoricoExecucao.wed_state_atual_interrupcao()
end function
```

Pseudocódigo 5

```
1:  function OFERECIMENTO_ADAPTATIVO_AUX(ws_inicial, ws_atual)
2:      if HistoricoExecucao.é_primeiro_wed_state_da_instancia(ws_inicial) then
3:          return
4:      end if
5:
6:      t ← HistoricoExecucao.transicao_geradora(ws_inicial)
7:      execucoes_paralelas ← HistoricoExecucao.execucoes_paralelas_problematicas(t)
8:      if execucoes_paralelas ≠ nulo then
9:          ws_alvo ← HistoricoExecucao.disparador_mais_antigo_de(execucoes_paralelas)
10:     end if
11:
12:     if ws_alvo ≠ nulo then
13:         ws_equivalente_ao_alvo ← Encadeamento_de_compensacoes(
                ws_inicial,
                {equivalente_a : ws_alvo},
                ws_atual)
14:
15:         if ws_equivalente_ao_alvo ≠ nulo then
16:             Oferecimento_Adaptativo_AUX(ws_alvo, ws_equivalente_ao_alvo)
17:         end if
18:     end if
19:
20: end function
```

Os Pseudocódigos 4 e 5 ilustram a implementação do mecanismo de oferecimento adaptativo no GEREC. A ideia básica desse algoritmo é procurar por WED-transitions que foram abortadas

quando eram executadas paralelamente e compensá-las de forma a permitir que no reinício da execução da instância haja a possibilidade de que elas sejam reexecutadas. Para explicarmos o pseudocódigo, vamos dividí-lo em três partes:

(1) Identificação de execuções paralelas;

(2) Compensação de WED-transitions;

(3) Recursão.

Na primeira parte do algoritmo, no Pseudocódigo 4, a primeira ação é verificar se o WED-state ws é consistente. Caso ele seja, o histórico de execução é consultado para recuperar o tipo de WED-transition que gerou ws. Se for um WED-S^{-1} ou uma WED-compensation, devido a possibilidade de se efetuar encadeamentos de compensações, são necessários alguns cuidados especiais já que esses tipos de WED-transitions não tem compensações associadas. É necessário, então, descobrir qual foi o WED-state equivalente a ws, resultado da execução de um WED-S^{-1} ou WED-compensation, e gerado por uma WED-transition. Esse WED-state sobrescreverá ws.

O método Oferecimento_Adaptativo_AUX(ws,ws) (Pseudocódigo 5) é executado. Nele, logo de início, lançamos mão, novamente, do histórico de execução da instância para descobrir se $ws\_inicial$ é o primeiro WED-state da instância. Caso isso ocorra, mesmo que haja execuções de WED-transitions abortadas, elas certamente poderão ser executadas novamente após $ws\_inicial$ ser oferecido para todos os WED-triggers. Assim, o método Oferecimento_Adaptativo_AUX devolve o controle para o método Oferecimento_Adaptativo, o qual, por sua vez, retorna o WED-state atual da interrupção.

Caso $ws\_inicial$ não seja o primeiro WED-state da instância, é recuperada do histórico de execução a WED-transition t que gerou $ws\_inicial$. Em seguida, é verificado se t foi executada paralelamente com outras WED-transitions que tiveram algum tipo de problema. Problema, nesse contexto, significa o aborto, compensação ou pulo de uma WED-transition. A identificação da execução paralela é um dos passos mais importantes dessa primeira parte e consideramos benéfico explicar esse processo em mais detalhes.

Para descobrirmos se uma WED-transition foi executada paralelamente a uma outra, é necessário observar os atributos $created\_at$ e $completed\_at$ presentes em todas as entradas do histórico de execução. Esses dois atributos guardam o momento em que uma WED-transition começou a ser executada e o momento em que ela terminou (com ou sem sucesso) respectivamente. A Figura 5.7 ilustra as possíveis formas em que execuções paralelas podem ocorrer (relativas a WED-transition wt).

Uma consulta ao histórico de execução recupera todas as WED-transitions executadas paralelamente à WED-transition t, que gerou $ws\_inicial$, caso alguma delas tenha sido abortada, compensada ou pulada, e devolve o WED-state (ws_alvo) mais antigo que disparou alguma delas. Se nenhuma WED-transition tiver sido executada em paralelo, ou se nenhuma delas tiver sido abortada, compensada ou pulada, então nenhuma "adaptação" é necessária e Oferecimento_Adaptativo_AUX devolve o controle para o método Oferecimento_Adaptativo o qual retornará o WED-state atual da interrupção.

No caso em que o ws_alvo é não nulo, passamos à segunda parte do algoritmo. Nela é executado um encadeamento de compensações, a partir de $ws\_inicial$, tendo como condição de parada

a geração de um WED-state equivalente a *ws_alvo* e com *ws_atual* como o WED-state que será usado de ponto de partida para a execução da primeira WED-compensation. O objetivo disso é permitir que o WED-state gerado pelo encadeamento de compensações permita que WED-transitions abortadas possam ser executadas novamente.

A terceira parte do algoritmo refere-se ao caso em que o encadeamento de compensações termina com sucesso e retorna um WED-state consistente *ws_equivalente_ao_alvo*. Para garantir que todas as WED-transitions terão a chance de ser reexecutadas, é necessário verificar se o encadeamento de compensações não compensou parte de execuções que foram executadas em paralelo. Isso é feito através da chamada recursiva *Oferecimento_Adaptativo_AUX(ws_alvo,ws_equivalente_ao_alvo)* que repete todo o processo discutido nesta seção até que não haja violações a WED-transitions executadas em paralelo. Quando isso ocorrer, o WED-state atual da interrupção, devidamente atualizado pelas execuções de WED-compensations, é retornado pelo método Oferecimento_Adaptativo (Pseudocódigo 4) e oferecido a todos os WED-triggers. O oferecimento adaptativo é explicado em detalhes sem levar em consideração detalhes da implementação na Seção 4.3.4.

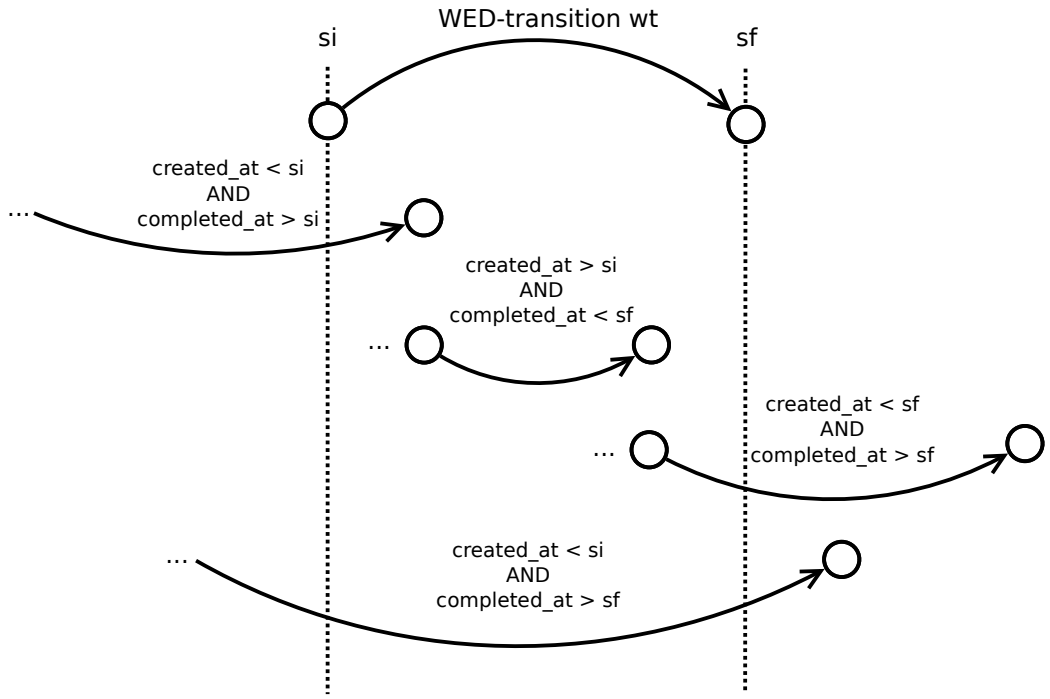


Figura 5.7: *Análise das formas em que a execução paralela pode acontecer em relação a execução de uma WED-transition wt.*

5.4.4 WED-S^{-1}

O WED-S^{-1} (Seção 4.3.7) é um mecanismo de pulo *backward* usado em situações em que o WED-state atual da interrupção é inconsistente. Em linhas gerais, sua função é, por meio de um tipo especial de WED-transition, gerar um WED-state equivalente a algum outro presente no histórico de execuções da instância interrompida.

O Pseudocódigo 6 ilustra a implementação do WED-S^{-1}. Ele tem como parâmetros o método $wedS^{-1}$ definido em *Ruby* pelo administrador de sistema e um WED-state si que servirá de base para a geração de um novo WED-state sf. Por meio da execução de $wedS^{-1}$, o método $HistoricoExecucao.recupera\_wed\_state\_equivalente\_a(sf)$ é executado. Sua função é procurar, no histórico de execução da instância, por algum WED-state gerado por uma WED-transition que não tenha sido compensada ou "pulada" e que, além disso, seja equivalente ao WED-state sf passado como argumento ao método. Caso esse WED-state exista, o WED-state atual da interrupção é atualizado, o histórico de execução das WED-transitions que foram "puladas" é atualizado, os detalhes da execução de $wedS^{-1}$ são registrados e o WED-state sf é retornado. Caso não haja WED-state equivalente a sf, o valor "nulo" é retornado, indicando que o método $wedS^{-1}$ não foi bem modelado.

Pseudocódigo 6 Programa que executa um WED-S^{-1}. Ele recebe como entrada um WED-S^{-1} definido pelo administrador de sistema e um WED-state si, que servirá de base para o WED-state gerado pelo WED-S^{-1}.

```
1:  function WED-S^-1(wedS^-1, si)
2:      if Consistente(si) then
3:          return nulo
4:      end if
5:      sf ← wedS^-1.executa(si)                ▷ Executa método definido pelo administrador.
6:      equivalente ← HistoricoExecucao.recupera_wed_state_equivalente_a(sf)
7:      if equivalente ≠ nulo then
8:          atualiza_wed_state_atual_interrupcao(sf)
9:          HistoricoExecucao.atualiza_detalhes_execucao(si, sf)
10:         HistoricoExecucao.registra_detalhes_execucao(wedS^-1)
11:         return sf
12:     else
13:         return nulo
14:     end if
15: end function
```

Após a execução com sucesso de um WED-S^{-1}, o WED-state atual da interrupção será consistente e o administrador de sistema terá que optar por executar um outro mecanismo de recuperação a fim de reiniciar a execução da instância. As opções são a execução de um oferecimento adaptativo do novo WED-state gerado, a execução de um encadeamento de compensações, ou a execução de um WED-S^{+a}.

5.4.5 WED-S^{+a}

O WED-S^{+a} (Seção 4.3.7) é um mecanismo de pulo *forward* que se diferencia do WED-S^{-1} por permitir sua execução a partir de WED-states consistentes e inconsistentes, e por gerar WED-states não equivalentes aos presentes no histórico de execuções da instância interrompida.

O Pseudocódigo 7, que ilustra a implementação de um WED-S^{+a}, é quase idêntico ao Pseudocódigo 6, diferenciado-se desse último pela condição na linha 4 que é satisfeita caso não haja WED-states equivalentes no histórico e pela retirada do comando que atualizava as WED-transitions do histórico que foram "puladas".

Vale a pena lembrar que ao fim da execução com sucesso de um WED-S^{+a}, o WED-state gerado por ele deve ser oferecido a todos os WED-triggers do WED-flow para que a instância possa recomeçar.

Pseudocódigo 7 Programa que executa um WED-S^{+a}. Ele recebe como entrada um WED-S^{+a} definido pelo administrador de sistema e um WED-state si, que servirá de base para o WED-state gerado pelo WED-S^{+a}.

```
1:  function WED-S^{+a}(wedS^{+a}, si)
2:      sf ← wedS^{+a}.executa(si)              ▷ Executa método definido pelo administrador.
3:      equivalente ← HistoricoExecucao.recupera_wed_state_equivalente_a(sf)
4:      if equivalente = nulo then
5:          atualiza_wed_state_atual_interrupcao(sf)
6:          HistoricoExecucao.registra_detalhes_execucao(wedS^{+a})
7:          return sf
8:      else
9:          return nulo
10:     end if
11: end function
```

5.5 Sumário

Neste capítulo discutimos alguns detalhes da implementação do Gerenciador de Recuperação (GEREC) da ferramenta WED-tool, que é um protótipo de um sistema de gestão de processos de negócio que implementa a abordagem WED-flow.

A implementação dos métodos de tratamento de exceções disponíveis no GEREC é um bom indicativo de viabilidade de nossa proposta, o que consideramos uma importante contribuição para a área de estudos de processos de negócio. Apesar da inquestionável importância dos conceitos teóricos apresentados em outros trabalhos acadêmicos, poucos disponibilizam algum tipo de implementação como parte de suas contribuições.

O GEREC, além de satisfazer as necessidades de tratamento de exceções da versão atual da ferramenta WED-tool, também foi construído de forma a incorporar facilmente novos métodos de tratamento de exceções e se adaptar a novas necessidades da ferramenta.

Capítulo 6

Conclusão

Neste trabalho, aprimoramos e implementamos o modelo de tratamento de exceções da abordagem de modelagem e gerenciamento de processos de negócio WED-flow. A justificativa para isso foi a necessidade de se solucionar uma série de deficiências relacionadas a sistemas de gestão de processos de negócio e suas formas de tratamento de exceções. Neste trabalho, abordamos, em especial, 4 dessas deficiências:

- **Modelagem evolutiva:** Processos de negócio evoluem, ou seja, mudam ao longo do tempo, logo, modelos de processos de negócio também deveriam evoluir, entretanto, é muito comum que a alteração de um modelo seja uma tarefa burocrática e que só possa ser efetuada durante a fase de modelagem dos processos, sendo muitas vezes proibida nas fases de instanciação de processos e de execução de instâncias. Nesses casos, para que os projetistas possam intervir para alterar o modelo, é necessário que todas as execuções tenham sido concluídas. Em muitos casos, aguardar a execução de todas as instâncias pode ser uma situação inviável.

- **Expressividade da linguagem de modelagem:** Existem linguagens de modelagem de processos de negócio que, por vezes, dificultam a concepção de modelos devido a limitações relativas a definições de fluxo (condições, laços, etc.) e de atividades. Apesar de haver claramente uma limitação associada ao tipo de linguagem de modelagem utilizado, é possível, com o objetivo de aumentar a capacidade de representação de uma linguagem, usar características de linguagens baseadas em grafos em linguagens baseadas em regras e vice-versa. Essa mistura de tipos de linguagem de modelagem também permite assegurar mecanismos de flexibilidade **e** de apoio ao sistema de gestão de processos de negócio ou de gerenciamente de workflows.

- **Correção e consistência transacional:** A utilização de Modelos Transacionais Avançados para embasar sistemas de gestão de processos de negócio não é uma abordagem universal e, de fato, existem vários sistemas que não dispõem de tratamento de exceções ou dispõem de um tratamento de exceções muito simplificado. Assim, corre-se o risco de perder a garantia de propriedades importantes como correção e consistência transacional.

- **Tratamento de exceções não esperadas:** Exceções não esperadas são aquelas cujas formas de detecção não são descritas no modelo de processos de negócios. A maioria dos sistemas de gestão não é capaz de detectar esse tipo de exceção, deixando-os vulneráveis às consequências imprevisíveis da ocorrência de exceções não esperadas.

Conforme discutido no Capítulo 3, essas deficiências têm a ver com a relação paradoxal entre os conceitos de flexibilidade e apoio. As deficiências que levantamos, especificamente, são peculiares de sistemas de gestão de processos de negócio que têm poucos mecanismos que assegurem flexibilidade. A utilização de linguagens de modelagem baseadas em grafos pela maioria dos sistemas de gestão de processos de negócio e pela maioria dos sistemas de gerenciamento de workflow explicam esse fenômeno. Linguagens de modelagem baseadas em grafos permitem a criação de mecanismos de apoio de forma mais direta, porém, não têm condições ideais para o desenvolvimento de mecanismos que garantam flexibilidade. Entretanto, insistimos que uma solução que resolva deficiências de flexibilidade e não ofereça soluções de apoio não é satistória, pois, como discutimos anteriormente, ambos os conceitos são importantes. É necessário encontrar um **meio-termo**.

A abordagem de tratamento de exceções que propomos neste trabalho busca o equilíbrio entre flexibilidade e apoio na medida do possível, de acordo com as limitações de linguagens de modelagem baseadas em regras. Herdamos da abordagem WED-flow a expressividade que nos permite a definição de condições lógicas e atividades complexas assim como permite o reaproveitamento de estruturas do modelo, como WED-transitions, WED-conditions, WED-S^{-1}, WED-S^{+a} e WED-compensations, facilitando assim sua manutenção e evolução. Além disso, a extensão da possibilidade de alteração de modelos de WED-flow também à fase de tratamento de exceções garante mais robustez e adaptabilidade ao sistema.

O tratamento de exceções inesperadas é feito por meio da intervenção *ad hoc* de um administrador de sistema. Ele tem à sua disposição uma série de métodos de recuperação baseados em Modelos Transacionais Avançados que proporcionam maior controle e precisão à resolução de situações excepcionais. Esses métodos podem afetar tanto o modelo de um processo de negócio, por meio da alteração da definição de um WED-flow, quanto somente a instância problemática. Com os métodos de recuperação, mais explicitamente, o oferecimento adaptativo, as compensações encadeadas, o WED-S^{-1} e o WED-S^{+a}, o administrador pode escolher diferentes graus de flexibilidade dependendo da necessidade de cada situação. Quanto ao apoio oferecido, além de ser garantido que a execução de instâncias respeita as definições do modelo, há estudos em andamento sobre formas de também se apoiar a modelagem, por meio, principalmente, da verificação prévia da correção do modelo de um WED-flow.

O desenvolvimento do gerenciador de recuperação da ferramenta WED-tool, que é um protótipo de sistema de gestão de processos de negócio que implementa a abordagem WED-flow, nos mostrou que é possível aplicar na prática os conceitos propostos neste trabalho. Apesar de ainda estar em estágios preliminares, consideramos essa indicação de viabilidade uma contribuição importante dada a quantidade de trabalhos acadêmicos que, apesar de importantes contribuições conceituais, não apresentam protótipos ou não os tornam públicos.

Acreditamos que com as garantias transacionais adquiridas de Modelos Transacionais Avançados, com uma expressiva linguagem de modelagem baseada em regras, que fundamenta um processo de modelagem evolutivo, com um tratamento de exceções capaz de tratar exceções esperadas e inesperadas, e com o auxílio de métodos de recuperação predefinidos ou definidos de forma *ad hoc*, temos um modelo bastante flexível e poderoso.

6.1 Contribuições

As contribuições deste trabalho foram as seguintes:

- **Aprimoramento do modelo de tratamento de exceções da abordagem WED-flow:** Além de revisarmos os conceitos previamente estabelecidos em trabalhos anteriores, sistematizamos o processo de tratamento de exceções, embasamos conceitualmente os métodos de tratamento de exceções que já existiam e propusemos novos. Essas contribuições foram objeto de estudo do artigo resumido "Uma abordagem *ad hoc* para o tratamento de exceções em processos transacionais" [SBF12], publicado nos anais do 27º Simpósio Brasileiro de Bancos de Dados e apresentado no mesmo evento.

- **Implementação do modelo de tratamento de exceções da abordagem WED-flow:** O módulo de gerenciamento de recuperação da ferramenta WED-tool foi desenvolvido de forma que vários dos métodos de tratamento de exceção propostos neste trabalho fossem implementados. Além dos métodos, fizemos também todos os ajustes necessários na ferramenta para que o processo de tratamento de exceções fosse realizado da forma mais semelhante possível ao proposto teoricamente. A ferramenta WED-tool foi o tema do artigo "WED-tool: uma ferramenta para o controle de execução de processos de negócio transacionais" [GSBF12] publicado e apresentado na Seção de Demos do 27º Simpósio Brasileiro de Bancos de Dados.

6.2 Trabalhos futuros

Como neste trabalho o tratamento de exceções está intimamente relacionado com a abordagem WED-flow, que é responsável pela modelagem e controle de execução de instâncias de processos de negócio, é possível pensar em trabalhos futuros em duas frentes: relativos ao tratamento de exceções, de fato, e relativos a melhorias na abordagem WED-flow (que acabam refletindo no tratamento de exceções).

- **Desenvolvimento de mecanismos de apoio:** A abordagem WED-flow utiliza uma linguagem de modelagem baseada em regras para definir modelos de processos de negócio. Na versão atual dessa abordagem, apesar de haver mecanismos de apoio à execução, faltam mecanismos de apoio à modelagem. Dentre eles, faltam, em especial, mecanismos de verificação prévia de modelos. Embora, como discutimos ao longo do texto, a definição de mecanismos de apoio, de forma geral, seja mais difícil de ser feita em linguagens baseadas em regras, no caso específico de mecanismos de validação prévia, existem alguns trabalhos que apontam soluções interessantes ([vdAPS09, JLM+12]) para essa questão.

- **Integração com serviços *Web*:** Na ferramenta WED-tool, a única restrição imposta ao código *Ruby* que descreve uma WED-transition é quanto ao seu valor de retorno, que deve ser um dicionário com os valores dos WED-attributes previamente definidos no modelo. É possível, por exemplo, que o código de uma WED-transition especifique a realização de chamadas a serviços *Web* e utilize suas respostas para definir alterações sobre WED-attributes. Entretanto é importante notar que um controle de execução que permita, de forma nativa, a execução de atividades como serviços *Web* é algo muito mais complexo e que precisa ter

seus detalhes melhor especificados. Apesar de já haver trabalhos nesse sentido [Rod12], há, atualmente, pesquisadores do grupo DATA-IME [DIa] estudando esse problema. O modelo de tratamento de exceções nesse contexto, embora guarde várias semelhanças em sua essência, será diferente e, assim sendo, será necessário reavaliá-lo.

- **Aprimoramento da ferramenta WED-tool:** É interessante que a ferramenta WED-tool, atualmente em versão de protótipo, evolua rapidamente para uma versão mais estável e abrangente para que assim testes mais complexos e mais próximos da realidade possam ser realizados. Como exemplos de melhorias, podemos citar a criação de mecanismos de apoio à modelagem, como algum tipo de ferramenta visual que facilite o desenvolvimento de modelos WED-flow; o desenvolvimento de uma interface mais amigável para o *menu* de controle; o desenvolvimento de testes automatizados tanto para o Núcleo de Controle quanto para o Gerenciador de Recuperação; desenvolvimento de um módulo que explicite a interação entre aplicações e a ferramenta WED-tool; etc.

- **Comparação qualitativa entre WED-tool e outras ferramentas:** Uma vez que a ferramenta WED-tool alcance um estágio mais avançado de desenvolvimento, seria interessante estabelecer critérios de comparação e utilizá-los para situar melhor nossas propostas de tratamento de exceções entre as principais ferramentas acadêmicas e comerciais. Em nossa busca por trabalhos relacionados, não encontramos nenhum que se voltasse especificamente para a comparação de abordagens de tratamento de exceções. Quanto aos trabalhos que encontramos sobre comparação de sistemas de gestão de processos de negócio [LS07, GJCV09, EVWH06], o tratamento de exceções foi analisado de maneira superficial e secundária.

- **Desenvolvimento de metodologia de modelagem:** Para facilitar o processo de modelagem de WED-flows, seria interessante o desenvolvimento de uma metodologia que auxiliasse especialistas na identificação de eventos e dados relevantes ao processo de negócio, e no projeto do modelo WED-flow correspondente.

Apêndice A

Exemplos

A.1 Modelo inicial de um WED-flow

Nesta Seção apresentamos um exemplo de modelo de WED-flow inicial baseado no cenário de aluguel de carros (ver Seção 2.4). Listamos a parte do modelo descrita em XML e também as classes *Ruby* que descrevem as WED-transitions e as WED-compensations.

A.1.1 XML

```
<WED-flow-initial-schema>
  <WED-attributes>
    <Attribute Name="id_usuario" Type="string" />
    <Attribute Name="documentos_usuario" Type="string" />
    <Attribute Name="periodo_reserva" Type="integer" />
    <Attribute Name="categoria_carro" Type="string" />
    <Attribute Name="modelo_carro" Type="string" />
    <Attribute Name="placa_carro" Type="string" />
    <Attribute Name="status_reserva" Type="string" />
    <Attribute Name="status_carro" Type="string" />
    <Attribute Name="avarias" Type="string" />
    <Attribute Name="multas" Type="float" />
    <Attribute Name="total_a_pagar" Type="float" />
  </WED-attributes>

  <WED-conditions>
    <Condition Name="c_reserva_requisitada" >
      <Predicate Id="1"> status_reserva = 'requisitada' </Predicate>
      <Predicate Id="2"> id_usuario = null </Predicate>
      <Expression> 1 AND 2 </Expression>
    </Condition>

    <Condition Name="c_condicao_para_inicializar_reserva" >
      <Predicate Id="1"> status_reserva = 'iniciada' </Predicate>
      <Predicate Id="2"> id_usuario != null </Predicate>
      <Predicate Id="3"> modelo_carro = null </Predicate>
      <Predicate Id="4"> categoria_carro = null </Predicate>
      <Predicate Id="5"> periodo_reserva = null </Predicate>
      <Predicate Id="6"> documentos_usuario = null </Predicate>
      <Expression> 1 AND 2 AND 3 AND 4 AND 5 AND 6 </Expression>
```

```
    </Condition>

    <Condition Name="c_dados_reserva_coletados" >
      <Predicate Id="1"> modelo_carro != null </Predicate>
      <Predicate Id="2"> categoria_carro != null </Predicate>
      <Predicate Id="3"> periodo_reserva != null </Predicate>
      <Predicate Id="4"> documentos_usuario != null </Predicate>
      <Predicate Id="5"> status_reserva = 'iniciada' </Predicate>
      <Expression> 1 AND 2 AND 3 AND 4 AND 5</Expression>
    </Condition>

    <Condition Name="c_reserva_confirmada" >
      <Predicate Id="1"> status_reserva = 'confirmada' </Predicate>
      <Predicate Id="2"> status_carro = null </Predicate>
      <Expression> 1 AND 2 </Expression>
    </Condition>

    <Condition Name="c_carro_retirado" >
      <Predicate Id="1"> status_carro = 'retirado' </Predicate>
      <Predicate Id="2"> avarias = null </Predicate>
      <Predicate Id="3"> multas = null </Predicate>
      <Expression> 1 AND 2 AND 3</Expression>
    </Condition>

    <Condition Name="c_carro_avariado" >
      <Predicate Id="1"> status_carro = 'devolvido' </Predicate>
      <Predicate Id="2"> avarias != null </Predicate>
      <Predicate Id="3"> avarias != 'avaria imprevista' </Predicate>
      <Predicate Id="4"> multas = null </Predicate>
      <Expression> 1 AND 2 AND 3 AND 4</Expression>
    </Condition>

    <Condition Name="c_carro_vistoriado" >
      <Predicate Id="1"> status_carro = 'devolvido' </Predicate>
      <Predicate Id="2"> avarias = null </Predicate>
      <Predicate Id="3"> multas != null </Predicate>
      <Expression> (1 AND 3) OR (1 AND 2 AND 3)  </Expression>
    </Condition>

    <Condition Name="c_condicao_final" >
      <Predicate Id="1"> status_reserva = 'rejeitada' </Predicate>
      <Predicate Id="2"> status_reserva = 'paga' </Predicate>
      <Expression> 1 OR 2 </Expression>
    </Condition>
  </WED-conditions>

  <WED-transitions>
    <Transition Name="t_inicializar_reserva" >
      <UpdatedAttribute AttrName="id_usuario" />
      <UpdatedAttribute AttrName="documentos_usuario" />
      <UpdatedAttribute AttrName="periodo_reserva" />
      <UpdatedAttribute AttrName="categoria_carro" />
      <UpdatedAttribute AttrName="modelo_carro" />
```

```
    <UpdatedAttribute AttrName="placa_carro" />
    <UpdatedAttribute AttrName="status_carro" />
    <UpdatedAttribute AttrName="avarias" />
    <UpdatedAttribute AttrName="multas" />
    <UpdatedAttribute AttrName="total_a_pagar" />
  </Transition>

  <Transition Name="t_escolher_carro" >
    <UpdatedAttribute AttrName="modelo_carro" />
    <UpdatedAttribute AttrName="categoria_carro" />
    <UpdatedAttribute AttrName="periodo_reserva" />
  </Transition>

  <Transition Name="t_receber_documentos" >
    <UpdatedAttribute AttrName="documentos_usuario" />
  </Transition>

  <Transition Name="t_encaminhar_para_validacao_do_gerente" >
    <UpdatedAttribute AttrName="status_reserva" />
  </Transition>

  <Transition Name="t_registrar_retirada_carro" >
    <UpdatedAttribute AttrName="status_carro" />
          <UpdatedAttribute AttrName="placa_carro" />
  </Transition>

  <Transition Name="t_registrar_devolucao_e_vistoriar_carro" >
    <UpdatedAttribute AttrName="status_carro" />
    <UpdatedAttribute AttrName="avarias" />
  </Transition>

  <Transition Name="t_computar_multas" >
    <UpdatedAttribute AttrName="multas" />
  </Transition>

  <Transition Name="t_receber_pagamento" >
    <UpdatedAttribute AttrName="status_reserva" />
    <UpdatedAttribute AttrName="total_a_pagar" />
  </Transition>
</WED-transitions>

<WED-compensations>
  <Compensation Name="c_registrar_retirada_carro" ForWedTransition="
      t_registrar_retirada_carro">
    <UpdatedAttribute AttrName="status_carro" />
    <UpdatedAttribute AttrName="placa_carro" />
  </Compensation>

  <Compensation Name="c_registrar_devolucao_e_vistoriar_carro"
      ForWedTransition="t_registrar_devolucao_e_vistoriar_carro" >
    <UpdatedAttribute AttrName="status_carro" />
    <UpdatedAttribute AttrName="avarias" />
  </Compensation>
```

```
        <Compensation Name="c_computar_multas" ForWedTransition="
            t_computar_multas">
      <UpdatedAttribute AttrName="multas" />
    </Compensation>

    <Compensation Name="c_encaminhar_para_validacao_do_gerente" ForWedTransition
        ="t_encaminhar_para_validacao_do_gerente">
      <UpdatedAttribute AttrName="status_reserva" />
    </Compensation>

        <Compensation Name="c_receber_documentos" ForWedTransition="
            t_receber_documentos">
      <UpdatedAttribute AttrName="documentos_usuario" />
    </Compensation>

        <Compensation Name="c_escolher_carro" ForWedTransition="t_escolher_carro
            ">
      <UpdatedAttribute AttrName="modelo_carro" />
      <UpdatedAttribute AttrName="categoria_carro" />
      <UpdatedAttribute AttrName="periodo_reserva" />
    </Compensation>

        <Compensation Name="c_inicializar_reserva" ForWedTransition="
            t_inicializar_reserva">
      <UpdatedAttribute AttrName="id_usuario" />
      <UpdatedAttribute AttrName="documentos_usuario" />
      <UpdatedAttribute AttrName="periodo_reserva" />
      <UpdatedAttribute AttrName="categoria_carro" />
      <UpdatedAttribute AttrName="modelo_carro" />
      <UpdatedAttribute AttrName="placa_carro" />
      <UpdatedAttribute AttrName="status_carro" />
      <UpdatedAttribute AttrName="avarias" />
      <UpdatedAttribute AttrName="multas" />
      <UpdatedAttribute AttrName="total_a_pagar" />
    </Compensation>
  </WED-compensations>

  <WED-flows>
    <Flow Name="aluguel_de_carros" FinalStateCondName="c_condicao_final">
      <Trigger CondName="c_reserva_requisitada"
        TransName="t_inicializar_reserva" Period="3s" />

      <Trigger CondName="c_condicao_para_inicializar_reserva"
        TransName="t_escolher_carro" Period="5s" />

      <Trigger CondName="c_condicao_para_inicializar_reserva"
        TransName="t_receber_documentos" Period="5s" />

      <Trigger CondName="c_dados_reserva_coletados"
        TransName="t_encaminhar_para_validacao_do_gerente"  Period="7s" />

      <Trigger CondName="c_reserva_confirmada"
```

```
        TransName="t_registrar_retirada_carro"  Period="6s" />

      <Trigger CondName="c_carro_retirado"
        TransName="t_registrar_devolucao_e_vistoriar_carro"  Period="4s" />

      <Trigger CondName="c_carro_avariado"
        TransName="t_computar_multas"  Period="3s" />

      <Trigger CondName="c_carro_vistoriado"
        TransName="t_receber_pagamento"  Period="5s" />
    </Flow>
  </WED-flows>

  <AWICs>
    <Constraint CondName="c_condicao_final" />
  </AWICs>

</WED-flow-initial-schema>
```

Trecho A.1: *XML de modelo inicial de WED-flow baseado no cenário de aluguel de carros*

A.1.2 Classes em Ruby

WED-transitions

```
#Arquivo TComputarMultas.rb
class TComputarMultas
  def self.run(initialState)
    return {:multas => 10.0}
  end
end

#Arquivo TEncaminharParaValidacaoDoGerente.rb
class TEncaminharParaValidacaoDoGerente
  def self.run(initialState)

    #quando o identificador da instância é múltiplo de 7
    #o status_reserva é definido como 'rejeitada'.

    if initialState.wed_flow_instance_id % 7 == 0
      return {:status_reserva => 'rejeitada'}
    else
      return {:status_reserva => 'confirmada'}
    end
  end
end

#Arquivo TEscolherCarro.rb
class TEscolherCarro
  def self.run(initialState)
    return {:modelo_carro => 'Ford Fiesta', :categoria_carro => 'AA', :
        periodo_reserva => 4}
```

```
  end
end

#Arquivo TInicializarReserva.rb
class TInicializarReserva
  def self.run(initialState)
    return {:id_usuario => (initialState.wed_flow_instance_id * 10),
      :status_reserva => 'iniciada',
      :documentos_usuario => nil,
      :periodo_reserva => nil,
      :categoria_carro => nil,
      :modelo_carro => nil,
      :placa_carro => nil,
      :status_carro => nil,
      :avarias => nil,
      :multas => nil}
  end
end

#Arquivo TReceberDocumentos.rb
class TReceberDocumentos
  def self.run(initialState)
    return {:documentos_usuario => '/home/user/docs/rg.png'}
  end
end

#Arquivo TReceberPagamento.rb
class TReceberPagamento
  def self.run(initialState)
        multas = initialState.multas || 0
        total = (20.0*initialState.periodo_reserva) + multas
    return {:status_reserva => 'paga', :total_a_pagar => total}
  end
end

#Arquivo TRegistrarDevolucaoEVistoriarCarro.rb
class TRegistrarDevolucaoEVistoriarCarro
  def self.run(initialState)
    #Simulamos algumas avarias que dependem do identificador da instância

    return {:status_carro => 'devolvido', :avarias => 'avaria imprevista'} if
        initialState.wed_flow_instance_id == 1
    return {:status_carro => 'devolvido', :avarias => 'sem calota da roda
        dianteira direita'} if initialState.wed_flow_instance_id == 3
    return {:status_carro => 'devolvido'}
  end
end

#Arquivo TRegistrarRetiradaCarro.rb
class TRegistrarRetiradaCarro
  def self.run(initialState)
    return {:status_carro => 'retirado', :placa_carro => 'ABC-0123'}
  end
```

```
end
```

Trecho A.2: *Classes Ruby de WED-transition pertencentes ao modelo inicial de WED-flow baseado no cenário de aluguel de carros*

WED-compensations

```ruby
#Arquivo CComputarMultas.rb
class CComputarMultas
  def self.run(initial_state)
    return {:multas => nil}
  end
end

#Arquivo CEncaminharParaValidacaoDoGerente.rb
class CEncaminharParaValidacaoDoGerente
   def self.run(initial_state)
      return {:status_reserva => 'iniciada'}
   end
end

#Arquivo CEscolherCarro.rb
class CEscolherCarro
  def self.run(initial_state)
    return {:modelo_carro => nil, :categoria_carro => nil,
        :periodo_reserva => nil}
  end
end

#Arquivo CInicializarReserva.rb
class CInicializarReserva
  def self.run(initial_state)
    return {:id_usuario => nil,
      :status_reserva => 'requisitada',
      :documentos_usuario => nil,
      :periodo_reserva => nil,
      :categoria_carro => nil,
      :modelo_carro => nil,
      :placa_carro => nil,
      :status_carro => nil,
      :avarias => nil,
      :multas => nil}
  end
end

#Arquivo CReceberDocumentos.rb
class CReceberDocumentos
  def self.run(initial_state)
    return {:documentos_usuario => nil}
  end
end
```

```ruby
#Arquivo CRegistrarDevolucaoEVistoriarCarro.rb
class CRegistrarDevolucaoEVistoriarCarro
    def self.run(initial_state)
        return {:status_carro => nil, :avarias => nil}
    end
end

#Arquivo CRegistrarRetiradaCarro.rb
class CRegistrarRetiradaCarro
    def self.run(initial_state)
        return {:status_carro => nil, :placa_carro => nil}
    end
end
```

Trecho A.3: *Classes Ruby de WED-compensation pertencentes ao modelo inicial de WED-flow baseado no cenário de aluguel de carros*

A.2 Exemplos completos de execução de instâncias

Nesta Seção apresentamos alguns exemplos completos de execução de instâncias de WED-flow baseadas no modelo descrito na Seção A.1. Os exemplos abordam as seguintes situações:

- **Seção A.2.1:** Execução de uma instância de WED-flow que termina com sucesso sem que nenhuma exceção seja detectada ;
- **Seção A.2.2:** Execução de uma instância de WED-flow em que uma exceção é detectada e o tratamento de exceções realizado faz uso de um encadeamento de compensações seguido da execução de um oferecimento adaptativo. Após a resolução da situação excepcional, a execução termina com sucesso;
- **Seção A.2.3:** Execução de uma instância de WED-flow em que uma exceção é detectada e o tratamento de exceções realizado somente faz uso do mecanismo de oferecimento adaptativo. Após a resolução da situação excepcional, a execução termina com sucesso;
- **Seção A.2.4:** Execução de uma instância de WED-flow em que uma exceção é detectada e o tratamento de exceções realizado faz uso de um WED-S^{+a}. Após a resolução da situação excepcional, a execução termina com sucesso;
- **Seção A.2.5:** Execução de uma instância de WED-flow em que uma exceção é detectada e o tratamento de exceções realizado faz uso de um WED-S^{-1} seguido de um oferecimento adaptativo. Após a resolução da situação excepcional, a execução termina com sucesso;
- **Seção A.2.6:** Execução de uma instância de WED-flow em que uma exceção é detectada e o tratamento de exceções realizado faz uso de um WED-S^{-1} seguido da execução de um encadeamento de compensações e, finalmente, de um oferecimento adaptativo. Após a resolução da situação excepcional, a execução termina com sucesso;

Apesar dos exemplos serem baseados no modelo da Seção A.1, foi necessário em alguns momentos, especialmente para a simulação de exceções, alterá-lo levemente.

O modelo inicial assim como o código fonte da ferramenta WED-tool e todos os artigos relacionados à abordagem WED-flow estão disponíveis para *download* no sítio do Grupo DATA-IME dedicado à essa abordagem [DIb].

A.2.1 Execução sem exceções

Nos exemplos ilustrados nas Figuras A.1 e A.2, temos a execução de instâncias de WED-flow em que nenhuma exceção é detectada. No Exemplo A.1, o útimo WED-state gerado, $S7$, satisfaz a condição final do WED-flow (ver modelo XML na Seção A.1) e, por isso, a execução é finalizada com sucesso quando ele é gerado. De forma análoga, o último WED-state gerado no exemplo ilustrado na Figura A.2, $S4$, também satisfaz a condição final do WED-flow, caracterizando a conclusão com sucesso da execução da instância.

	id usuário	Status reserva	Mod. do carro	Categoria carro	Período reserva	Doc. usuário	Placa carro	Status carro	multa	avarias	total
S0	*null*	requisitada	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*
S1	**120**	**iniciada**	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*
S2	120	iniciada	*null*	*null*	*null*	**rg.png**	*null*	*null*	*null*	*null*	*null*
S3	120	iniciada	**Ford Fiesta**	**AA**	**01/02 a 20/02**	rg.png	*null*	*null*	*null*	*null*	*null*
S4	120	**confirmada**	Ford Fiesta	AA	01/02 a 20/02	rg.png	*null*	*null*	*null*	*null*	*null*
S5	120	confirmada	Ford Fiesta	AA	01/02 a 20/02	rg.png	**ABC 0123**	**retirado**	*null*	*null*	*null*
S6	120	confirmada	Ford Fiesta	AA	01/02 a 20/02	rg.png	ABC 0123	**devolvido**	*null*	*null*	*null*
S7	120	**paga**	Ford Fiesta	AA	01/02 a 20/02	rg.png	ABC 0123	devolvido	*null*	*null*	**90.0**

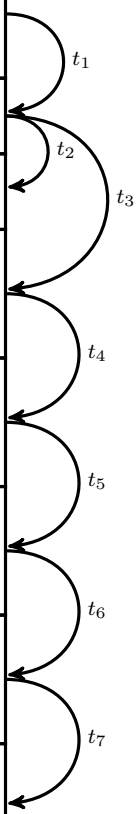


Figura A.1: *Execução de instância em que não há detecção de exceções*

	id usuário	Status reserva	Mod. do carro	Categoria carro	Período reserva	Doc. usuário	Placa carro	Status carro	multa	avarias	total
S0	*null*	requisitada	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*
S1	**120**	**iniciada**	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*
S2	120	iniciada	*null*	*null*	*null*	**rg.png**	*null*	*null*	*null*	*null*	*null*
S3	120	iniciada	**Ford Fiesta**	**AA**	**01/02 a 20/02**	rg.png	*null*	*null*	*null*	*null*	*null*
S4	120	**rejeitada**	Ford Fiesta	AA	01/02 a 20/02	rg.png	*null*	*null*	*null*	*null*	*null*

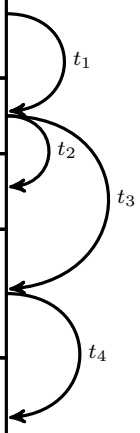


Figura A.2: *Execução de instância em que não há detecção de exceções*

A.2.2 Execução com detecção de exceção: encadeamento de compensações + oferecimento adaptativo

No exemplo ilustrado na Figura A.3, a execução da instância é interrompida devido ao aborto de uma WED-transition (t_5). Como, nesse caso, o WED-state atual da interrupção é consistente, é possível executar, a partir dele, um encadeamento de compensações. O encadeamento de compensações tem dois passos e sua execução resulta na geração dos WED-state $S5$ e $S6$. A partir de $S6$, é acionado o mecanismo de oferecimento adapatativo, que detecta um paralelismo ocorrido durante as execuções de t_2 e t_3, e, por isso, executa outro encadeamento de compensações a partir de $S6$, mas, nesse caso, de apenas um passo, gerando $S7$, que, finalmente, pode ser oferecido aos WED-triggers do WED-flow. Feito isso, com a geração do WED-state $S14$, que satisfaz a condição final do WED-flow, a execução termina com sucesso.

	id usuário	Status reserva	Mod. do carro	Categoria carro	Período reserva	Doc. usuário	Placa carro	Status carro	multa	avarias	total
S0	*null*	requisitada	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*
S1	**120**	**iniciada**	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*
S2	120	iniciada	*null*	*null*	*null*	**rg.png**	*null*	*null*	*null*	*null*	*null*
S3	120	iniciada	**Ford Fiesta**	**AA**	**01/02 a 20/02**	rg.png	*null*	*null*	*null*	*null*	*null*
S4	120	**confirmada**	Ford Fiesta	AA	01/02 a 20/02	rg.png	*null*	*null*	*null*	*null*	*null*
S5 ‖ S3	120	**iniciada**	Ford Fiesta	AA	01/02 a 20/02	rg.png	*null*	*null*	*null*	*null*	*null*
S6 ‖ S2	120	iniciada	***null***	***null***	***null***	rg.png	*null*	*null*	*null*	*null*	*null*
S7 ‖ S1	120	iniciada	*null*	*null*	*null*	***null***	*null*	*null*	*null*	*null*	*null*
S9	120	iniciada	*null*	*null*	*null*	**rg.png**	*null*	*null*	*null*	*null*	*null*
S10	120	iniciada	**Ford Fiesta**	**AA**	**01/02 a 20/02**	rg.png	*null*	*null*	*null*	*null*	*null*
S11	120	**confirmada**	Ford Fiesta	AA	01/02 a 20/02	rg.png	*null*	*null*	*null*	*null*	*null*
S12	120	confirmada	Ford Fiesta	AA	01/02 a 20/02	rg.png	**ABC 0123**	**retirado**	*null*	*null*	*null*
S13	120	confirmada	Ford Fiesta	AA	01/02 a 20/02	rg.png	ABC 0123	**devolvido**	*null*	*null*	*null*
S14	120	**paga**	Ford Fiesta	AA	01/02 a 20/02	rg.png	ABC 0123	devolvido	*null*	*null*	**90.0**

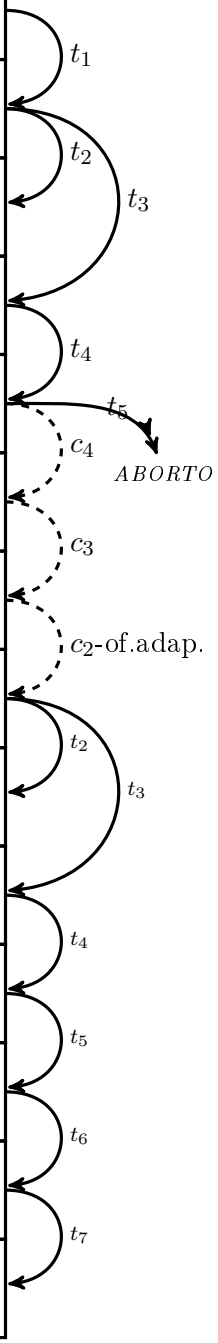


Figura A.3: *Execução de uma instância de WED-flow em que uma exceção é detectada e tratada pela execução de um encadeamento de compensações seguido de um oferecimento adaptativo.*

A.2.3 Execução com detecção de exceção: Oferecimento adaptativo

No exemplo ilustrado na Figura A.4, a execução da instância é interrompida devido ao aborto de uma WED-transition (t_5). Em alguns casos em que o WED-state atual da interrupção é consistente, o administrador de sistema pode considerar que o oferecimento de um WED-state, sem a execução de encadeamentos de compensação ou WED-S^{+a}s, seja suficiente para dar prosseguimento a execução da instância. Simulamos essa situação executando um oferecimento adaptativo a partir de $S4$.

Como não é detectado nenhum paralelismo durante a execução de t_4, o oferecimento adaptativo vai simplesmente oferecer $S4$ a todos os WED-triggers do WED-flow. Feito isso, com a geração do WED-state $S7$, que satisfaz a condição final do WED-flow, a execução termina com sucesso.

	id usuário	Status reserva	Mod. do carro	Categoria carro	Período reserva	Doc. usuário	Placa carro	Status carro	multa	avarias	total
S0	*null*	requisitada	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*
S1	**120**	**iniciada**	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*
S2	120	iniciada	*null*	*null*	*null*	**rg.png**	*null*	*null*	*null*	*null*	*null*
S3	120	iniciada	**Ford Fiesta**	**AA**	**01/02 a 20/02**	rg.png	*null*	*null*	*null*	*null*	*null*
S4	120	**confirmada**	Ford Fiesta	AA	01/02 a 20/02	rg.png	*null*	*null*	*null*	*null*	*null*
S5	120	confirmada	Ford Fiesta	AA	01/02 a 20/02	rg.png	**ABC 0123**	**retirado**	*null*	*null*	*null*
S6	120	confirmada	Ford Fiesta	AA	01/02 a 20/02	rg.png	ABC 0123	**devolvido**	*null*	*null*	*null*
S7	120	**paga**	Ford Fiesta	AA	01/02 a 20/02	rg.png	ABC 0123	devolvido	*null*	*null*	**90.0**

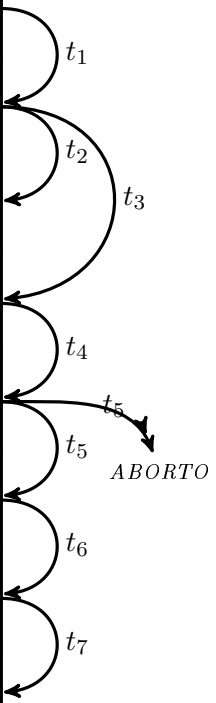


Figura A.4: *Execução de uma instância de WED-flow em que uma exceção é detectada e tratada pela execução de um oferecimento adaptativo.*

A.2.4 Execução com detecção de exceção: WED-S^{+a}

No exemplo ilustrado na Figura A.5, a execução da instância é interrompida devido a geração de um WED-state inconsistente $S6$. Nesse exemplo, $S6$ é inconsistente porque ele é o último WED-state da instância e, além disso, não satisfaz a condição de nenhum WED-trigger. Semanticamente, o que ocorre é a devolução de um veículo que tem uma avaria imprevista, ou seja, que não consta na lista de possíveis avarias da empresa de aluguel de carros. Assim, é necessária a intervenção de um administrador de sistema para decidir o que fazer. Nesse exemplo, optou-se por executar um WED-S^{+a}, o qual gerou o WED-state $S7$ que associa uma multa para a avaria imprevista e calcula o valor total a pagar. Como $S7$ satisfaz a condição final do WED-flow, a execução da instância é finalizada com sucesso.

	id usuário	Status reserva	Mod. do carro	Categoria carro	Período reserva	Doc. usuário	Placa carro	Status carro	multa	avarias	total
S0	*null*	requisitada	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*
S1	**120**	**iniciada**	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*
S2	120	iniciada	*null*	*null*	*null*	**rg.png**	*null*	*null*	*null*	*null*	*null*
S3	120	iniciada	**Ford Fiesta**	**AA**	**01/02 a 20/02**	rg.png	*null*	*null*	*null*	*null*	*null*
S4	120	**confirmada**	Ford Fiesta	AA	01/02 a 20/02	rg.png	*null*	*null*	*null*	*null*	*null*
S5	120	confirmada	Ford Fiesta	AA	01/02 a 20/02	rg.png	**ABC 0123**	**retirado**	*null*	*null*	*null*
S6	120	confirmada	Ford Fiesta	AA	01/02 a 20/02	rg.png	ABC 0123	**devolvido**	*null*	**imprevista**	*null*
S7	120	**paga**	Ford Fiesta	AA	01/02 a 20/02	rg.png	ABC 0123	devolvido	**3000.00**	imprevista	**3090.0**

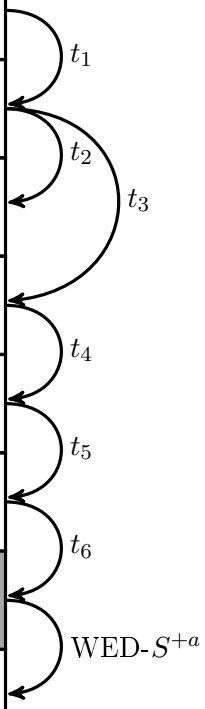


Figura A.5: *Execução de uma instância de WED-flow em que uma exceção é detectada e tratada pela execução de um WED-S^{+a}.*

A.2.5 Execução com detecção de exceção: WED-S^{-1} + oferecimento adaptativo

O exemplo da Figura A.6 é similar ao exemplo ilustrado na Figura A.5. Novamente, um WED-state inconsistente $S6$ é detectado, mas, nessa situação, o administrador de sistema optou pela execução de um WED-S^{-1} em vez de um WED-S^{+a}. Semanticamente, temos a mesma situação do exemplo da Figura A.5 em que uma avaria imprevista foi detectada, entretanto, agora supomos que o administrador oriente o funcionário a enquadrar a avaria do carro em um outro tipo de avaria (avaria "no banco", nesse exemplo). O WED-S^{-1} gera o WED-state $S8$, que é equivalente a $S5$, e, em seguida, é feito um oferecimento adaptativo desse estado. Observe que $S9$ tem o valor "nos bancos" para o WED-attribute avarias. Como $S9$ satisfaz a condição final do WED-flow, a execução da instância é finalizada com sucesso.

	id usuário	Status reserva	Mod. do carro	Categoria carro	Período reserva	Doc. usuário	Placa carro	Status carro	multa	avarias	total
S0	*null*	requisitada	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*
S1	**120**	**iniciada**	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*
S2	120	iniciada	*null*	*null*	*null*	**rg.png**	*null*	*null*	*null*	*null*	*null*
S3	120	iniciada	**Ford Fiesta**	**AA**	**01/02 a 20/02**	rg.png	*null*	*null*	*null*	*null*	*null*
S4	120	**confirmada**	Ford Fiesta	AA	01/02 a 20/02	rg.png	*null*	*null*	*null*	*null*	*null*
S5	120	confirmada	Ford Fiesta	AA	01/02 a 20/02	rg.png	**ABC 0123**	**retirado**	*null*	*null*	*null*
S6	120	confirmada	Ford Fiesta	AA	01/02 a 20/02	rg.png	ABC 0123	**devolvido**	*null*	**impre vista**	*null*
S8 ‖ S5	120	confirmada	Ford Fiesta	AA	01/02 a 20/02	rg.png	ABC 0123	**retirado**	*null*	***null***	*null*
S9	120	confirmada	Ford Fiesta	AA	01/02 a 20/02	rg.png	ABC 0123	**devolvido**	*null*	**nos bancos**	*null*
S10	120	**paga**	Ford Fiesta	AA	01/02 a 20/02	rg.png	ABC 0123	devolvido	**200.00**	nos bancos	**290.0**

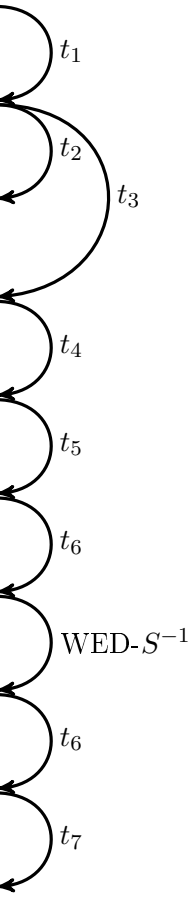


Figura A.6: *Execução de uma instância de WED-flow em que uma exceção é detectada e tratada pela execução de um WED-S^{-1} seguida da execução de um oferecimento adaptativo.*

A.2.6 Execução com detecção de exceção: WED-S^{-1} + encadeamento de compensações + oferecimento adaptativo

No exemplo ilustrado na Figura A.7, a execução da instância é interrompida devido a geração de um WED-state inconsistente $S4$, que é o último WED-state da instância e não satisfaz a WED-condition de nenhum WED-trigger. Semanticamente, o que ocorre é que, devido a alguma pane ou *bug*, foi atribuído o valor "desconhecido" ao WED-attribute "status_reserva" do WED-state $S4$. Assim, é necessária a intervenção de um administrador de sistema para decidir como resolver essa situação. Nesse exemplo, supomos que o administrador corrigiu as causas que levaram à geração do valor incorreto do WED-attribute e, em seguida, executou um WED-S^{-1}, o qual gerou o WED-state consistente $S5$ (equivalente a $S3$). A partir de $S5$ o administrador de sistema opta pela execução de um encadeamento de compensações de dois passos, que gerou os WED-states $S6$ e $S7$. A partir de $S7$ é executado o mecanismo de oferecimento adaptativo, que, não detectando nenhuma execução paralela, oferece esse WED-state a todos os WED-triggers do WED-flow. Com a geração de $S14$, que satisfaz a condição final do WED-flow, a execução da instância é finalizada com sucesso.

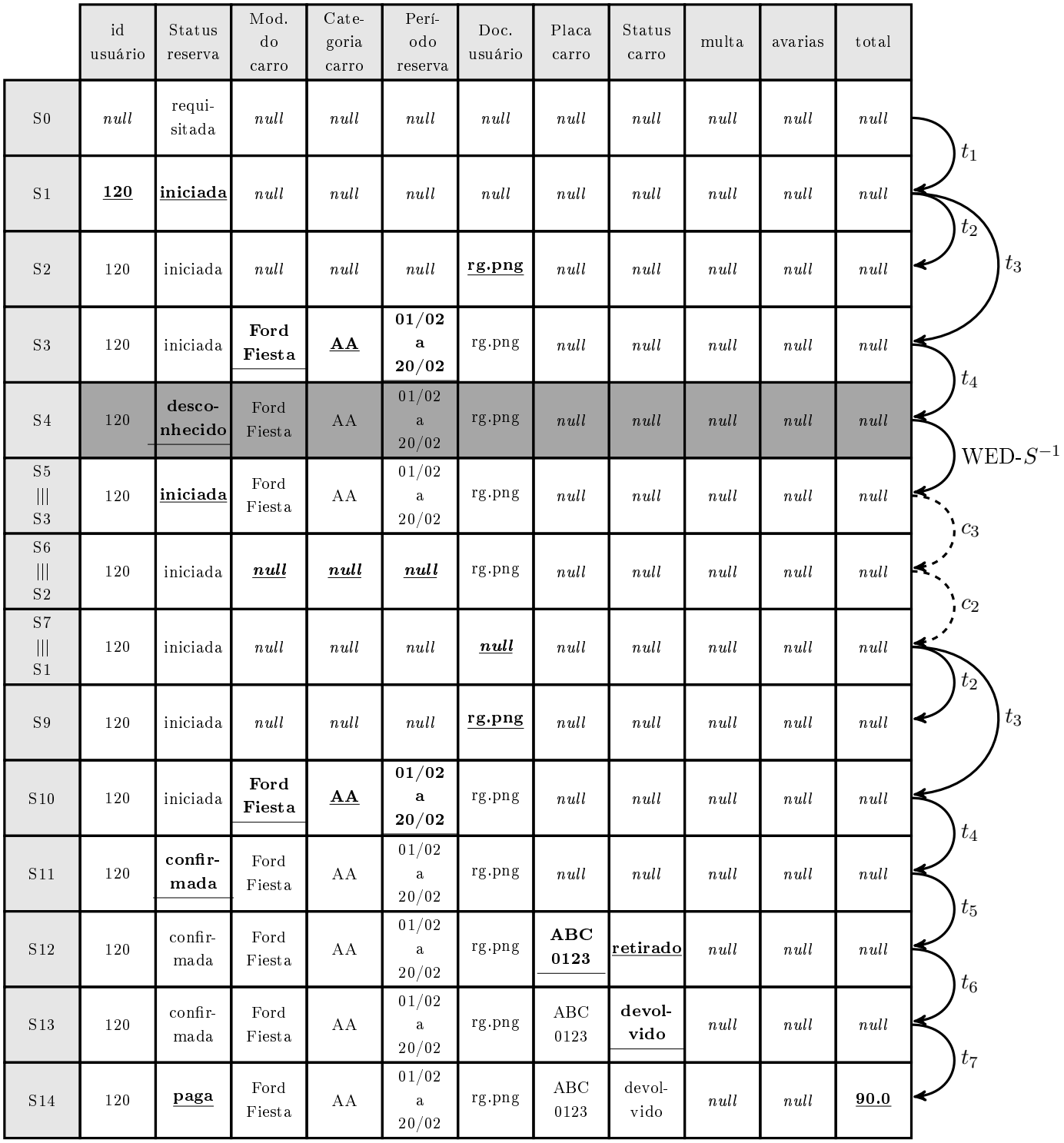


Figura A.7: *Execução de uma instância de WED-flow em que uma exceção é detectada e tratada pela execução de um WED-S^{-1} seguida da execução de um encadeamento de compensações e de um oferecimento adaptativo.*

Referências Bibliográficas

[Alo05] Gustavo Alonso. Transactional business processes. Em *Process-Aware Information Systems*. Wiley, 2005. 5, 11, 14

[Ant11] Pedro Antunes. BPM and exception handling: Focus on organizational resilience. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 41(3):383–392, 2011. 2, 16, 28

[App] Appian. Sítio da ferramenta "Appian BPM". http://www.appian.com/. [Último acesso em Abril de 2013]. 30

[Aur] Aurea. Sítio da ferramenta "Savvion". http://www.aurea.com/savvion/. [Último acesso em Abril de 2013]. 30

[BGG+11] Sami Bhiri, Walid Gaaloul, Claude Godart, Olivier Perrin, Maciej Zaremba e Wassim Derguech. Ensuring customised transactional reliability of composite services. *Journal of Database Management*, 22(2):64–92, 2011. 27

[BN97] Philip .A. Bernstein e Eric Newcomer. *Principles of Transaction Processing*. Morgan Kaufmann, 2 edição, 1997. 1, 5, 6, 11, 13, 14

[Bon] Bonitasoft. Sítio da ferramenta de gerenciamento de workflows "Bonitasoft". http://br.bonitasoft.com/. [Último acesso em Abril de 2013]. 29

[CCPP99] Fabio Casati, Stefano Ceri, Stefano Paraboschi e Guiseppe Pozzi. Specification and implementation of exceptions in workflow management systems. *ACM Transactions on Database Systems*, 24(3):405–451, Setembro 1999. 13, 14, 26, 28

[Dav78] Charles T. Davies. Data processing spheres of control. *IBM Systems Journal*, 17(2):179–198, Junho 1978.

[DGG95] Klaus R. Dittrich, Stella Gatziu e Andreas Geppert. The active database management system manifesto: A rulebase of adbms features. Em *Rules in Database Systems*, volume 985, páginas 3–20, 1995. 12

[DIa] Grupo DATA-IME. Sítio do grupo DATA-IME. http://data.ime.usp.br. [Último acesso em Abril de 2013]. 2, 49, 68

[DIb] Grupo DATA-IME. Sítio do WED-flow. http://data.ime.usp.br/wedflow. [Último acesso em Abril de 2013]. 56, 77

[EL95] Johann Eder e Walter Liebhart. The workflow activity model WAMO. Em *Proceedings of the 3rd international conference on Cooperative Information Systems*, páginas 87–98, 1995. 15, 26, 27

[EN05] Ramez Elmasri e Shamkant B. Navathe. *Sistemas de Banco de Dados*. Pearson Addison Wesley, 4 edição, 2005. 5, 6

[EVWH06] Sharanya Eswaran, David Del Vecchio, Glenn S. Wasson e Marty Humphrey. Adapting and evaluating commercial workflow engines for e-science. Em *Second International Conference on e-Science and Grid Technologies (e-Science 2006), Amsterdam, The Netherlands*, página 20, 2006. 30, 68

[FBTP12] João E. Ferreira, Kelly R. Braghetto, Osvaldo K. 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*, páginas 303–310, 2012. 2, 4, 16, 19, 22, 31, 33, 37, 41, 44, 46

[FF00] João E. Ferreira e Marcelo Finger. *Controle de concorrência e distribuição de dados: a teoria clássica, suas limitações e extensões modernas*. IME-USP, 2000. 6

[Fon] YAWL Fondation. Sítio do "YAWL Fondation". http://www.yawlfoundation.org. [Último acesso em Abril de 2013]. 29

[FTMP10] João E. Ferreira, Osvaldo K. Takai, Simon Malkowski e Calton Pu. Reducing exception handling complexity in business process modeling and implementation: the WED-flow approach. Em *Proceedings of the 18th International Conference on Cooperative Information Systems*, páginas 150–167, 2010. 2, 4, 16, 19, 22, 31

[Gar13] Marcela O. Garcia. Implementação do arcabouço wed-flow para controle de processos transacionais, 2013. 33, 49, 51, 52, 53, 54, 56

[GJCV09] Ricardo Garcês, Tony de Jesus, Jorge Cardoso e Pedro Valente. Open source workflow management systems: A concise survey. Em Layna Fischer, editor, *2009 BPM & Workflow Handbook*, páginas 179–190. Future Strategies Inc., Lighthouse Point, FL, USA, 2009. 29, 68

[GMS87] Hector Garcia-Molina e Kenneth Salem. Sagas. volume 16, páginas 249–259. ACM, December 1987. 9, 17

[GR93] Jim Gray e Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers, 1993. 5, 7, 8, 9

[GSBF12] Marcela O. Garcia, Pedro Paulo S. B. Silva, Kelly R. Braghetto e João E. 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*, páginas 19–24, 2012. 2, 3, 4, 17, 47, 49, 67

[IBM] IBM. Sítio da ferramenta "IBM BPM". http://www-01.ibm.com/software/integration/business-process-manager/. [Último acesso em Abril de 2013]. 30

[JLM+12] Sven Jörges, Anna-Lena Lamprecht, Tiziana Margaria, Ina Schaefer e Bernhard Steffen. A constraint-based variability modeling framework. *STTT*, 14(5):511–530, 2012. 67

[KBTB98] Peter J. Kammer, Gregory Alan Bolcer, Richard N. Taylor e Mark Bergman. Techniques for supporting dynamic and adaptive workflow. *Computer Supported Cooperative Work*, 9:269–292, 1998. 25

[LS07] Ruopeng Lu e Shazia Wasim Sadiq. A survey of comparative business process modeling approaches. Em Witold Abramowicz, editor, *Proceedings of the 10th international conference on Business information systems*, volume 4439 of *Lecture Notes in Computer Science*, páginas 82–94, Berlin, 2007. Springer. 11, 25, 26, 27, 68

[LSKM00] Zongwei Luo, Amit Sheth, Krys Kochut e John Miller. Exception handling in workflow systems. *Applied Intelligence*, 13:125–147, 2000. 1, 14, 15, 25

[MaA07] Hernâni Mourão e Pedro Antunes. Supporting effective unexpected exceptions handling in workflow management systems. Em *In Proceedings of the 2007 ACM Symposium on Applied Computing (Seoul, Korea, March 11 - 15, 2007). SAC '07.* ACM Press, 2007. 28

[Mar12] An implementation of a transaction model for business process systems. *Journal of Information and Data Management*, 2012. 2, 32, 33, 51

[Mica] Microsoft. Sítio da ferramenta "Microsoft BizTalk Server". http://www.microsoft.com/biztalk/en/us/default.aspx. [Último acesso em Abril de 2013]. 30

[Micb] Microsoft. Sítio da ferramenta "Windows Workflow Foundation". http://msdn.microsoft.com/en-us/vstudio/jj684582.aspx. [Último acesso em Abril de 2013]. 30

[Mos81] Eliot B. Moss. Nested transactions: an approach to reliable distributed computing. Relatório técnico, Cambridge, MA, USA, 1981. 8

[MS02] Peter Mangan e Shazia Sadiq. On building workflow models for flexible processes. *Australian Computer Science Community*, 24(2):103–109, Janeiro 2002. 25, 26

[Ora] Oracle. Sítio da "Oracle BPEL Process Manager". http://www.oracle.com/technetwork/middleware/bpel/overview/index.html. [Último acesso em Abril de 2013]. 30

[Pet62] Carl A. Petri. *Kommunikation mit Automaten.* Tese de Doutorado, Institut für instrumentelle Mathematik, Bonn, 1962. 12

[Pos] PostgreSQL. Sítio da linguagem de programação "Ruby". http://www.postgresql.org. [Último acesso em Abril de 2013]. 50

[RD98] Manfred Reichert e Peter Dadam. Adept flex - supporting dynamic changes of workflows without loosing control. *Journal of Intelligent Information Systems*, 10:93–129, 1998. 25, 26, 27

[RDB03] Manfred Reichert, Peter Dadam e Thomas Bauer. Dealing with forward and backward jumps in workflow management systems. *Software and System Modeling*, 2(1):37–58, 2003. 2, 25, 27, 34

[Rec] Active Record. Sítio do ferramenta da biblioteca "Active Record". http://api.rubyonrails.org/classes/ActiveRecord/Base.html. [Último acesso em Abril de 2013]. 50

[Rod12] Maurício C. Rodrigues. Tratamento de eventos aplicado à composição de serviços *web.*, 2012. 68

[RoTFoIT07] N.C. Russell e Queensland University of Technology. Faculty of Information Technology. *Foundations of Process-aware Information Systems.* Queensland University of Technology, Brisbane, 2007. 11, 12

[Rub] Ruby. Sítio da linguagem de programação "Ruby". http://www.ruby-lang.org/pt/. [Último acesso em Abril de 2013]. 50

[RvdAtH] Nick Russell, Wil M. P. van der Aalst e Arthur H. M. ter Hofstede. Workflow exception patterns. Em *Proceedings of 18th CAiSE*, páginas 288–302. 12

[Saa93] Heikki Saastamoinen. Rules and exceptions. *Information Modeling and Knowledge Bases IV: Concepts, Methods and Systems*, páginas 271–286, 1993. 1, 14, 15

[SABS02] Heiko Schuldt, Gustavo Alonso, Catriel Beeri e Hans-Jörg Schek. Atomicity and isolation for transactional processes. *ACM Transactional Database Systems*, 27(1):63–116, 2002. 27

[SBF12] Pedro Paulo S. B. Silva, Kelly R. Braghetto e João E. Ferreira. Uma abordagem ad-hoc para o tratamento de exceções em processos transacionais. Em *Proceedings of the 27th Brazillian Symposium on Databases*, páginas 153–160, 2012. 67

[Spe] Workflow Management Coalition Specification. *Workflow Management Coalition Terminology & Glossary (Document No. WFMC-TC-1011)*. Workflow Management Coalition Specification. 11, 13

[SR93] Amit P. Sheth e Marek Rusinkiewicz. On transactional workflows. *IEEE Data Engineering Bulletin*, 16(2):37–40, 1993. 14

[SSO01] Shazia Sadiq, Wasim Sadiq e Maria Orlowska. Pockets of flexibility in workflow specification. Em *Proceedings of the 20th International Conference on Conceptual Modeling: Conceptual Modeling*, páginas 513–526, London, UK, UK, 2001. Springer-Verlag. 25

[vdAADtH04] Wil M. P. van der Aalst, Lachlan Aldred, Marlon Dumas e Arthur H. M. ter Hofstede. Design and implementation of the yawl system. Em *Advanced Information Systems Engineering, 16th International Conference, CAiSE 2004, Riga, Latvia, June 7-11, 2004, Proceedings*, volume 3084 of *Lecture Notes in Computer Science*, páginas 142–159. Springer, 2004. 29

[vdAHW03] Wil M. P. van der Aalst, Arthur H. M. Ter Hofstede e Mathias Weske. Business process management: A survey. Em *Proceedings of the 1st International Conference on Business Process Management, volume 2678 of LNCS*, páginas 1–12. Springer-Verlag, 2003. 13

[vdAPS09] Wil M. P. van der Aalst, Maja Pesic e Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113, 2009. 12, 25, 26, 27, 28, 29, 67

[VV11] Krishnamurthy Vidyasankar e Gottfried Vossen. Multi-level modeling of web service compositions with transactional properties. *Journal of Database Management*, 22(2):1–31, 2011. 27

[WdLB03] Jacques Wainer e Fábio de Lima Bezerra. Constraint-based flexible workflows. Em *Groupware: Design, Implementation, and Use – Proceedings of the 9th International Workshop, CRIWG 2003*, volume 2806 of *Lecture Notes in Computer Science*, páginas 151–158. Springer, 2003. 25, 27, 28

[WH93] Gerhard Weikum e Christof Hasse. Multi-level transaction management for complex objects: implementation, performance, parallelism. *The VLDB Journal*, 2(4):407–454, Outubro 1993. 10

[ZNBB94] Aidong Zhang, Marian Nodine, Bharat Bhargava e Omran Bukhres. Ensuring relaxed atomicity for flexible transactions in multidatabase systems. *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, 23(2):67–78, Maio 1994. 10, 27