PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Programa de Pós-Graduação em Informática

VERIFICAÇÃO AUTOMATIZADA DE REGRAS DE SEQUENCIAMENTO DE EXECUÇÃO EM WORKFLOWS

Cristiano de Magalhães Barros

Belo Horizonte

2010

Cristiano de Magalhães Barros

VERIFICAÇÃO AUTOMATIZADA DE REGRAS DE SEQUENCIAMENTO DE EXECUÇÃO EM WORKFLOWS

Dissertação apresentada ao Programa de Pós-Graduação em Informática como requisito parcial para obtenção do Grau de Mestre em Informática pela Pontifícia Universidade Católica de Minas Gerais.

Orientador: Mark Alan Junho Song

Belo Horizonte

2010

FICHA CATALOGRÁFICA

Elaborada pela Biblioteca da Pontifícia Universidade Católica de Minas Gerais

Barros, Cristiano de Magalhães
B277v Verificação automatizada de regras de sequenciamento de execução em workflows / Cristiano de Magalhães Barros. – Belo Horizonte, 2010.
83f. : il.

Orientador: Mark Alan Junho Song
Dissertação (Mestrado) – Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-graduação em Informática.
Bibliografia.

1. Análise de sistemas – Teses. 2. Fluxo de trabalho. 3. Fluxograma I. Song, Mark Alan Junho. II. Pontifícia Universidade Católica de Minas Gerais. III. Título.

CDU: 681.3.031

Bibliotecário: Fernando A. Dias – CRB6/1084

PUC Minas
Programa de Pós-graduação em Informática

FOLHA DE APROVAÇÃO

Verificação automatizada de regras de sequenciamento de execução em workflows

CRISTIANO DE MAGALHÃES BARROS

Dissertação defendida e aprovada pela seguinte banca examinadora:

Prof. Mark Alan Junho Song - Orientador (PUC Minas)
Doutor em Ciências da Computação - UFMG

Prof. Autran Macedo (UFU)
Doutor em Ciência da Informação - UFMG

Prof. Zenilton Kleber Gonçalves do Patrocínio Júnior (PUC Minas)
Doutor em Ciências da Computação - UFMG

Belo Horizonte, 27 de agosto de 2010.

Anel Rodoviário – KM 23,5, Rua Walter Ianni, 255 – São Gabriel / Belo Horizonte – MG / CEP: 31980-110
Fone: (31) 3439-5204 – Fax: (31) 3439-5204 – e-mail: mestradoinfsec@pucminas.br

Dedicado à
Manuela,
Christiane,
e aos meus pais.

AGRADECIMENTOS

Agradeço inicialmente aos meus pais pelo apoio e pelo exemplo.

Agradeço também ao apoio geral fornecido por minha amada esposa Christiane.

Agradeço fortemente aos que tiveram impacto direto sobre a realização deste trabalho. Destaco a importância do aprendizado e dedicação de meu orientador, professor Mark. Agradeço ao professor Zenilton pelo apoio durante a fase de projeto e a ajuda na formatação do texto. Agradeço também a todos os professores do Mestrado da PUC Minas São Gabriel, que me proporcionaram a oportunidade de vislumbrar o quão vasto e interessante é o campo da computação/informática, característica essa que passou despercebida durante a graduação.

Agradeço ao Tiago Carvalho e ao Eduardo Borges pelas ajudas técnicas indispensáveis.

Agradeço finalmente aos colegas de trabalho e amigos, Marconi, Márcia, Fabrício, Pedro e Wagner, pelo compartilhamento de experiências em pós-graduação e ajudas diversas para que o caminho pudesse ser percorrido.

RESUMO

No ambiente atual de alta competitividade as organizações têm investido na melhoria de seus processos de produção e de gestão em busca de maior eficiência. Para isso necessariamente deve-se formalizar tais processos em modelos de modo a analisá-los. Assim como é importante analisar artefatos da engenharia de *software* antes de iniciar a construção de sistemas de informação, é importante validar as definições contidas nos modelos de negócio sob a forma de regras de negócio para que o modelo reflita a realidade do processo e suas exigências. Para isso o trabalho corrente objetivou a definição de uma abordagem com a criação de um *software* de apoio à realização de validações das regras de sequenciamento de execução em fluxos de trabalho (*workflows*).

Palavras-chave: modelagem de negócios, fluxos de trabalho, validação, workflows, verificação de modelos.

ABSTRACT

In the present context of high competition, the enterprises are investing in the improvement of the efficiency of their production and management processes. To accomplish such tasks, one viable way is to formalize those processes into business models, so it is possible to evaluate and improve them. As important as the analysis of software engineering artifacts prior to building information systems, also is important the analysis of the definitions existing in the business models, to ensure that they reflect the reality of the process and its demands. This dissertation aimed to create a software that supports the execution of validations of execution rules incorporated to workflows.

Keywords: workflows, business modeling, validation, model checking.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

SUMÁRIO

1 INTRODUÇÃO

Diversos autores caracterizam o ambiente atual de negócios como altamente competitivo, sendo essencial buscar tanto formas de diferenciação de seus produtos e serviços, como também formas de ampliação da eficiência de modo a obter sucesso em suas atividades empresariais. Além disso, nos últimos anos a área de análise de negócios vem tomando uma maior importância por atacar problemas decorrentes da relação entre os setores de negócio e os setores de tecnologia da informação, sejam eles internos ou externos a uma mesma organização (TAKEMURA, 2008). Nesse sentido, é usual utilizar-se da revisão dos processos de negócio como forma de aumentar a eficiência dos mesmos: exclusão de atividades desnecessárias, integração entre processos, simplificação de procedimentos, informatização de execuções repetitivas, entre outras (INTERNATIONAL INSTITUTE OF BUSINESS ANALYSIS, 2009).

Por processos de negócio, *workflows* ou fluxos de trabalho entende-se o encadeamento de tarefas visando atingir determinado objetivo. O progresso de execução de um fluxo dá origem a uma série de estados específicos. Ao mesmo tempo, uma série de regras gerais e específicas se aplica ao modelo de negócios, tais como a ausência de *deadlocks* ou qualquer restrição sobre a execução das tarefas.

Dentro da engenharia de sistemas, a modelagem de negócios formaliza as regras de encadeamento de fluxos de trabalho de forma a subsidiar a criação de sistemas de informação cujo propósito seria o de apoiar e automatizar tarefas deste fluxo. É reconhecida como boa prática a modelagem do processo suportado antes da efetiva elaboração de um sistema de informação para que este seja criado utilizando premissas e regras que sejam efetivamente usadas e não somente aquelas idealizadas ou formalmente demandadas. Dessa forma, não seria desejável investir na criação de uma ferramenta de automatização que se baseia em informações e execuções ineficientes, incorretas ou não executadas na realidade do processo.

A importância de modelar fluxos de trabalho previamente à elaboração de sistemas

pode ser diretamente afetada se a modelagem de negócio não refletir a realidade, mais especificamente se ela não atender às regras de negócio. Validar o modelo de modo a garantir a exatidão do mesmo em relação às regras de negócio é de grande valia para evitar incorreções na modelagem de *softwares* e necessidade de reimplementações e correções. Esta afirmação se torna mais presente se aplicada ao desenvolvimento de aplicações complexas ou que envolva a integração com vários outros sistemas de informação.

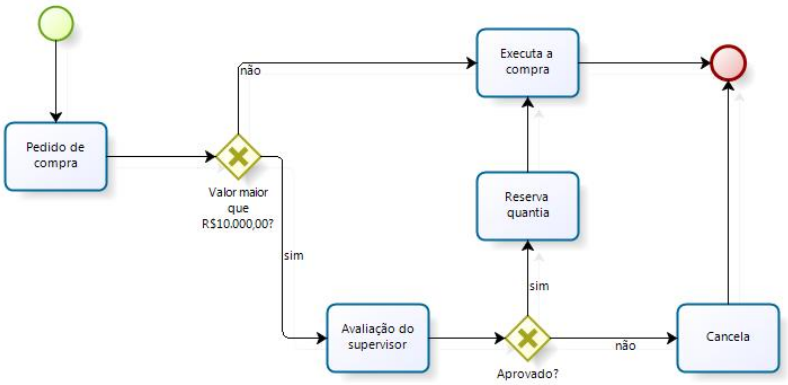

Figura 1: Diagrama de exemplo

Tome como exemplo a Figura 1, em que algumas verificações podem ser realizadas:

- Todas as compras acima de R$10.000,00 precisam ser aprovadas?
- Toda vez que a compra é processada, a quantia é previamente reservada?
- Toda vez que se faz o pedido, a compra é efetivamente realizada?
- Existe a possibilidade de o supervisor aprovar pedidos abaixo de R$10.000,00?

No caso das perguntas anteriores, a resposta será obtida utilizando *model checking*. A técnica de verificação de modelos ou *model checking* pressupõe a avaliação de afirmativas a partir de um modelo de transição de estados. Tal modelo pode ser elaborado a partir de diversos campos do conhecimento, incluindo o tema de processos de negócio. O propósito da técnica é fornecer um meio objetivo de avaliar expressões de lógica temporal buscando confirmar afirmativas ou obter contraexemplos no caso de reprovações. Dessa forma, a técnica é essencial por fornecer um meio de explorar todos os caminhos possíveis de execução dentro de um fluxo de trabalho, além de fornecer a avaliação das afirmativas.

A pesquisa se justifica inicialmente pela importância que o tema de modelagem de negócio vem assumindo juntamente com a engenharia de *software*. Com uma maior

profissionalização no desenvolvimento de *softwares* por grande parte das empresas, tornou-se evidente os ganhos obtidos na utilização de métodos formais de engenharia de sistema e mais especificamente da engenharia de *software*(ANDRADE et al., 2004).

Com o predomínio da microinformática sobre as plataformas altas, o processo de elaboração de sistemas ganhou maior evidência, ao mesmo tempo em que evidenciou os prejuízos decorrentes dessa elaboração sem um correto processo de planejamento e pré-modelagem. Dentre as melhores práticas no desenvolvimento de sistemas, a importância de se avaliar previamente os processos de negócio que serão afetados pelos sistemas tem aumentado a partir da percepção do ganho final obtido por sua adoção.

Outra prática de destaque é a adequada observação das técnicas de teste de *software* quando se passou a perceber o impacto negativo dos erros de *software* e a frequência com a qual aparecem na adoção dos sistemas. Os desenvolvedores de *software* passaram a dar grande atenção nos testes de homologação para reduzir a incidência de erros em *software* entregues. No entanto, alguns erros não têm origem somente nas etapas de desenvolvimento e análise de *software*, mas podem ser heranças da etapa de modelagem de negócios (BAKER, 2001).

Erros decorrentes de modelagens de negócio incorretas são de difícil detecção e podem impactar de forma decisiva a aplicabilidade do *software*. Além disso, a modelagem de negócio aumenta a eficiência do processo de análise e desenvolvimento de sistemas pois detalha os aspectos que devem ser considerados na execução destas tarefas.

1.1 Objetivos

O presente trabalho tem como objetivo a proposta de uma abordagem para a verificação formal de regras de sequenciamento de execução de tarefas presentes em *workflows* escritos na linguagem XPDL (*XML Process Definition Language*) de descrição de fluxos de trabalho.

Mais detalhadamente, o objetivo do trabalho foi a criação de um método de tradução da especificação de fluxos de trabalho que permita a verificação da validade de regras de execução de tarefas em um verificador de modelos, podendo este método ser automatizado por meio de um *software*.

1.2 Trabalhos relacionados

A aplicação de *model checking* sobre a verificação formal de regras de negócio em fluxos de trabalho ainda é restrita se comparada com a aplicação sobre as fases de análise de requisitos de *software*. No entanto, tem se mostrado adequada e útil. Existem diversos trabalhos dedicados à validação de *workflow* por meio de verificação de modelos, como será apresentado a seguir.

A maioria dos trabalhos analisados trata da verificação de regras gerais, intrínsecas ao próprio formato do *workflow*, como a procura pela inexistência de estados inalcançáveis, *deadlocks* ou *livelocks*. *Deadlocks* ocorrem quando uma atividade aguarda por uma resposta vinda de outra atividade que está aguardando uma resposta da primeira. *Livelocks* ocorrem quando uma atividade repassa um recurso para outra atividade e esta o devolve sem modificações. Além destes, os trabalhos geralmente tratam da verificação de propriedades do tipo *safety* ou *liveness*. A verificação de *safety* procura responder se determinada propriedade será sempre garantida em todos os caminhos possíveis, como por exemplo se o processo sempre vai atingir o nó final do fluxo. Já a verificação de *liveness* procura pela satisfação de uma propriedade em algum momento do fluxo (PETRUCCI; CARVALHO; FELIX, 2006).

No trabalho de Aalst (1997), o autor utiliza-se de Redes Petri para a verificação das regras gerais de *deadlocks* ou *livelocks*, sem se preocupar com questões de restrição temporais ou de alocação de recursos.

Um amplo trabalho de aplicação de *model checking* na validação de regras de negócio foi desenvolvido em Eshuis (2002) incluindo a criação de ferramenta de verificação automatizada de modelos UML sobre o framework "Toolkit for Conceptual Modeling" (TCM) e o verificador NuSMV. O objetivo do trabalho foi a definição de uma semântica formal para diagramas de atividade adequada para a modelagem de fluxo de trabalho passível de verificação de requisitos funcionais por *model checking*. Por meio de diversos exemplos foi possível concluir sobre a importância da verificação de regras de negócio e a viabilidade de adoção da notação UML para a definição de fluxos de trabalho, desde que um padrão formal seja imposto na modelagem.

Alguns outros trabalhos avaliam aspectos relacionados a outros tipos de regras de negócio. Em Santos, Ferreira e Tribolet (2007), os autores utilizam a linguagem PROMELA como descritor da especificação dos fluxos de trabalho e o verificador SPIN. Os objetivos do trabalho são a execução de simulações de tempo de execução de processos de

atendimento de um setor de urgência de um hospital e também a viabilidade da verificação formal de regras de negócio por *model checking*. No entanto tal trabalho, em relação à validação de regras de negócio, não fornece nada além da conclusão sobre a viabilidade de utilização da técnica para tal fim. Já em relação à possibilidade de simulação de cenários e tempos de execução de processos, apesar de não estar incluída no escopo deste trabalho, o trabalho demonstrou uma metodologia mais adequada.

Em Guelfi e Mammar (2004) os autores avaliam a verificação de regras de negócio em comércio eletrônico por meio da linguagem PROMELA, para a representação da lógica do processo, e a ferramenta de verificação de *model checking* SPIN. Nesse trabalho avaliou-se como exemplo a verificação quanto a não ocorrência de interrupções e o cumprimento dos requisitos de prontidão, pontualidade e rapidez de processamento de pedidos, obtendo resultados na avaliação que o fizeram corrigir seu mapeamento do processo de negócio.

Em Petrucci, Carvalho e Felix (2006), os autores procuram avaliar propriedades do tipo *safety* e *liveness* de forma a determinar o cumprimento de regras de sincronização e paralelismo de atividades concorrentes. Para isso utilizam a técnica de álgebra de processo CCS (*Calculus of Communicating Systems*) e *model checking*. A técnica CSS permite representar a transição de atividades e divisões do fluxo do tipo AND e OR como uma linguagem próxima à da matemática. Após o processo ser traduzido para esta linguagem ele pode ser incorporado ao verificador.

Em Gruhn e Laue (2006), os autores criaram um tradutor da especificação de *workflows* em máquinas de estados temporais para a avaliação também de propriedades temporais destas especificações pelo verificador UPPAAL, tais como o cumprimento de prazos de execução. Além disso, o trabalho também inclui na verificação propriedades relacionadas à ocupação de recursos. Se, por exemplo, alguma tarefa exigir a intervenção de uma determinada pessoa que esteja alocada em outro processo ou até em outra instância do mesmo processo, o fluxo entrará em *deadlock*.

Existem também trabalhos não diretamente relacionados à verificação de fluxos de trabalho mas sim aos passos subsequentes do processo de validação de *software*, como os apresentados a seguir.

Em Khöler, Tirenni e Kumaran (2002) os autores demonstram um método de comparação entre a modelagem de processos de negócio e sua representação de como solução informatizada no formato de um autômato não-determinístico de estados, por meio da aplicação de *model checking*. Criou-se uma metodologia para determinar se certo estado da representação do modelo tecnológico é alcançável, especificando opcionalmente

o número máximo de passos possíveis.

Em Pfeiffer, Rossak e Speck (2004), os autores utilizam a técnica para validar se a execução de blocos de código de *software* a especificação do fluxo de trabalho associado ao mesmo. Sendo assim, o trabalho se aplicaria ao passo seguinte à validação dos processos de negócio, tema central desta dissertação, que seria a avaliação da adequação do código-fonte ao processo de negócio envolvido.

O trabalho de Schaad, Sohr e Drouineaud (2007) demonstra uma metodologia de verificação de processos para impedir violações de regras de acesso de usuário em fluxos que permitam delegação e revocação de tarefas.

Em Sasaki e Iijima (2007), além da verificação de regras de ordem de execução das tarefas, também se verifica a ocorrência de violações em estados invariantes do processo como a alteração não planejada de características de cada atividade. No exemplo descrito no trabalho, o verificador determina se um parâmetro contendo o número do cartão de crédito de um processo de compra não é alterado durante toda a execução do fluxo.

Existem também trabalhos que inovam procurando a verificação de regras específicas por meio de técnicas não dependentes de *model checking*. Em Happel e Stojanovic (2006) é demonstrada uma técnica diferente de verificação de regras de negócio por meio da utilização de ontologias buscando executar verificações semânticas de regras de negócio. Os autores se utilizam da ferramenta sBPM para desenhar o fluxo e realizar a verificação de regras descritas no mesmo aplicativo. Como forma de testar a metodologia foram utilizados fluxos de negócio de processamento de carne animal para consumo humano e as regulações aplicáveis a tal atividade foram utilizadas como base para a verificação de conformidade. Apesar de o artigo tratar somente da apresentação da técnica, sem demonstrar detalhadamente a viabilidade da metodologia, a forma como as verificações foram executadas e os resultados obtidos, tal técnica se mostra promissora por permitir validações mais complexas e mais acessíveis a profissionais não especialistas por se utilizar de ontologias e verificações semânticas.

Finalmente, alguns trabalhos tratam também das regras de sequenciamento de tarefas e alcance de estados, assim como o presente trabalho. Em Janssen et al. (1999) propôs a utilização de *model checking* para a verificação de modelos de negócio utilizando a ferramenta Testbed, a linguagem AMBER e o verificador SPIN, demonstrando a capacidade de utilização de tal ferramental por equipes não especialistas com o objetivo de melhorar o desenho de processos por meio de simulações de execução. O objetivo de seu trabalho é prover a validação de quatro tipos de relacionamento entre as atividades

de um fluxo de trabalho: rastreamento, consequência, ocorrência combinada e precedência. Dentre estes, o presente trabalho utiliza-se do padrão de consequência para traçar a validação da sequência de atividades.

A tese de Matousek (2003), que utilizou a linguagem XPDL e o verificador de modelos SPIN, demonstra uma metodologia de verificação por meio de um estudo de caso de processos de uma agência de viagens. As validações executáveis por meio de sua metodologia são, não somente a validação de existência de atividades inalcançáveis, de *deadlocks* e *livelocks*, mas também a validação do sequenciamento das atividades. A especificação XPDL é manualmente traduzida para a linguagem PROMELA que é posteriormente executada manualmente no verificador SPIN.

Em Fisteus e Lopez (2004) cria-se um conjunto de ferramentas chamado VERBUS para a modelagem e verificação de processos de negócio, tendo demonstrado sua aplicabilidade. O objetivo do trabalho também foi o de avaliar propriedades invariantes e o alcance de estados. No entanto tal ferramenta avalia também se existem transições que não são alcançáveis em nenhum dos caminhos possíveis. Segundo o autor, este último caso é um indicador de problemas de especificação ou desenho do processo.

Na mesma direção do trabalho anterior, Huang (2006) utiliza a linguagem PROMELA e o verificador SPIN para verificar especificações de processos corporativos de venda em XPDL. Seu objetivo é a criação de um sistema que permita verificar a existência de características sistêmicas (inexistência de *deadlocks* e *livelocks*) e também a verificação de propriedades *ad hoc* de interdependência entre atividades.

Também utilizando Redes Petri, o artigo de Takemura (2008) descreve um método para avalição se todas as tarefas são concluídas quando um processo é encerrado e também a capacidade de se atingir uma determinada tarefa ou o nó final em um fluxo.

A pesquisa contida neste presente trabalho apresenta algumas diferenças em relação aos últimos trabalhos apresentados, não constituindo portanto, uma mera reprodução dos mesmos.

No trabalho Janssen et al. (1999), percebe-se que a grande diferença diz respeito à necessidade de utilização de um ferramental próprio (Testbed Studio), o qual não mais se encontra disponível para uso, para executar a modelagem e assim, poder executar validações nos fluxos.

Para o trabalho Matousek (2003), além de serem utilizados linguagem de execução e verificador diferentes (PROMELA e SPIN respectivamente) não foi criado um tradutor

automatizado assim como não foi implementada uma interface que permita ao usuário definir as consultas de verificação dos modelos.

Em Fisteus e Lopez (2004) também foi iniciada a criação de um ferramental para a realização da modelagem chamado de VERBUS, ainda não finalizada. Pelo fato dessa pesquisa estar inativa há 4 anos e pela falta de maiores detalhes sobre a metodologia, não foi possível analisar as diferenças além das linguagens de execução utilizadas (CLIPS e PROMELA) e o verificador SPIN.

Em relação à pesquisa em Huang (2006), destaca-se a diferença provocada pela necessidade de definição formal das regras para validação por meio da linguagem PROMELA, que é a mesma utilizada para a tradução da especificação do modelo. Isso gera a necessidade de usuários especialistas não somente na sintaxe da linguagem mas também com noções de técnicas de programação e lógica temporal. A pesquisa baseia-se também na tradução de modelagens UML para PROMELA por meio de outros *softwares* e utiliza tal tradução para a verificação do modelo.

Finalmente, o trabalho de Takemura (2008), não somente se diferencia por se basear em Redes Petri mas também não envolve a criação ou utilização de ferramentas de apoio à verificação mas somente a análise quanto à viabilidade de utilização de tal metodologia para a verificação de alguns propriedades tais como alcance de estados.

Ressalta-se, no entanto, que o presente trabalho apresenta uma restrição se comparado aos trabalhos anteriormente descritos. Tendo definido o público-alvo da ferramenta como sendo especialistas em análise de negócios em detrimento de especialistas em lógica temporal e *model checking*, o produto final desta pesquisa se mostra menos flexível nas verificações possíveis. Foi necessário limitar em um certo número de verificações disponibilizadas na interface, restrição que não existe em outras pesquisas nas quais as consultas são manualmente elaboradas e executadas.

2 VERIFICAÇÃO DE MODELOS

2.1 Introdução

Com o crescente uso de tecnologia da informação por meio de *hardwares* e *softwares*, o esforço de garantir a correção destes vem se tornando cada vez maior. Tanto pelo aumento na complexidade destes sistemas quanto pelo aumento da criticidade de seus usos. Para garantir que os produtos finais respeitem inteiramente as regras aplicáveis, os principais métodos são: simulações, testes, provadores de teorema, ou verificação dedutiva, e a verificação de modelos (CLARKE, 1999).

Simulações e testes de hardware ou software são experimentos realizados sobre seus modelos ou abstrações, no caso de simulações, ou sobre o produto final, no caso de testes. Ambos pretendem estudar as respostas obtidas para uma série de diferentes entradas, buscando determinar a exatidão dos resultados. Tais procedimentos apresentam a limitação de dificilmente conseguirem incluir todas as possibilidades de resultados para todas as possibilidades de entrada, ainda mais no caso de sistemas complexos.

Já os métodos formais são linguagens, técnicas e ferramentas matemáticas usadas para explorar todo o conjunto de relações entrada/saída dos sistemas. Existem duas abordagens principais de métodos formais: os provadores de teoremas e a verificação de modelos.

Os provadores de teoremas, também conhecidos como verificação dedutiva, usam lógica matemática para expressar um sistema e suas propriedades por meio de um conjunto de fórmulas. O procedimento de verificação consiste em encontrar uma prova para essas fórmulas. A verificação por provadores de teoremas não é totalmente automatizável, exigindo a atuação de um especialista em provas matemáticas para a sua realização, sendo a técnica ideal para verificar sistemas modeláveis por um número infinito de estados ou sistemas críticos que exijam uma prova matemática de sua exatidão.

Por sua vez, a verificação de modelos foi idealizada para sistemas que possam ser

representados por um conjunto finito de estados. Ela permite explorar sistemas de estados finitos como um grafo de transição efetuando validações em uma lógica temporal. Por ser executada em sistemas de estado finitos, a automatização do processo de determinação de todos os conjuntos de estado é possível, reduzindo consideravelmente o esforço de verificação se comparado com o método matemático-dedutivo. Além de apresentar se as regras testadas são verdadeiras ou falsas para determinado modelo, as ferramentas de *model checking* apresentam um contra-exemplo quando a propriedade não é valida (SANTOS; FERREIRA; TRIBOLET, 2007).

No grafo de transição de estados que representa o modelo a ser verificado, cada vértice corresponde a um estado do sistema, correspondente aos conjuntos de valores possíveis para cada uma de suas variáveis. Já as arestas correspondem a transições entre os estados. A verificação consiste em percorrer todos os estados do modelo e verificar se o mesmo atende as propriedades a serem testadas.

Formalmente, representa-se o modelo com uma estrutura *Kripke*, que consiste em um conjunto de estados, um conjunto de transições entre estes estados e os conjuntos de propriedades que são verdadeiras (CLARKE, 1999). Por exemplo, em um modelo que possui três variáveis *booleanas* a, b e c, os conjuntos de estado $(a = 1, b = 1, c = 1)$, $(a = 1, b = 0, c = 1)$ e $(a = 1, b = 0, c = 0)$ são possíveis de ocorrer. Para a representação simbólica de tais estados pode-se utilizar o formato $(a, \overline{b}, c)$ ou $(a, \sim b, c)$, na qual b significa que a variável é verdadeira e, $\overline{b}$ ou $\sim b$ significam que a variável é falsa. Na Figura 2 é mostrada uma estrutura *Kripke* que resume o funcionamento de um forno de microondas.

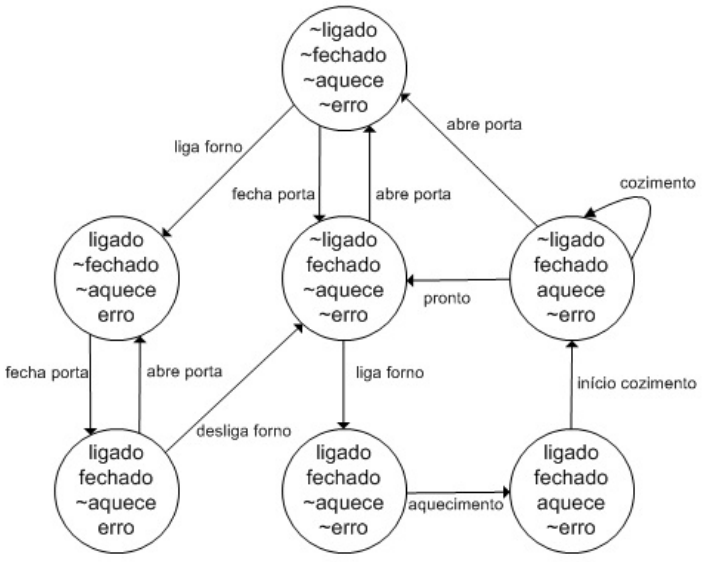

Figura 2: Exemplo de funcionamento de um forno microondas. Fonte: (CLARKE, 1999)

A técnica de *model checking* tem sido amplamente utilizada e evoluída para a verificação formal de *software*. Em Sreemani e Atlee (1996) aplica-se a técnica na verificação da especificação de uma aeronave demonstrando sua viabilidade em especificações não triviais. Assim também acontece em Markosian, Mansouri-Samani e Mehlitz (2006) em que se apresenta um estudo de caso do sistema de gestão de vôos de espaçonaves da NASA. Também em Chan et al. (1998) observa-se a aplicabilidade de sua adoção para sistemas de grande porte.

Em todos os três casos anteriores, a grande limitação da técnica é apresentada e contornada: a ocorrência da explosão do número de variáveis de estado. Esse fenômeno acontece pela característica da técnica em explorar todos os possíveis estados e suas transições, gerando um conjunto de caminhos cujo crescimento é exponencial, criando um limite prático para a avaliação de fluxos de grande porte. Em todos esses casos, a aplicação da técnica demandou a utilização prévia de técnicas de simplificação do conjunto de estados. Godefroid e Huth (2005) demonstra a simplificação semântica de modelos lógico-temporais como forma de reduzir o custo da aplicação da técnica.

O processo de verificação de modelos é organizado em três etapas: modelagem, especificação e verificação. A modelagem trata da conversão do modelo representado pela estrutura Kripke em um código aceito pelo verificador, nos moldes de um compilador. No caso deste trabalho, o código é escrito usando a linguagem SMV para ser executado no verificador NuSMV. A especificação é a definição de regras que serão aplicadas ao modelo e cuja validade será determinada pelo verificador. Usualmente utiliza-se a lógica temporal para a definição das regras, determinando assim comportamentos esperados para o modelo. Finalmente, a terceira etapa é a realização da verificação propriamente dita, na qual os estados possíveis para o modelo são obtidos e a validade da regra é determinada. As próximas seções detalham o método necessário para as etapas de modelagem (pela linguagem SMV) e de especificação (pela lógica temporal).

2.2 A Linguagem SMV

Symbolic Model Verifier (SMV) é uma ferramenta de verificação de especificações descritas em lógica temporal CTL sobre sistemas de estados finitos, utilizando para isso uma linguagem específica. A lógica CTL permite descrever as propriedades temporais de cada elemento do modelo e as condições para as mudanças de estado, proporcionando uma descrição estruturada de uma estrutura *Kripke*.

```
1  MODULE main
2  VAR
3          ligado: boolean;
4          fechado: boolean;
5          aquece: boolean;
6          erro: boolean;
7
8          liga_forno: boolean;
9          desliga_forno: boolean;
10         abre_porta: boolean;
11         fecha_porta: boolean;
12         cozimento: boolean;
13         aquecimento: boolean;
14         inicio_cozimento: boolean;
15         pronto: boolean;
16
17 ASSIGN
18         init(ligado) := 0;
19         init(fechado) := 0;
20         init(aquece) := 0;
21         init(erro) := 0;
22
23         init(liga_forno) := 0;
24         init(desliga_forno) := 0;
25         init(abre_porta) := 0;
26         init(fecha_porta) := 0;
27         init(cozimento) := 0;
28         init(aquecimento) := 0;
29         init(inicio_cozimento) := 0;
30         init(pronto) := 0;
31
32         next(ligado) :=
33                 case
34                         liga_forno = 1 : 1;
35                         desliga_forno = 1: 0;
36                 esac;
37
38         next(fechado) :=
39                 case
40                         abre_porta = 1 : 0;
41                         fecha_porta = 1 : 1;
42                 esac;
```

Código 2.1: Parte da tradução do funcionamento do microondas

O Código 2.1 expressa parte do modelo contido na Figura 2 na linguagem SMV.

O exemplo descreve um modelo e uma especificação em lógica CTL. Os estados do modelo são definidos por uma coleção de variáveis, que podem ser *booleanas* ou do tipo escalar. As variáveis são declaradas logo após a instrução VAR. Neste exemplo, cada uma das quatro variáveis assim como as transições entre os estados são definidas como variáveis booleanas.

As relações de transição da estrutura *Kripke* e os estados iniciais são determinadas por um conjunto de atribuições paralelas, que são codificadas após a instrução $ASSIGN$. A instrução *init* inicializa uma variável.

O valor da expressão *case* é determinado pela primeira expressão do lado direito dos dois pontos (:) desde que a condição do lado esquerdo seja verdadeira. Portanto, para

a variável *ligado*, se a propriedade $liga\_forno$ for verdadeira, então essa também será verdadeira. Caso a propriedade $desliga\_forno$ for verdadeira, então a variável *ligado* será falsa. Tais condições são avaliadas na ordem de definição, de tal forma que a segunda condição somente será avaliada e processada caso a primeira seja falsa, e assim por diante.

A propriedade a ser verificada no modelo aparece como uma formula CTL, como será descrito na próxima seção, definida pela instrução $SPEC$. O verificador irá percorrer todos os estados alcançáveis verificando se a propriedade é satisfeita ou não.

2.3 Lógica temporal

Lógica temporal é um formalismo utilizado para especificar a ordenação temporal de eventos em sistemas concorrentes, e que também pode ser usada para descrever uma sequência temporal de transições entre estados. Esta passou a ser combinada com a verificação de modelos pelo trabalho de alguns autores (Clarke, Emerson, Quielle e Sifakis), tendo sido possibilitada também a automatização desta combinação (CLARKE, 1999).

Se destacam neste tema as lógicas CTL (*Computer Tree Logic*) e LTL (*Linear Temporal Logic*). A lógica LTL possui somente operadores de lógica linear enquanto CTL combina tanto operadores de ramificação quanto lineares. Ambas são subconjuntos da lógica CTL*, a qual descreve propriedades de árvores computacionais.

CTL é uma das lógicas que podem ser utilizadas para expressar propriedades a serem validadas por um verificador de modelos, tendo se mostrado como a mais adequada ao propósito do trabalho. As árvores de computação utilizadas nessa lógica são derivadas do grafo de transição de estados definidos pela estrutura *Kripke*. O grafo de transição de estados é transformado em uma árvore infinita cuja raiz é o estado inicial do grafo. Veja um exemplo na Figura 3. Os caminhos na árvore de computação representam todas as computações possíveis no modelo.

A linguagem para CTL possui operadores que podem ser aplicados sobre os caminhos de uma árvore de computação. Em uma fórmula CTL os operadores devem aparecer aos pares e na seguinte ordem: um quantificador de caminho seguido de um operador temporal. O quantificador de caminho define o escopo de caminho em que a fórmula f deve ser verdadeira. Existem dois quantificadores de caminho: **A** que significa "para todos os caminhos"; e **E** que significa "existe um caminho qualquer". Os operadores temporais, por sua vez, definem o comportamento temporal que deve ocorrer ao longo do caminho relacionado à fórmula x. Os operadores temporais são:

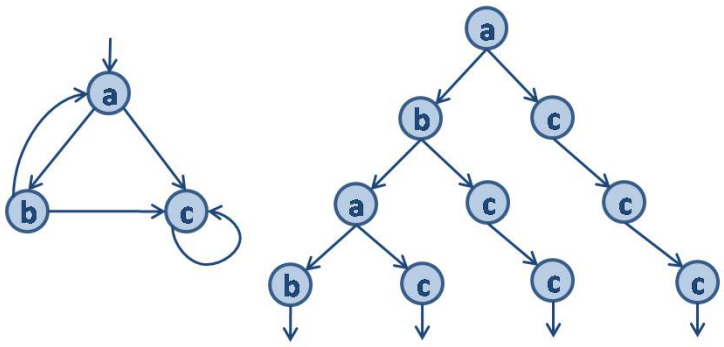

Figura 3: Grafo de transição de estados e sua respectiva árvore de computação.

- **F** (no futuro) - iniciando da raiz da árvore de computação, x deve ser verdadeira em algum estado do caminho;
- **G** (globalmente) - iniciando da raiz da árvore de computação, x deve ser verdadeira em todos os estados do caminho;
- **X** (no próximo estado) - iniciando da raiz da árvore de computação, x deve ser verdadeira no segundo estado do caminho.
- **U** (até, *until*) - significa que existe um estado a no caminho onde a fórmula y é válida, e que em todos os estados anteriores a a, x é valido;
- **R** (libera, *release*) - iniciando da raiz da árvore de computação, x deve ser verdadeira em todos os estados do caminho até que a fórmula y seja verdadeira, quando então x pode deixar de ser verdadeira. Entretanto, y pode nunca ocorrer;

No exemplo da Figura 2, uma verificação passível de ser feita é saber se em todas as vezes que o forno for ligado, o mesmo irá aquecer. Na lógica CTL esta validação é traduzida por: $AG(ligado \Rightarrow AF(aquece))$. O operador inicial AG indica que a propriedade será avaliada em todos os caminhos da árvore (A) e em todos os estados de cada caminho (G). O operador AF indica que a propriedade *aquece* será avaliada se em todos os caminhos (A) e em um estado futuro qualquer (F) tal propriedade é válida. Dessa forma, a tradução completa seria: em todos os caminhos e em todos os estados a partir do qual a propriedade *ligado* for verdadeira implicará que em todos os caminhos, no futuro, haverá ao menos um estado em que a propriedade *aquece* é verdadeira.

Outra validação de exemplo é a avaliação se existe alguma sequência de execução em que o forno esteja ligado e aquecendo com a porta aberta. Nesse caso, a expressão a ser usada seria: $EF(ligado \ \& \sim fechado \ \& \ aquece)$. Detalhadamente, essa expressão pode

ser traduzida por: existe um estado em que, no futuro, a propriedade *ligado* é verdadeira, a propriedade $fechado$ é falsa e a propriedade *aquece* é verdadeira.

3 MODELAGEM DE NEGÓCIOS

3.1 Fluxos de trabalho

Processos de negócio, também conhecidos como fluxos de trabalho ou *workflows*, segundo Aguilar-Savén (2004) são "combinações de conjuntos de atividades em uma organização com uma estrutura que descreve sua ordem lógica e dependências, cujo objetivo é a produção de um determinado resultado".

Na última década é possível perceber um maior interesse das instituições na adoção de técnicas de modelagem de processos de negócio como forma de ampliar a eficácia e eficiência de fluxos de trabalhos corporativos e até científicos, sendo estes apoiados ou não por sistemas de informação (ANDRADE et al., 2004) (CONERY; CATCHEN; LYNCH, 2005). A construção de sistemas informatizados como apoio à realização de fluxos de trabalho inadequados, não otimizados ou não aderentes à modelagem desenvolvida releva-se um fator de desperdício de recursos e um problema de difícil detecção e solução.

No contexto empresarial atual de grande competição e informatização, esse cenário se torna ainda mais impactante no sucesso e sobrevivência de empreendimentos. Baker (2001) deixa claro que, quando os requisitos de negócio são definidos de maneira inadequada, as inconsistências no nível de negócio se propagam e se ampliam dentro dos sistemas automatizados. Segundo ele, a modelagem de negócios é um conjunto de atividades para a visualização e entendimento de processos de negócio e é um passo importante para a construção de sistemas. Keen e Qureshi (2006) comparam a eficácia da utilização de modelagem de negócio com métodos mais tradicionais de formalização de estratégias corporativas por meio de estudos de casos e conclui sobre a capacidade desse instrumento em transformar as estruturas organizacionais em favor dos objetivos de negócio.

A modelagem de negócios pode ser formalizada com modelos/desenhos que identifiquem atividades, informações e o fluxo destes sob a forma de processos de produção ou de gestão (AN; JENG, 2005). Para isso, existem padrões de notação de fluxos de trabalho que são padrões gráficos de desenho, associando formas com tipos de elemento (TSAI;

LUO; WANG, 2007). Duas são as notações mais utilizadas atualmente para este fim: *Unified Modeling Language* (UML) e *Business Process Modelling Notation* (BPMN) (WHITE, 2004).

A linguagem de notação UML tornou-se nas últimas décadas o padrão em notação para especificação, análise e desenvolvimento de *software*, incluindo também o conjunto de notação para a modelagem de negócio (ERIKSSON; PENKER, 2000). Porém, alguns trabalhos tratam da inadequação do diagrama de estados para aplicações de *workflows* que necessitam de um maior formalismo e maior suporte à modelagem utilizando recursos ou aspectos organizacionais (GUELFI; MAMMAR, 2006) e (RUSSEL et al., 2006). Embora o trabalho de Eshuis e Wieringa (2001) trate da criação de um formalismo ao diagrama de atividades da UML, adequado ao objetivo de verificação por *model checking*, esta adoção representaria um esforço extra de conformidade e análise ao presente trabalho.

Já a notação BPMN[1], por se especializar na diagramação exclusiva de processos de negócio, tem tido considerável aceitabilidade no mercado (OMG, 2009).

Para que a notação seja formalizada em um formato executável por software, é necessário transformar o desenho em uma especificação de processos, utilizando para isso linguagens de especificação de processos (JIANG; MAIR; NEWMAN, 2003). Vários são os padrões existentes no momento: PSL, DAML-S, RFP, XPDL, BPML e BPEL. Dentre estes, o padrão XPDL[2] - XML Process Definition Language - vem sendo utilizado em diversos trabalhos sobre o tema. Esta define uma semântica formal e padronizada para a representação das tarefas, suas transições, os recursos necessários, atores envolvidos, etc. O propósito principal da linguagem é servir como padrão para a interoperabilidade de sistemas que utilizam os vários padrões citados (HALLER; GAALOUL; MATEUSZ, 2008).

Por meio da modelagem de negócios devem ficar claras as regras de negócio que definem e detalham a execução das atividades de um processo. Segundo Zsifkov e Campeanu (2004), as regras de negócio corporativas são definidas como restrições ou metadados relativos às operações de negócio. Tais regras podem se originar da própria lógica do processo, de métodos científicos (no caso de pesquisas acadêmicas, por exemplo), ou também de legislações e normatizações (MUEHLEN; INDULSKA; KAMP, 2007). Em sistemas de informação tais regras podem estar definidas dentro das notações de modelagem de negócio ou em linguagens próprias.

[1] A notação BPMN foi criada pelo grupo OMG (Object Management Group) com o objetivo de se tornar o padrão específico para a representação gráfica de fluxos de trabalho.

[2] Esta linguagem foi criada pela WfMC - *Workflow Management Coalition* (WFMC, 2008). Tanto a OMG quanto a WfMC são entidades que congregam diversas empresas e pesquisadores especializados no assunto.

Em ambientes com processos de negócio de ampla complexidade, tais como hospitais onde sistemas *real-time* controlam vidas humanas, processos de negócio que sejam inadequados podem produzir sérias consequências. Nesses casos a verificação das regras se torna mais relevante (SANTOS; FERREIRA; TRIBOLET, 2007). Verificar regras de negócio seria então a atividade de avaliação da conformidade das mesmas em relação à realidade dos negócios. Assim como a verificação de artefatos de *software* representa um ganho de produtividade no desenvolvimento dos mesmos (CHAN et al., 1998), a verificação das regras de negócio durante a modelagem dos processos de negócio contribuiria para a formalização de regras mais aderentes à realidade da ação de uma corporação. Desse modo, caso os processos de negócio venham a ser alvos de automatização e apoio prestado por sistemas informatizados, as regras que devem ser respeitadas estarão em um nível superior de maturidade e adequação.

3.2 XPDL

A linguagem XPDL tem como objetivo formalizar a especificação de fluxos de trabalho (MATOUSEK, 2003). Ela traduz a definição feita por meio de diversas padrões de notação (desenhos) como, por exemplo, o BPMN ou até a UML. As definições são escritas no formato XML e obedecem o padrão da linguagem definida pelo consórcio de empresas *Workflow Management Coalition* (WfMC) (WFMC, 2008).

Os itens que compõem o padrão e que são utilizados neste trabalho são as definições de processo, as atividades e as transições. Outros componentes do padrão não foram considerados por não afetarem o resultado da análise proposta, tais como os elementos participantes, aplicações, artefatos, entre outros.

A atividade é o formato utilizado para representar ações ou tarefas dentro de um fluxo de trabalho. No caso do XPDL, estas representam também os objetos de decisão, pelos quais o fluxo pode ser desviado dependendo de uma determinada condição. O Código 3.1 exemplifica uma especificação XPDL contendo uma atividade comum e uma atividade que especifica uma divisão no fluxo. O que diferencia tais tipos é a declaração que pode ser vista na linha 20 do código de exemplo, que especifica que a atividade de nome "Disponível?" é um objeto de decisão e não uma atividade pois ele é um objeto *Route*. Já a atividade "Efetuar reserva", cuja especificação inicia-se na linha 1 do código, é uma atividade comum.

Existem quatro tipos de modificadores de fluxo *Route*: exclusivos (ou *XOR*),

```
1  <Activity Id="_2JuKGpgZEd6Y0shy6Im_cw" Name="Efetuar reserva">
2          <Implementation>
3                  <No/>
4          </Implementation>
5          <ExtendedAttributes>
6                  <ExtendedAttribute Name="SplitSimulationData">
7                          <simulation:SplitSimulationData>
8                                  <simulation:ParameterDeterminedSplit>true</
                                       simulation:ParameterDeterminedSplit>
9                                  <simulation:SplitParameter ParameterId=""/>
10                         </simulation:SplitSimulationData>
11                 </ExtendedAttribute>
12         </ExtendedAttributes>
13         <NodeGraphicsInfos>
14                 <NodeGraphicsInfo BorderColor="0,0,128" FillColor="255,219,74" Height="
                       64.0" LaneId="_7cGfIJgYEd6Y0shy6Im_cw" ToolId="XPD" Width="96.0">
15                     <Coordinates XCoordinate="339.0" YCoordinate="74.0"/>
16                 </NodeGraphicsInfo>
17         </NodeGraphicsInfos>
18 </Activity>
19 <Activity Id="_njyBkJogEd6zUa-M_oKdSA" Name="Disponível?">
20         <Route GatewayType="XOR"/>
21         <ExtendedAttributes>
22                 <ExtendedAttribute Name="SplitSimulationData">
23                         <simulation:SplitSimulationData>
24                                 <simulation:ParameterDeterminedSplit>true</
                                       simulation:ParameterDeterminedSplit>
25                                 <simulation:SplitParameter ParameterId=""/>
26                         </simulation:SplitSimulationData>
27                 </ExtendedAttribute>
28         </ExtendedAttributes>
29         <NodeGraphicsInfos>
30                 <NodeGraphicsInfo BorderColor="0,0,128" FillColor="255,219,74" Height="
                       45.0" LaneId="_7cGfIJgYEd6Y0shy6Im_cw" ToolId="XPD" Width="43.0">
31                     <Coordinates XCoordinate="302.0" YCoordinate="179.0"/>
32                 </NodeGraphicsInfo>
33         </NodeGraphicsInfos>
34 </Activity>
```

Código 3.1: Fragmento de código em XPDL (exemplo de definição de objeto de decisão)

inclusivos (ou *OR*), paralelos (ou *AND*) e complexos. Apesar das definições *XOR*, *OR* e *AND* estarem obsoletas na especificação atual do XPDL, elas ainda são usadas para alguns softwares de modelagem. Tais definições foram alteradas para *Exclusive*, *Inclusive* e *Parallel*, respectivamente.

A mesma linha 20 do exemplo (Código 3.1) define que esta atividade é um objeto *Route* do tipo *XOR*. Ou seja, necessariamente o fluxo deverá tomar somente uma transição a partir dessa atividade, podendo haver ou não a confluência dos caminhos exclusivos. Além dos objetos do tipo *XOR* pode-se ter também os tipos *AND* e *OR*. No caso de *routes* do tipo *AND*, eles indicam a divisão do fluxo em dois ou mais caminhos paralelos que devem ser executados até serem unidos por um novo objeto do mesmo tipo. Em outras palavras, ambos os caminhos devem ser executados e concluídos para que o fluxo siga a sua execução. No caso de *routes* do tipo *OR*, o fluxo é dividido em caminhos que podem ou não serem executados paralelamente. Porém essa definição também necessita que os caminhos atinjam um mesmo *route* do tipo *OR* para indicar o final dessa divisão.

No caso de divisão do fluxo por meio de objetos de decisão ou parelelismo, configura-se como boa prática a utilização de objetos do tipo *join* (junção) para a reunificação de caminhos exclusivos ou paralelos. Tais elementos definem os requisitos para a continuidade, podendo ser especificado, por exemplo, que todos os caminhos de uma parelelismo devem ser executados para que se dê continuidade na execução do fluxo (WFMC, 2008). Caso tal estratégia não seja seguida, adota-se como padrão para a junção dos fluxos o tipo exclusivo (XOR) para a divisão por decisão e o tipo paralelo (AND) para o paralelismo.

As transições indicam a ordem de execução do fluxo, determinando quais são as atividades atingíveis a partir de uma determinada atividade. Nas notações, elas são indicadas frequentemente por setas saindo da atividade anterior e chegando nas próximas. No padrão XPDL pode haver transições de sequência, fluxos de mensagens ou associações entre objetos. Somente os primeiros foram considerados neste trabalho por representarem efetivamente os caminhos de execução.

Outro componente da linguagem utilizado neste trabalho é a divisão de fluxos em subfluxos. Neste caso, uma atividade pode ser detalhada em um novo subfluxo, ainda pertencente ao fluxo original. Ao invés de uma execução pontual de uma atividade, a execução de um subfluxo obedece às mesmas regras de um fluxo comum. Dessa maneira é possível criar um fluxo simplificado e detalhar cada atividade em subfluxos, criando uma árvore de execução, sem limites de níveis de detalhamento.

```
1 <Activity Id="_j2Yx8NQ4Ed6XPO7ah7EKDA" Name="Receber e distribuir PA">
2         <Implementation>
3                 <SubFlow Id="_YEo60NQ7Ed66BtkiytxsJQ"/>
4         </Implementation>
```

Código 3.2: Fragmento de código em XPDL (exemplo de definição de subfluxo)

No caso da linguagem XPDL, tanto o fluxo de maior abstração quanto os subfluxos de maior nível de detalhamento são mantidos em um mesmo arquivo, organizado de tal maneira a descrever a relação entre uma atividade em um fluxo que é detalhado em um subfluxo. Para expressar essa relação, inclui-se o código do subfluxo na opção *Implementation* da definição da atividade a ser detalhada, conforme o exemplo (Código 3.2).

4 ABORDAGEM PROPOSTA

4.1 Elementos avaliados

Na geração do modelo a ser validado pelo verificador NuSMV, cada atividade, nós iniciais e nós finais são traduzidos em variáveis booleanas que representam se tal atividade foi executada ou não. Todas as variáveis iniciam-se como não executado, com exceção do nó inicial que já assume o estado de executado. Após a declaração das variáveis e suas inicializações, são definidas para cada um dos elementos as regras de execução. Para a tradução dos comportamentos de transição das atividades do fluxo de trabalho, identificou-se alguns casos específicos.

4.1.1 Transições simples

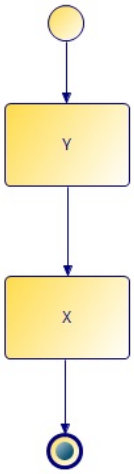

Figura 4: Diagrama de exemplo de uma sequência simples de execução

O primeiro caso traduz as ocorrências mais simples, na qual uma atividade só recebe ligações simples vindas de atividades. Para este caso, a única condição para a execução da atividade avaliada é a execução das atividades anteriores. Se, por exemplo, a execução de uma atividade X depender exclusivamente da execução de uma atividade

Y (Figura 4), então a regra a ser gerada é descrita no Código 4.1.

Ou seja, se a atividade X já foi executada, ela continua com o valor verdadeiro. Se a atividade Y já foi executada mas a X ainda não, então ela vai assumir nessa transição o valor verdadeiro. Em todos os outros casos, ela mantém o valor falso.

```
1 next(atividadeX) :=
2         case
3                 atividadeX = 1 : 1;
4                 atividadeX = 0 & atividadeY = 1 : 1;
5                 1: 0;
6         esac;
```

Código 4.1: Fragmento da tradução de uma atividade exclusivamente dependente de outra

O *software* trata a ocorrência de atividade que recebem mais de uma ligação sem a utilização de objetos de junção. Apesar desta ocorrência não ser considerada uma boa prática, a sua adoção é avaliada para aumentar a utilidade do mesmo. Por padrão, conforme dito anteriormente, tal junção é considerada como do tipo OR.

4.1.2 Nós iniciais e finais

Quando o tradutor recebe como parâmetro o nó inicial, este percorre o nó e os elementos imediatamente subsequentes. Se forem do tipo atividade, estes sempre serão executados de forma imediata pois só dependem do nó inicial que já foi executado.

Tal comportamento pode ser expresso no modelo lógico-temporal do Código 4.2. Neste define-se que a cada nova transição do modelo, a atividade de índice zero sempre estará como executada pois só depende da execução do nó inicial.

```
1 next(atividade0) := 1;
```

Código 4.2: Fragmento da tradução de um nó inicial

É importante diferenciar o nó inicial do fluxo geral mais abstrato dos nós iniciais contidos em subfluxos. Como dito anteriormente, os subfluxos em XPDL são declarados no mesmo arquivo do fluxo principal. O *software* incorpora os subfluxos ao fluxo principal no lugar das atividades que os encapsulavam, de forma similar à expansão de funções *inline* em algoritmos. Dessa forma, um fluxo detalhado em vários subfluxos em múltiplos níveis é transformado em um fluxo único. Assim, é esperado encontrar vários nós iniciais em modelos que contenham subfluxos, sendo um para cada subfluxo. Tais nós iniciais foram considerados meros conectores entre elementos do fluxo principal e os subfluxos — ao

contrário do nó inicial principal que possui características específicas. O mesmo acontece para os nós finais de subfluxos.

Este caso trata também dos nós finais do fluxo mais abstrato. Para efeitos de tradução, quando tais nós são testados, percorrem-se as atividades que se ligam nestes para determinar as condições de sua execução. Se ele for ligado por uma única atividade, a condição de execução do nó final é simplesmente a execução desta atividade. Se o nó final é ligado por mais de uma atividade, segundo a definição da linguagem XPDL, entende-se que se alguma delas for executada, o nó final também o será. Ou seja, quando atividades recebem diretamente ligações de outras atividades, sem a utilização de elementos de junção de caminhos, subentende-se que é uma junção do tipo OR.

4.1.3 Objetos de decisão/condicionais

Quando os elementos avaliados estão envolvidos com condicionais (Figura 5), é necessário implementar no modelo a ser verificado a característica de indeterminismo quanto à escolha dos caminhos que serão percorridos no fluxo de trabalho. Ou seja, o verificador deverá avaliar a validade das afirmativas lógico-temporais considerando todos os caminhos a partir de tal divisão do fluxo.

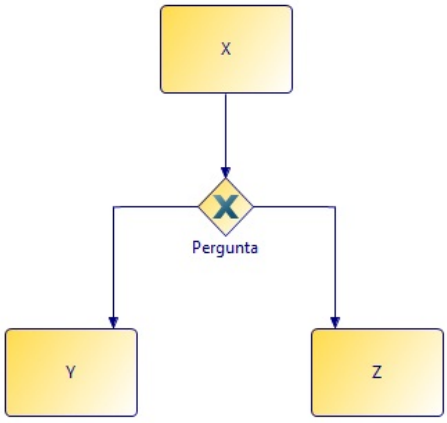

Figura 5: Diagrama de exemplo de um objeto de decisão

O indeterminismo é gerado no modelo conforme o Código 4.3. Neste exemplo é demonstrada a tradução para o diagrama contido na Figura 5. Pode-se notar a adoção de regras para garantir a permanência de valores atribuídos anteriormente e o indeterminismo sendo indicado para os demais casos (linha 5).

Um fator de diferenciação das regras de execução do fluxo é a existência de múltiplos objetos de decisão interligados, como demonstrado na Figura 6. Para a avaliação

```
next(atividadeY) :=
        case
                atividadeY=1: 1;
                atividadeX=1 & atividadeZ=1: 0;
                atividadeX=1: { 0, 1 };
                1: 0;
        esac;

next(atividadeZ) :=
        case
                atividadeZ=1: 1;
                atividadeX=1 & (atividadeY=1 | next(atividadeY=1)): 0;
                atividadeX=1 & atividadeY=0: 1;
                1: 0;
        esac;
```

Código 4.3: Fragmento da tradução de um objeto de decisão (exemplo de indeterminismo)

da sequência de execução de um fluxo é imperativo determinar as condições de execução, que em outras palavras define-se como as atividades anteriores que precisam ter sido executadas. No caso de múltiplos objetos de decisão essa determinação é mais complexa se comparada com os casos em que o objeto de decisão está isolado entre atividades, sendo necessário determinar as interferências entre cada elemento envolvido.

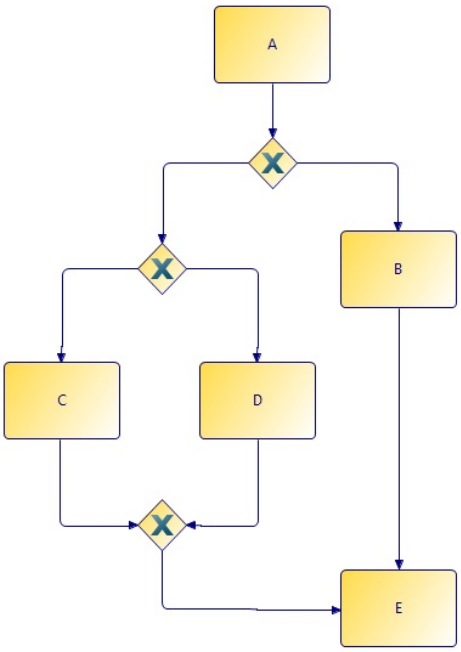

Figura 6: Diagrama de exemplo de decisões interligadas

O código 4.4 apresenta a tradução para este tipo de ocorrência.

Também age como fator de diferenciação do comportamento do fluxo a existência de desvios de execução (Figura 7), quando um dos caminhos se liga a alguma das outras atividades imediatamente posteriores à divisão do fluxo. Tal ocorrência necessita de ava-

```
next(atividadeB) :=
        case
                atividadeB=1: 1;
                atividadeA=1 & atividadeB=0 & (atividadeC=0 & atividadeD=0): {0, 1};
                1: 0;
        esac;

next(atividadeC) :=
        case
                atividadeC=1: 1;
                atividadeA=1 & (atividadeB=1 | next(atividadeB=1)): 0;
                atividadeA=1 & atividadeC=0 & atividadeD=0: {0, 1};
                1: 0;
        esac;

next(atividadeD) :=
        case
                atividadeD=1: 1;
                atividadeA=1 & (atividadeB=1 | next(atividadeB=1) | atividadeC=1 | next(
                    atividadeC=1)): 0;
                atividadeA=1 & atividadeB=0 & atividadeC=0 & atividadeD=0: 1;
                1: 0;
        esac;

next(atividadeE) :=
        case
                atividadeE=1: 1;
                (atividadeB=1 | atividadeC=1 | atividadeD=1) : 1;
                1: 0;
        esac;
```

Código 4.4: Fragmento da tradução de decisões interligadas

liação específica pois afeta de maneira singular as condições de execução da atividade que foi ligada pelo desvio, como pode ser observado no Código 4.5.

```
next(atividadeY) :=
        case
                atividadeY=1: 1;
                atividadeW=1: 1;
                atividadeX=1 & atividadeY=0 & atividadeZ=0: {0, 1};
                1: 0;
        esac;

next(atividadeZ) :=
        case
                atividadeZ=1: 1;
                atividadeX=1 & (atividadeY=1 | next(atividadeY=1)): 0;
                atividadeX=1 & atividadeY=0: 1;
                1: 0;
        esac;

next(atividadeW) :=
        case
                atividadeW=1: 1;
                atividadeZ=1: 1;
                1: 0;
        esac;
```

Código 4.5: Fragmento da tradução de um desvio de execução

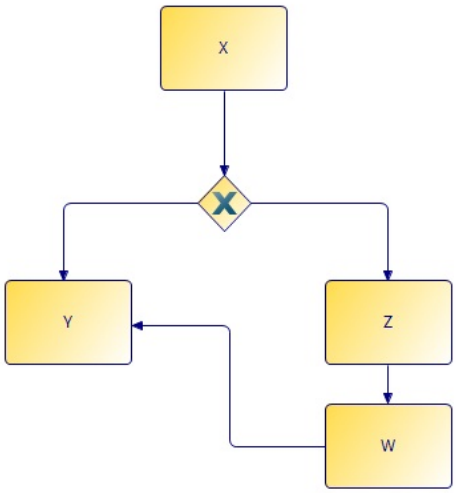

Figura 7: Diagrama de exemplo de desvio de execução

4.1.4 Paralelismo

Este caso se refere ao fenômeno de multiplicação do fluxo decorrente de objetos de paralelismo de execução. O paralelismo, conforme demonstrado na Figura 8, é imposto como no Código 4.6.

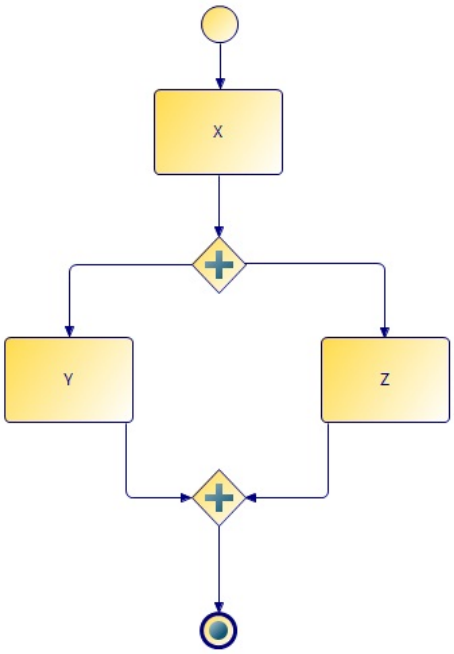

Figura 8: Diagrama de exemplo de paralelismo com junção ao final

Este caso trata tanto das atividades iniciais do paralelismo quanto das atividades em que o paralelismo se encerra. Tais ocorrências são as únicas dependentes de um

tratamento específico, sendo que os elementos intermediários dos caminhos paralelos são tratados de maneira genérica, como descrito anteriormente.

```
1  next(atividadeY) :=
2          case
3                  atividadeX=1: 1;
4                  1: 0;
5          esac;
6
7  next(atividadeZ) :=
8          case
9                  atividadeX=1: 1;
10                 1: 0;
11         esac;
```

Código 4.6: Fragmento da tradução de um objeto de paralelismo

Na ocorrência do objeto de junção dos caminhos paralelos, o encerramento do paralelismo deve ser indicado explicitamente, conforme recomendação de boa prática na modelagem de negócio (OMG, 2009). No entanto também é aceita a omissão de tal elemento nos casos em que o padrão de junção é do tipo *AND*. Neste caso, a atividade seguinte ao final do paralelismo é executada somente quando todos os caminhos paralelos forem encerrados. Da mesma forma, para junções de objeto de decisão aplica-se o padrão de junção do tipo OR quando não explicitamente declarado.

A abordagem proposta neste trabalho apresenta uma limitação quanto a capacidade de interpretação e tradução de elementos do tipo complexo para a divisão, multiplicação e junção de caminhos. Tal tipo de elemento define condições específicas para sua execução. Devido a esta característica de customização, o processo de tradução se mostrou bastante complexo, inviabilizando sua incorporação no trabalho corrente. Os elementos de junção não são traduzidos, assim como os elementos de condicional e paralelismo, mas alteram as condições de execução dos elementos subsequentes. Os tipos de junção, dessa forma, impactam nas regras de execução das tarefas seguintes a tal elemento.

4.2 Validações

Dentre as validações identificadas, verificou-se que algumas validações presentes em outros trabalhos com o mesmo tema não seriam incorporáveis à abordagem apresentada. Uma validação presente em Matousek (2003) não é passível de incorporação na estruturação proposta da abordagem: avaliação se uma determinada atividade não será executada mais de n vezes. Tal limitação se deve ao fato de não ter sido incorporado à abordagem o controle de quantidade de execuções para cada atividade.

As propriedades do tipo *soundness*, como identificada em Huang (2006), são usadas para verificar se todas as atividades iniciadas são finalizadas. A verificação de tal tipo de propriedade não é passível de incorporação na abordagem pelo fato de não ser controlado se uma atividade é iniciada e terminada, somente se foi executada ou não em determinado momento.

Ocorrências de *deadlock*, como definidas em vários trabalhos, são detectáveis pela abordagem de maneira indireta. Ao verificar se sempre o fluxo consegue atingir o final do fluxo, caso não exista ciclos sem limite de execução (como será descrito posteriormente), indica a existência de *deadlocks*. *Livelocks* não são incorporáveis à abordagem por não existir o controle de recursos dentro dos fluxos, somente a execução das atividades.

Sete tipos de validações foram incorporadas à abordagem por serem identificadas como frequentemente utilizadas para avaliar fluxos de trabalho. Tais validações serão apresentadas a seguir.

4.2.1 A execução de uma atividade ou decisão implica na execução de outra

O diagrama da Figura 9 representa a transição de estados necessária para a validade deste tipo de validação. Considerando que a variável X representa a primeira atividade escolhida e Y a segunda, o status 1 representa a execução da atividade enquanto 0 representa que esta não foi executada. Dessa forma, no estado inicial, indicado por um arco superior, ambas as atividades não foram executadas. Na sequência, a atividade X deve ser executada enquanto a Y permanece não executada. Posteriormente, em algum momento, o estado em que ambas as atividades sejam executadas deve ser alcançado e não mais abandonado até o final do fluxo.

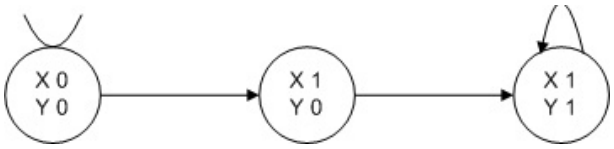

Figura 9: Diagrama "A execução de uma atividade ou decisão implica na execução de outra"

Para avaliar este tipo de relação, usa-se o seguinte comando, no qual x e y representam componentes do fluxo.

```
SPEC AG(atividadex -> AF(atividadey));
```

Tal comando pode ser traduzido como: "em todos os estados do modelo sempre que a atividade x for executada vai implicar que em todos os caminhos, no futuro, a atividade

y vai ser executada". Como exemplo, pode-se pensar em um processo corporativo de compras e na validação da seguinte regra: sempre que a compra for aprovada por um supervisor, a compra deverá ser realizada. Caso haja um encaminhamento em que a compra não seja realizada, a resposta será negativa.

4.2.2 A execução de uma atividade ou decisão implica na não execução de outra

Figura 10: Diagrama "A execução de uma atividade ou decisão implica na não execução de outra"

Para avaliar este tipo de relação (Figura 10), usa-se o seguinte comando:

```
SPEC AG(atividadex -> AG(!atividadey));
```

Ele pode ser traduzido como: "em todos os estados do modelo sempre que a atividade x for executada vai implicar que em todos os caminhos, no futuro, a atividade y não vai ser executada". Se tomado o mesmo exemplo anterior, poderia ser verificado se sempre que o supervisor reprovar uma compra, ela nunca será realizada. Caso haja um encaminhamento em que a compra possa ser realizada, a resposta será negativa.

4.2.3 A execução de uma atividade ou decisão significa a possibilidade da execução de outra

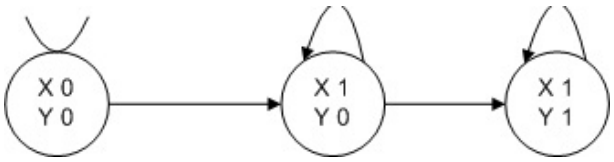

Figura 11: Diagrama "A execução de uma atividade ou decisão significa a possibilidade da execução de outra"

Para avaliar tal relação (Figura 11), usa-se o comando seguinte.

```
SPEC EF((atividadex -> EF(atividadey)) & (E[!atividadey U atividadex]));
```

Ele pode ser traduzido como: "se a atividade x for executada, existirá ao menos um caminho em que no futuro a atividade y também será, sendo que a atividade y não

pode ter sido executada antes da atividade x". Se mantivermos o exemplo de compras corporativas, uma regra que poderia ser testada por esta relação seria a verificação se uma compra, mesmo que reprovada por uma auditoria, seja concretizada. Tal hipótese aconteceria se, por exemplo, o fluxo tiver contemplado a possibilidade de a auditoria rever o processo e aprová-lo. Se no fluxo estiver definido que sempre que a auditoria reprovar um processo este deverá ser cancelado, então a resposta para a regra seria negativa.

4.2.4 A execução de uma atividade ou decisão depende da prévia execução de outra

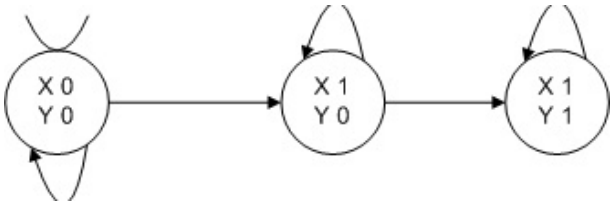

Figura 12: Diagrama "A execução de uma atividade ou decisão depende da prévia execução de outra"

Para esta regra (Figura 11), o comando completo desta validação seria:

```
SPEC !EF(!atividadex & atividadey) & !EF(!atividadex & !atividadey & EX(atividadex & atividadey));
```

Esta avaliação é feita em duas etapas. Primeiro pesquisa-se se não existe um caminho em que no futuro a atividade x (anterior) não é executada e a y (posterior) é. A segunda parte pesquisa se não existe um caminho no qual ambas atividades são executadas ao mesmo tempo. Com as duas verificações garante-se que a segunda atividade só é executada se a anterior também é e se elas não são executadas ao mesmo tempo. No exemplo do processo de compras, pode-se testar se tarefas que estejam modeladas como sendo de execução paralela, na verdade deveriam ser sequenciais. Se no fluxo for possível fazer a solicitação de um produto e em seguida o pagamento. Porém se este pagamento for dependente de uma tarefa sendo executada paralelamente, por exemplo a formalização do pedido, então esta última tarefa deve ser executada antes do pagamento. Ou seja, o pagamento não poderá ser paralelo à formalização do pedido, somente a solicitação.

4.2.5 É possível que uma determinada atividade não seja executada

Para avaliar este tipo de relação, demonstrada na Figura 13, usa-se o seguinte comando.

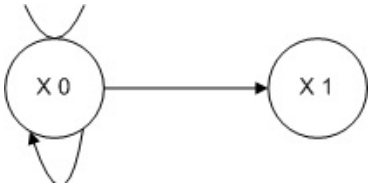

Figura 13: Diagrama "É possível que uma determinada atividade ou decisão não seja executada"

```
SPEC !EG( !atividadex );
```

O comando é traduzível por: "existe um caminho no qual a atividade x não é executada". No processo de compras, pode-se testar se existe a possibilidade de que a tarefa de avaliação do superior não seja executada, se por exemplo a compra for de valor pequeno.

4.2.6 Uma determinada atividade nunca é executada

Figura 14: Diagrama "Uma determinada atividade ou decisão nunca é executada"

O seguinte comando é usado para avaliar este tipo de relação, mostrado na Figura 14.

```
SPEC AG( !atividadex );
```

Por meio do comando pesquisa-se pela validade da afirmativa de que, em todos os caminhos possíveis, a atividade x não é executada. Em outras palavras, verifica-se se o elemento pesquisado possui grau zero de entrada, não possuindo transições que apontem para ele. A utilidade desta verificação diz respeito mais à verificação de erros de modelagem do que a verificação de erros de lógica de negócio.

4.2.7 É sempre possível atingir o final do fluxo

Para avaliar a regra usa-se o seguinte comando:

```
SPEC AG(atividadea -> AF(atividadeb));
```

Neste comando a atividade a é entendida como o nó inicial do fluxo abstrato e o b o

nó final geral. Ou seja, o comando é traduzível por: "sempre que o nó inicial for executado o nó final também será". Três ocorrências são capazes de gerar uma resposta negativa para esta regra. A primeira é a inexistência de nó final para o fluxo principal. Se por exemplo for testado um fluxo parcialmente modelado, não será sequer possível executar essa validação por não ser possível determinar qual é o nó final. A segunda possibilidade é a existência de caminhos com elementos desconexos, que não possibilitam que o caminho atinja o nó final. Finalmente, a existência de ciclos sem a especificação correta de condição de interrupção gera a possibilidade de um caminho que sempre permanece dentro do ciclo.

4.3 Validação do modelo

A tradução se encerra quando todos os caminhos presentes no fluxo são percorridos pelo caminhamento por profundidade. Em um fluxo corretamente modelado espera-se que os caminhos se encerrem em nós finais, mas esta não é uma condição essencial para o correto funcionamento do *software*. Tal flexibilidade permite que modelagens incompletas sejam avaliadas, obtendo resultados parciais semelhantes aos de avaliações de modelagens finalizadas.

Quando a tradução das regras de execução dos elementos do fluxo de trabalho é encerrada, o *software* gera a afirmativa lógico-temporal em linguagem CTL que será avaliada pelo verificador de modelos. Se o retorno do verificador é de que a afirmativa é válida, tal mensagem é exibida pela interface para o usuário. Caso seja falsa, o verificador exibe um contraexemplo que mostra a razão da não validade da afirmativa. Esse contraexemplo é disponibilizado em uma caixa de texto na interface para que este possa avaliar o resultado.

Apesar do desempenho do *software* não ter sido uma preocupação primordial durante seu desenvolvimento, esta se mostrou adequada em testes realizados. Para modelagens contendo 75 elementos (não contabilizando neste número as transições), gastou-se em média 3 segundos para a tradução, execução e recuperação dos resultados da verificação, em computador com a seguinte configuração:

- Processador: AMD Turion X2 2.10 RM-72 GHz;
- Memória RAM: 4 GB;
- Disco rígido: 150 GB;
- Sistema operacional: Windows Seven 64 bits.

Os testes de avaliação do *software* foram executados inicialmente para modelos simplificados e posteriormente foram definidos modelos que permitissem avaliar separadamente a tradução específica para cada um dos casos descritos anteriormente. Os testes iniciais tinham como propósito validar as verificações iniciais tais como da adequação do formato de XML e se o software aceitava modelos sem nó inicial, final ou sem transições. Após estes testes foram realizados outros contendo versões simplificadas dos modelos e seus componentes.

4.4 Automatização da tradução e verificação

Como forma de testar a viabilidade da abordagem proposta, criou-se uma estrutura de *software* que automatiza o processo de tradução da especificação dos fluxos de trabalho, a definição da regra a ser validada, a validação desta regra e a exibição dos resultado e do contraexemplo. A Figura 15 representa a forma geral de funcionamento do *software*. O tradutor se encarrega de percorrer o arquivo XPDL, traduzindo-o para um modelo de verificação a ser executado no verificador NuSMV, incluindo ao final as propriedades a serem validadas que são definidas pelo usuário. Concluída a execução, a interface se encarrega de analisar a resposta obtida e exibí-la ao usuário de uma forma simplificada.

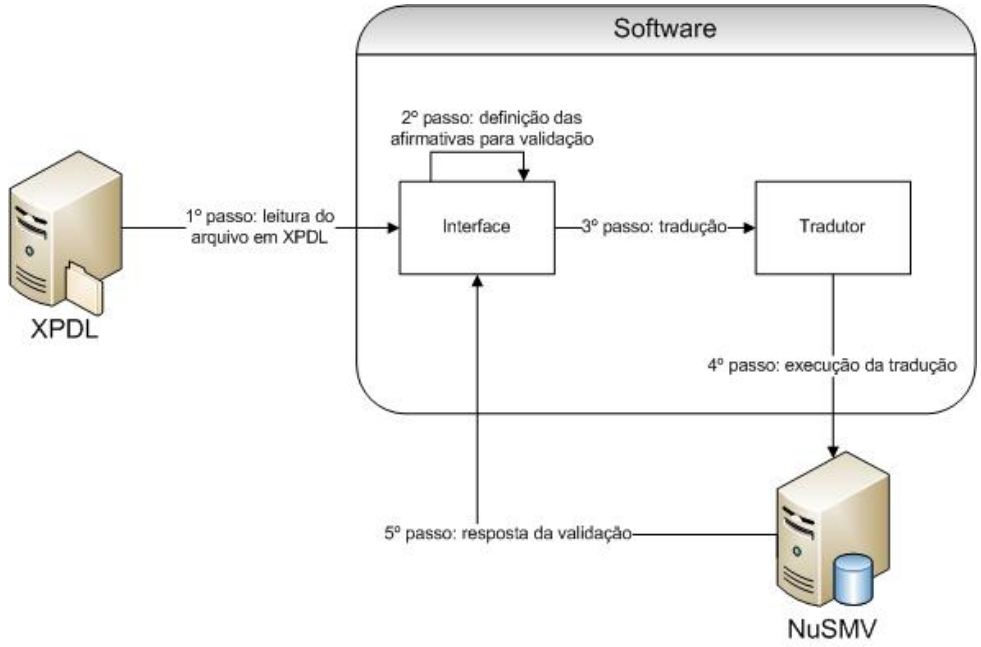

Figura 15: Diagrama de utilização do software

Para a criação do tradutor optou-se pela linguagem Java por sua ampla adoção no ambiente acadêmico e por sua adequação ao propósito da pesquisa. Utilizou-se esta

mesma linguagem para o desenvolvimento da interface, mostrada na Figura 16.

Preenchidos os parâmetros para a tradução, o *software* se encarrega de verificar a adequação dos seguintes pré-requisitos:

- Arquivo XPDL com notação XML correta, escolhido por meio da interface;
- As quatro caixas de opções da interface, preenchidas com valores adequados;
- Existência do diretório de instalação do NuSMV, com o endereço do sistema de arquivos corretamente indicado no código;
- Existência do diretório que servirá de repositório para os arquivos intermediários de tradução e execução, com o endereço do sistema de arquivos corretamente indicado no código;
- Existência de nó inicial e nó final no fluxo mais abstrato;
- Verificação se o nó inicial possui grau zero de entrada;
- Verificação se o nó final possui grau zero de saída.

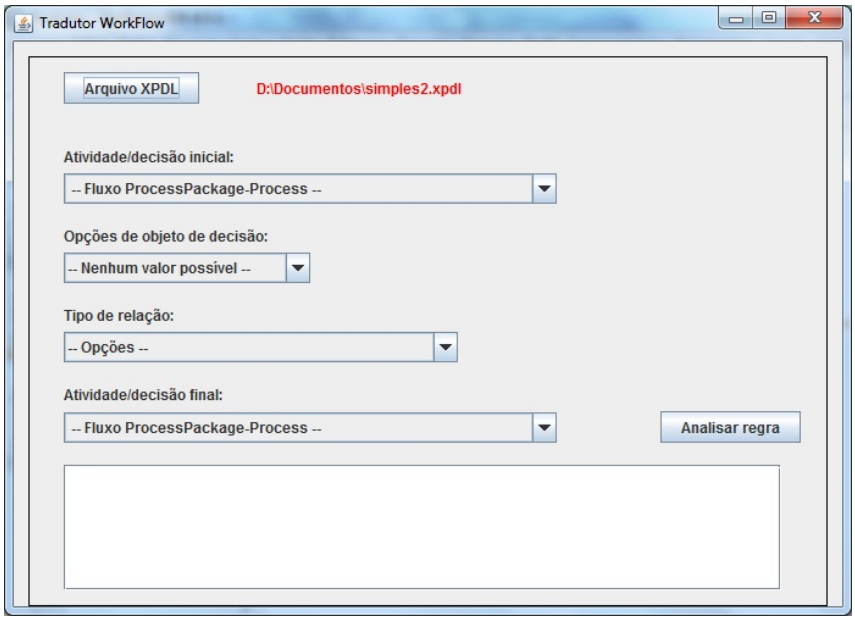

Figura 16: Interface do software

Tais verificações iniciais também se prestam a evitar a avaliação de modelos incompletos e que não se adequam às regras básicas de fluxos de trabalho.

O código responsável pela tradução foi criado de modo que tanto o percurso dos elementos por profundidade ou por largura, nos casos de divisão do fluxo, produzissem o mesmo efeito. A ideia é que a análise de cada elemento é realizada considerando os nós vizinhos e os caminhos possíveis de serem atingidos. Dessa forma, uma vez analisados, os nós não serão reanalisados, não importando a ordem de avaliação, desde que se respeite as especificações das transições.

A ordem a ser respeitada é a avaliação dos diferentes comportamentos possíveis dentro da lógica de fluxos de trabalho. Deve-se primeiramente analisar se um determinado nó é parte integrante de um comportamento complexo que envolve vários outros elementos para posterior análise dos casos mais simples. Se a ordem não for respeitada, corre-se o risco de avaliar isoladamente um elemento como um caso simples, mesmo este fazendo parte de casos mais complexos como paralelismo ou decisões cascateadas. Quando o *software* detecta um caso mais amplo, todos os elementos envolvidos devem ser explorados e traduzidos, marcando-os posteriormente como já analisados.

5 ESTUDO DE CASO

O estudo de caso a ser apresentado supre a necessidade de utilização de um processo não trivial para a validação da capacidade do *software* em cumprir com os objetivos propostos neste trabalho. Para este fim foi escolhido o fluxo de trabalho de tramitação do processo de concessão de aposentadoria aos servidores públicos estaduais do Governo do Estado de Minas Gerais. A modelagem de tal processo foi feita pela empresa Synergia, ligada ao Departamento de Ciência da Computação da Universidade Federal de Minas Gerais, a pedidos da Secretaria de Estado de Planejamento e Gestão (SEPLAG) no ano de 2006.

Cabe ressaltar a diferenciação na nomenclatura adotada na modelagem entre os termos processo, fluxo de trabalho e dossiê. Esta foi necessária para esclarecer a aplicação do termo processo, utilizado comumente para exprimir os três significados. O fluxo de trabalho seria o desenho de encadeamento de tarefas, definindo transições, regras de negócio, entre outros, e que se configura como um dos temas principais do presente trabalho. Já processo foi definido explicitamente como as instâncias de um fluxo, ou seja, se um funcionário público faz o pedido de aposentadoria, é criado um novo processo que irá percorrer os caminhos necessários do fluxo. Já dossiê é utilizado para designar a pasta de documentos relativa a um processo.

A modelagem do fluxo de concessão de aposentadoria tinha como objetivo a formalização do mesmo e também de servir como base para revisões do processo. Tem como premissa o detalhamento das atividades realizadas somente dentro da SEPLAG, apesar de registrar outras instituições como atores no processo. A notação utilizada foi a BPMN, o que garante a adequação do modelo aos propósitos deste trabalho.

O documento contém 199 páginas e descreve de maneira detalhada o processo. Nesta modelagem foi adotada a estratégia de modelagem em vários níveis de abstração. Dessa forma, o diagrama de maior abstração (Figura 17) é detalhado em vários níveis. Para efeito deste trabalho só se considerou o detalhamento do processo até o segundo

nível, o que reduz a quantidade de elementos descritos, tendo em vista a alta complexidade do fluxo. Tal restrição não foi imposta por limitação da abordagem ou do software desenvolvido mas sim para reduzir o esforço de transferência entre a modelagem original disponível somente em forma impressa para um formato eletrônico. Identificou-se também que a restrição não significou o abandono de algum caso ou comportamento passível de tradução. Por meio desta restrição o fluxo completo passou a conter 75 elementos (não contabilizadas as transições). Todos os diagramas para o fluxo estudado e seus subfluxos estão presentes no Anexo A enquanto que a tradução de todo o fluxo no formato SMV está no Anexo B. Dois outros fluxos, suas traduções e validações de teste estão presentes nos Anexos C e D.

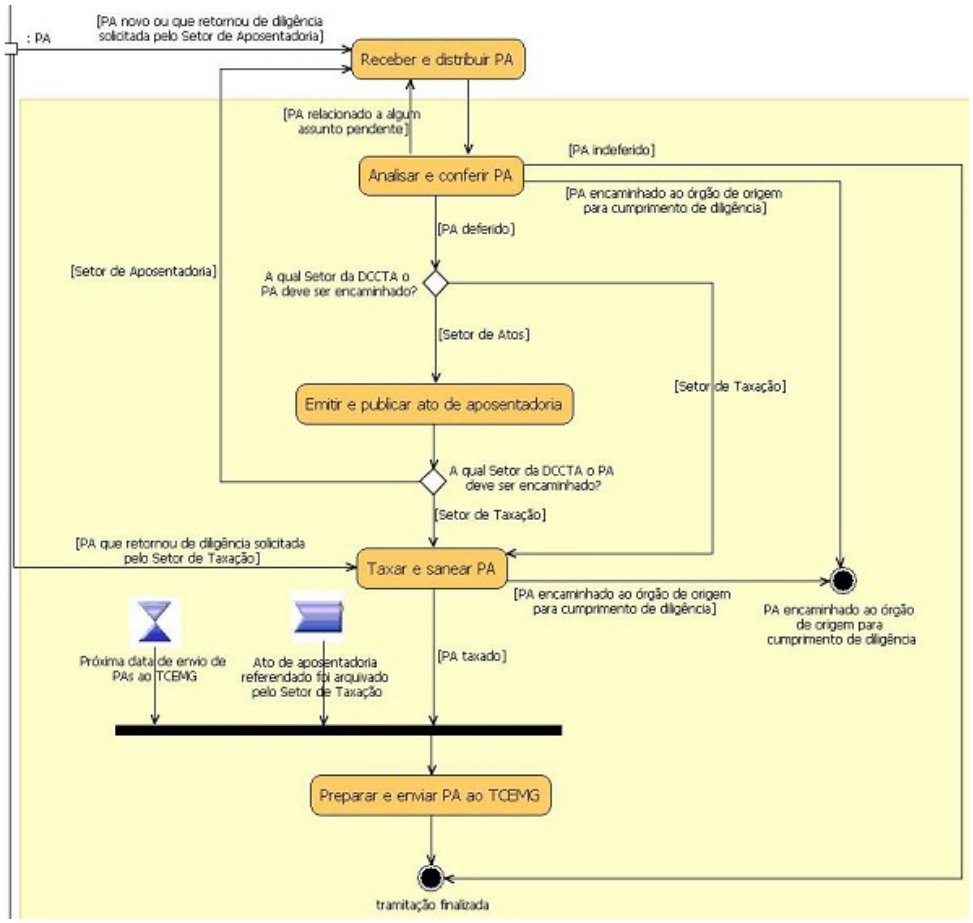

Figura 17: Modelagem do fluxo de tramitação do processo de aposentadoria

O modelo contido no documento foi remodelado para que pudesse ser usado nos testes. Para a recriação da modelagem e posterior exportação do modelo para a linguagem XPDL foram utilizados os *softwares* TIBCO Business Studio versão 3.2.0 e Bizagi Process Modeler versão 1.1. Adotou-se o critério de privilegiar ferramentas que fossem gratuitas como forma de demonstrar a viabilidade de adoção desta metodologia sem custos

adicionais.

5.1 Preparação da avaliação

A modelagem contida no documento foi realizada objetivando a análise da mesma por especialistas em negócios. Tal propósito justificou a adoção de um estilo de modelagem que se mostrou inadequado ao uso do modelo em estruturas de orquestração de processos, assim como seu uso pelo *software* criado nesta pesquisa. Apesar de ter sido utilizada a notação BPMN, a modelagem apresenta uma limitação para estes usos. Esta se refere à utilização de rótulos nas transições entre atividades para determinar condições de execução.

Na figura 17, como exemplo, a primeira transição depende da condição de que o processo de aposentadoria (PA) seja ou não o retorno de diligência solicitada pelo Setor de Taxação. Caso a condição seja positiva, a primeira atividade executada seria "Taxar e sanear PA". Caso contrário, a primeira atividade seria "Receber e distribuir PA". Para a utilização do modelo em ferramentas automatizadas, é necessário que tais condições estejam em elementos de decisão, criadas especificamente para este fim. Os rótulos de elementos, por terem um caráter não formal e somente informativo, impossibilitam a identificação e a interpretação de tais condições.

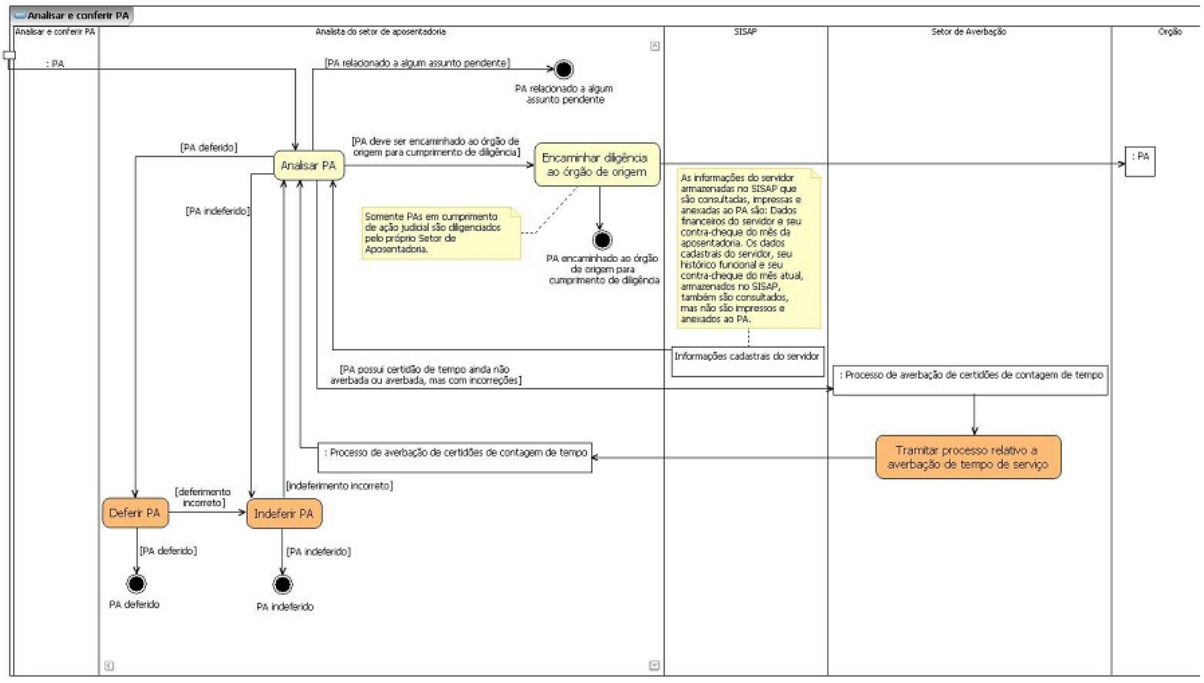

Figura 18: Fluxo "Analisar e conferir PA" original

Outra alteração realizada na transposição da modelagem original foi a remoção de itens que não são considerados na análise feita pelo software. Associação entre as atividades e atores responsáveis, rótulos, comentários, entre outros elementos, não foram incluídos pois não afetam a análise de condições de execução de atividades realizada pelo *software*. Ambas modificações podem ser verificadas nas Figuras 18 e 19, que representam um subfluxo do modelo geral em seu estado original e após a alteração.

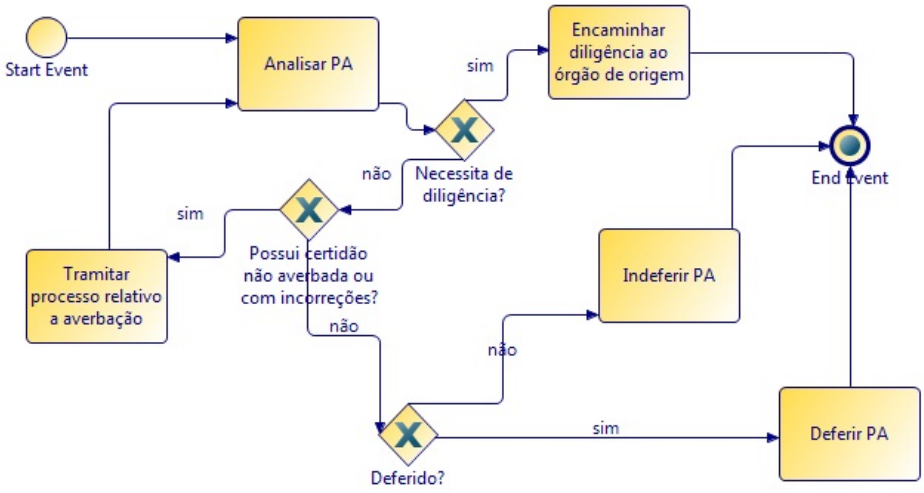

Figura 19: Fluxo "Analisar e conferir PA" alterado

A adaptação de modelos, como a descrita anteriormente, pode acarretar em alterações na lógica do processo, fazendo com que o modelo alterado não reflita todas as características do modelo original. Torna-se necessário validar o modelo alterado às vistas das regras identificadas e modeladas anteriormente. Tal necessidade reforça a utilidade da ferramenta desenvolvida pois esta permite validar se o modelo alterado se adequa à lógica do modelo original. Dessa forma, por exemplo, em um processo de adaptação de fluxos de trabalho para fins de orquestração, o *software* se mostraria útil para garantir a exatidão entre os modelos.

Foram executadas diversas avaliações para verificar a adequação do software às regras de sequenciamento de tarefas e dessa forma, analisar a adequação da ferramenta aos propósitos da pesquisa.

A validação inicial da capacidade do *software* em detectar deficiências na modelagem de *workflows* se deu por meio de pré-testes que envolveram a avaliação das condições de execução dos vários tipos de elemento isoladamente. Por meio dessa etapa foi possível atestar que o *software* foi capaz de definir de forma exata as condições descritas nas mo-

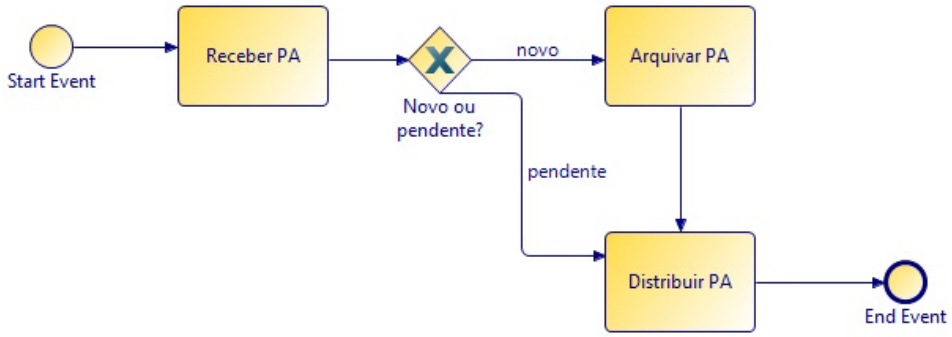

Figura 20: Fluxo "Receber e distribuir PA"

delagens dos fluxos utilizados como base de avaliação. Como exemplo, o fluxo descrito na Figura 20 foi traduzido no Código 5.1.

O Código 5.1 descreve as regras de execução para cada um dos elementos envolvidos. Nesse caso, o nó inicial é executado por padrão (linha 11). Todas as outras variáveis são inicializadas como falso. Elas assumem uma nomenclatura padrão (linhas 3 a 8) para facilitar a execução do modelo no verificador. Como a atividade "Receber PA" só depende do nó inicial, esta também é executada logo a seguir (linha 18). Entre as linhas 20 a 25 são definidas as regras referentes à atividade "Arquivar PA" para a manutenção de seu status de execução ou sua mudança não-determinística (linha 23). A atividade "Distribuir PA" não depende da execução da atividade "Arquivar PA" pois em qualquer um dos caminhos possíveis após o objeto de decisão "Novo ou pendente?" ela é executada (linhas 26-30). Finalmente, o nó final é executado se a atividade "Distribuir PA" é executada (linhas 31-35).

Para fazer a avaliação real da capacidade da ferramenta em detectar problemas e efetivamente auxiliar na validação de fluxos buscou-se obter modelagens reais e suas revisões anteriores e testá-las para traçar uma comparação entre o que foi corrigido na prática e quais correções seriam detectáveis pelo *software*. No entanto não foi possível obter tais documentos, tanto do estudo de caso, quanto de outros fluxos que pudessem servir como base para teste.

Como medida de avaliação dessa capacidade, optou-se pela estratégia de avaliação completa do fluxo à vista das regras de negócio do processo e também pela estratégia de inserção de erros controlados na modelagem do estudo de caso.

Faz-se necessário comentar a respeito do comportamento esperado do *software* no caso de existência de ciclos nos fluxos de trabalho. Apesar de não ser usual que um ciclo

```
MODULE main
VAR
        atividade0: boolean; -- Nó inicial
        atividade1: boolean; -- Arquivar PA
        atividade2: boolean; -- Receber PA
        -- atividade3: Novo ou pendente?
        atividade4: boolean; -- Distribuir PA
        atividade5: boolean; -- End Event

ASSIGN
        init(atividade0) := 1;
        init(atividade1) := 0;
        init(atividade2) := 0;
        init(atividade4) := 0;
        init(atividade5) := 0;

        next(atividade0) := 1;
        next(atividade2) := 1;

        next(atividade1) :=
                case
                        atividade1=1: 1;
                        atividade2=1: { 0,1 };
                        1: 0;
                esac;
        next(atividade4) :=
                case
                        (atividade2=1): 1;
                        1: 0;
                esac;
        next(atividade5) :=
                case
                        atividade4=1: 1;
                        1: 0;
                esac;
```

Código 5.1: Tradução do fluxo da Figura 20

em um *workflow* seja executado inúmeras vezes, teoricamente isso é possível e é desta forma que o verificador interpreta. No caso em que um ciclo ocorra em um fluxo, o *software* entende que não necessariamente o ciclo terá um fim. Isso não se trata de uma limitação do mesmo, justamente por ser teoricamente possível tal ocorrência.

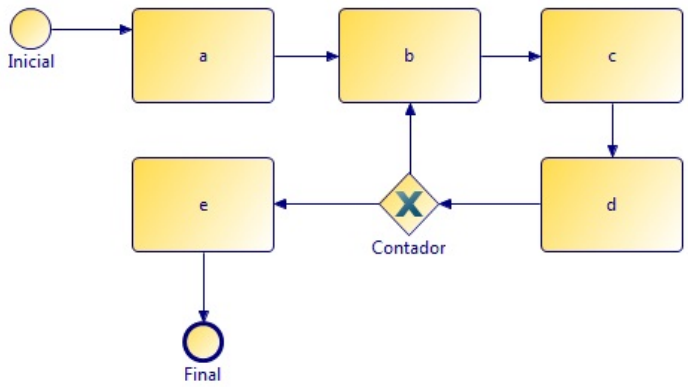

Figura 21: Modelo com ciclo não encapsulado

Entretanto, existe a possibilidade, por meio da especificação de BPMN, de definir

que uma atividade encapsule um ciclo usando a especificação de *Standard Loop* ou *Multi-instance Loop* (OMG, 2009). Por meio desta definição pode-se especificar a existência de um contador de execuções e impor um limite máximo para este contador. Seria necessário que todos os ciclos existentes no fluxo estivessem encapsulados em atividades com contadores para que sua execução fosse limitada.

Como exemplo da organização necessária, a Figura 21 demonstra um fluxo com um ciclo sem a utilização desta estratégia, enquanto a Figura 22 mostra a modelagem que atende a especificação de limitação de execuções. Nesta última, criou-se uma nova atividade denominada x que representa o encapsulamento do ciclo situado à direita e que se tornou um subfluxo associado a esta nova atividade.

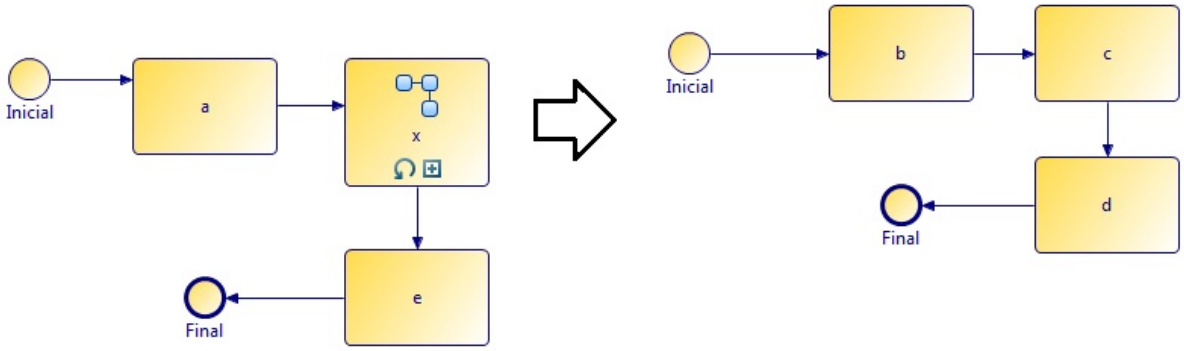

Figura 22: Modelo com encapsulamento do ciclo

Na tradução de tal elemento para fins de verificação, o contador existente na modelagem em XPDL é representado por uma variável numérica que é considerada para definir as regras de transição das atividades do ciclo. Nestes casos, o ciclo não é executado infinitamente e, pelo fato de ser um comportamento bastante específico, configura-se como um novo caso de tradução, além dos quatro anteriormente dispostos.

O comportamento do *software* para o modelo apresentado considera as condições de repetição do ciclo. O nó inicial do subfluxo x passa a ser executado sempre que a atividade a for executada ou quando o nó final deste subfluxo for executado e o contador não tiver atingido seu limite. Se a execução atingir o nó final do subfluxo sem ter esgotado o contador, as atividades são marcadas como não executadas e o ciclo é novamente iniciado. As condições para a execução da atividade e passam a ser não somente a execução do nó final do subfluxo mas também o alcance do limite pelo contador.

No caso de ciclos não encapsulados, como o existente na Figura 21, apesar de ser possível inserir um objeto de decisão simples ou complexo para executar a interrupção

do ciclo, essa alternativa não é passível de ser considerada pela ausência de um padrão ou de um elemento específico para este fim. Ao mesmo tempo, não faz parte do escopo deste trabalho determinar um padrão para tal detecção pelo *software* pois impactaria na aplicabilidade do mesmo, gerando a necessidade de conformação dos fluxos de trabalho a este padrão específico. A mesma lógica não se aplica à alternativa de encapsulamento dos ciclos pelo fato de ser derivada dos padrões contidos nas especificações de BPMN e XPDL.

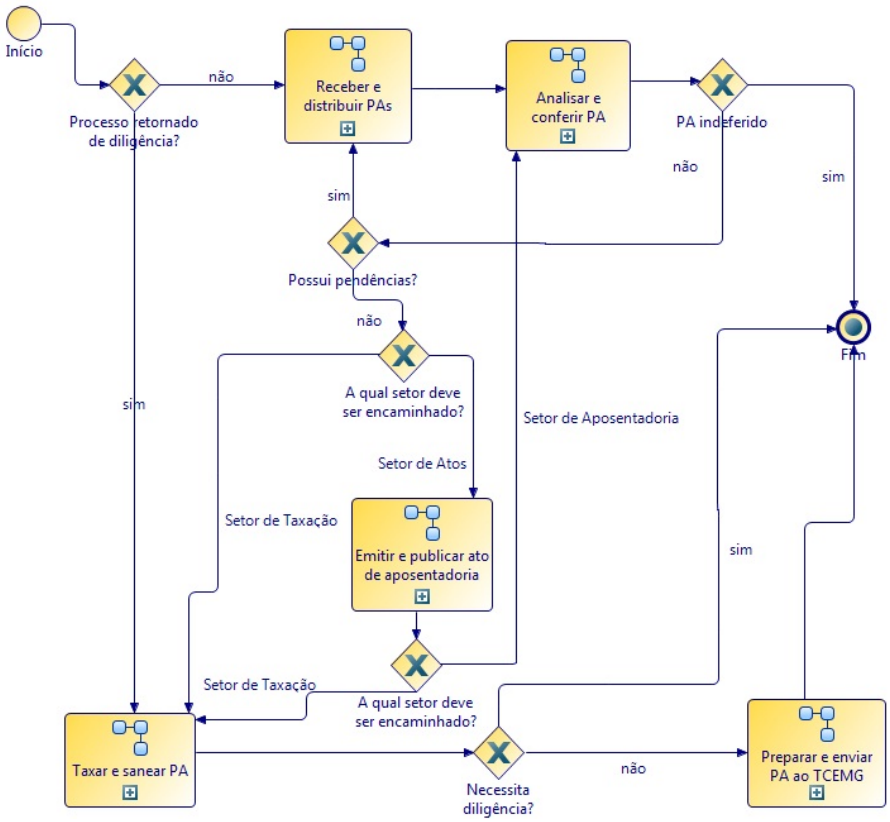

Figura 23: Fluxo de trabalho mais abstrato do estudo de caso

Tome como exemplo o ciclo apresentado na Figura 23, que representa o fluxo em sua visão mais abstrato. O ciclo envolve as atividade "Receber e distribuir PA" e "Analisar e conferir PA" e as decisões "PA indeferido?" e "Possui pendências?". Mesmo o ciclo possuindo dois objetos de decisão que permitem o encerramento do ciclo, este ainda pode persistir. E mesmo que fosse criada uma saída do ciclo caso este atingisse um determinado número de execuções, por não haver um padrão para definir esse limite, ainda assim o *software* consideraria a possibilidade de uma execução ininterrupta do ciclo.

5.2 Resultados do estudo de caso

Todas as validações identificadas para a abordagem, descritas no Capítulo 4, foram testadas em suas hipóteses de respostas positivas e negativas no *software*desenvolvido para verificar a viabilidade da metodologia. Privilegiou-se testes que utilizassem o estudo de caso sem alterações, quando isto fosse possível. Caso contrário, optou-se pela inserção de erros controlados que simulasse tais ocorrências.

O teste inicial, por meio de erros controlados, foi a remoção de nós iniciais ou finais para verificar o comportamento da ferramenta. No caso do nó inicial, tal verificação se dá antes da execução de qualquer tipo de validação. Sendo assim, o *software* acusa o problema e não executa a tradução e verificação. No caso do nó final, para permitir a validação de fluxos incompletos, como descrito anteriormente, o *software* permite a realização de verificações. No entanto, pelo fato do fluxo testado não estar completo, verificações que possam ser afetadas pela ausência do dito nó poderão ser incorretamente avaliadas enquanto essa situação se mantiver.

Apresenta-se a seguir os resultados para cada tipo de validação identificado.

5.2.1 A execução de uma atividade ou decisão implica na execução de outra

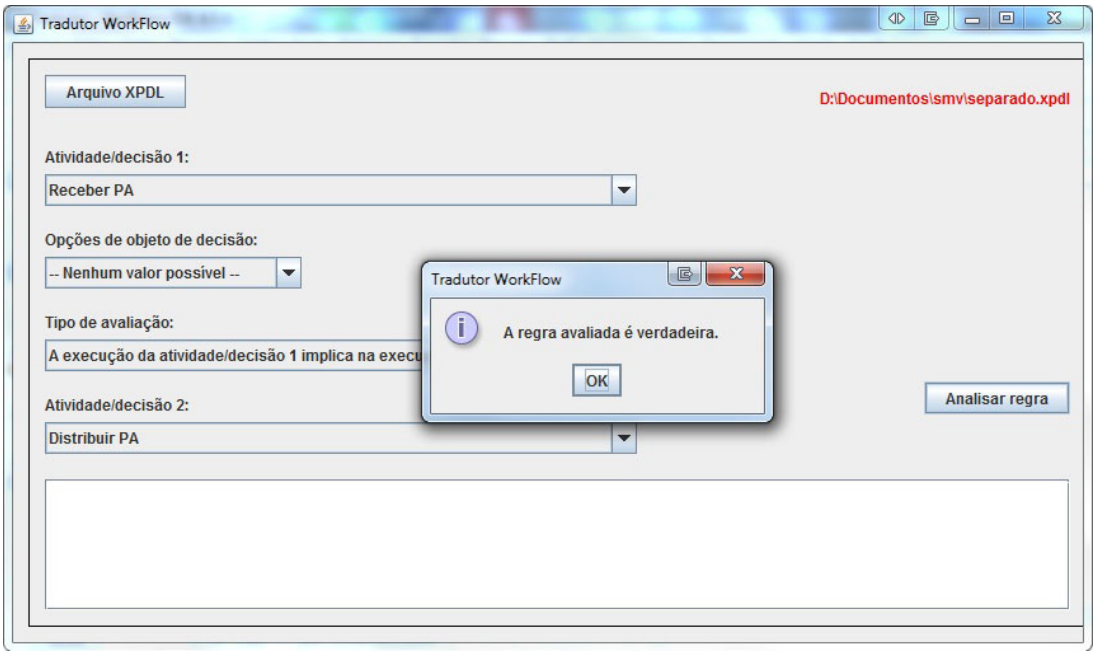

Figura 24: Resposta da ferramenta para validações de regras verdadeiras

Tome como exemplo o fluxo da Figura 20. Quando testada a regra de que sempre

que a atividade "Receber PA" for executada, também será executada a atividade "Distribuir PA", a resposta obtida pelo *software* foi de que tal regra é verdadeira (Figura 24). Tal resposta está correta pois em todos os caminhos de execução possíveis a partir da execução da primeira atividade, a segunda atividade também será executada.

Para outra validação, executável no mesmo fluxo, obteve-se resposta negativa. Se testarmos a afirmativa de que sempre que a atividade "Receber PA" for executada acontecerá também a execução da atividade "Arquivar PA", a resposta será como a exibida na Figura 25.

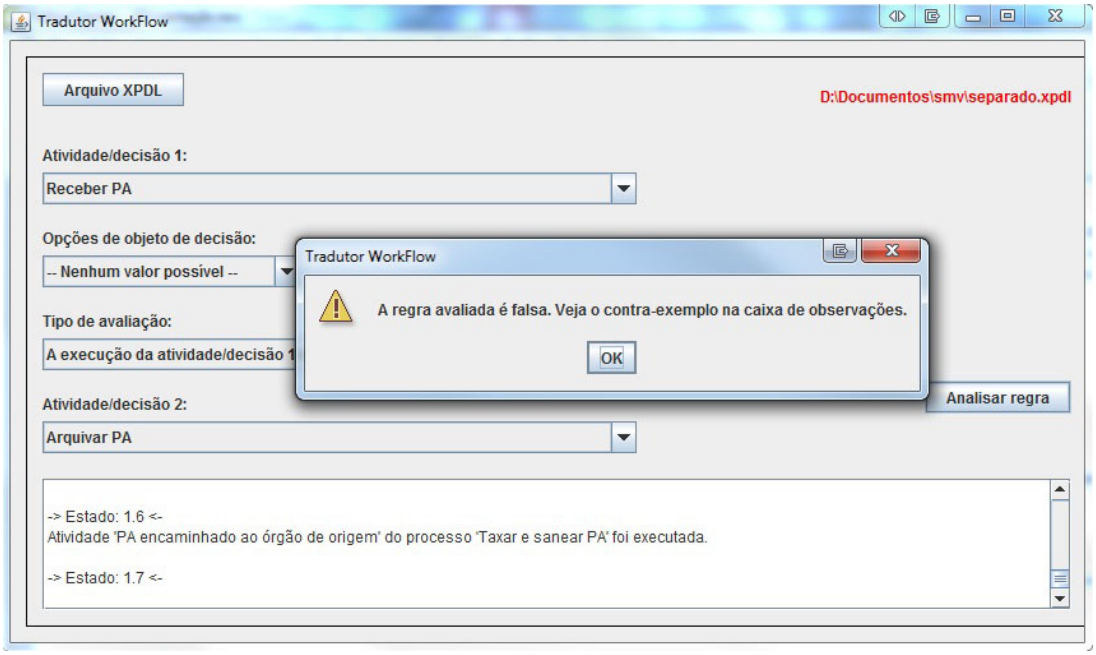

Figura 25: Resposta da ferramenta para validações de regras falsas

Nesse caso, a resposta se encontra novamente correta pois existe a possibilidade da segunda atividade não ser executada a partir da primeira, dependendo da validade da condição do elemento de decisão denominado "Novo ou pendente?".

Pode-se observar que em respostas negativas, um contraexemplo é fornecido em um campo específico da interface. Tal contraexemplo consiste na descrição de uma sequência de execução possível para o modelo que viole a regra analisada. Note o leitor que, ao contrário do que ocorre no modelo traduzido, o contraexemplo utiliza os nomes reais dos elementos do fluxo. Como descrito antes, os elementos são traduzidos em variáveis com nomenclatura padrão para facilitar o funcionamento interno do *software*. Porém estes são traduzidos nos contraexemplos para as respectivas atividades do modelo original, para não comprometer o entendimento dos usuários.

Outra hipótese de obtenção de uma resposta negativa para esse tipo de validação é a própria inexistência de caminho de execução entre os elementos analisados. Utilizando como base o fluxo da Figura 23, pode-se notar que não existe caminho de execução possível no qual a execução de algum elemento do subfluxo "Taxar e sanear PA" implique na execução de algum elemento do subfluxo "Receber e distribuir PA", por exemplo. Se validarmos tal hipótese na ferramenta usando a validação aqui analisada, obtém-se a resposta negativa, como esperado.

Este tipo de validação também é afetada no caso de ciclos ininterruptos. Invertendo-se a lógica da validação anterior obtém-se a regra de que a execução de alguma atividade do subfluxo "Receber e distribuir PA" implica na execução de algum elemento do subfluxo "Taxar e sanear PA". Para esta validação será obtida corretamente a resposta negativa devido ao fato de haver um ciclo ininterrupto envolvendo tal subfluxo, como pode ser observado na Figura 23. Havendo um ciclo ininterrupto, existe ao menos um caminho possível no qual a sequência de execução não consiga atingir o segundo subfluxo.

5.2.2 A execução de uma atividade ou decisão implica na não execução de outra

Este tipo de validação não é exatamente o contrário do tipo anterior. Em outras palavras, respostas negativas para o tipo anterior não significam necessariamente respostas positivas para este. Tal afirmação pode ser confirmada retomando-se a segunda validação realizada anteriormente, na qual se testou o relacionamento entre as atividades "Receber PA" e "Arquivar PA" da Figura 20. Ao testar a afirmativa de que sempre que a primeira atividade for executada, a segunda não será, obtém-se novamente uma resposta negativa. A resposta se encontra correta pois, mesmo que segunda atividade não seja executada sempre que a primeira é, isso não significa que ela nunca será executada quando a primeira for.

Dessa forma, respostas negativas são obtidas quando existir alguma possibilidade da execução das duas atividades em uma mesma sequência de execução. Para testar tal ocorrência, repetiu-se os dois primeiros testes do tipo de validação anterior e obteve-se a resposta esperada em ambos os casos.

Uma resposta positiva para esta validação é conseguida se forem avaliados elementos em caminhos excludentes. Caminhos excludentes são obtidos a partir de objetos de decisão que não estejam inseridos em ciclos. Se esses estiverem em ciclos, seria possível a execução de caminhos excludentes em alguma das sequências do ciclo. No fluxo estudado,

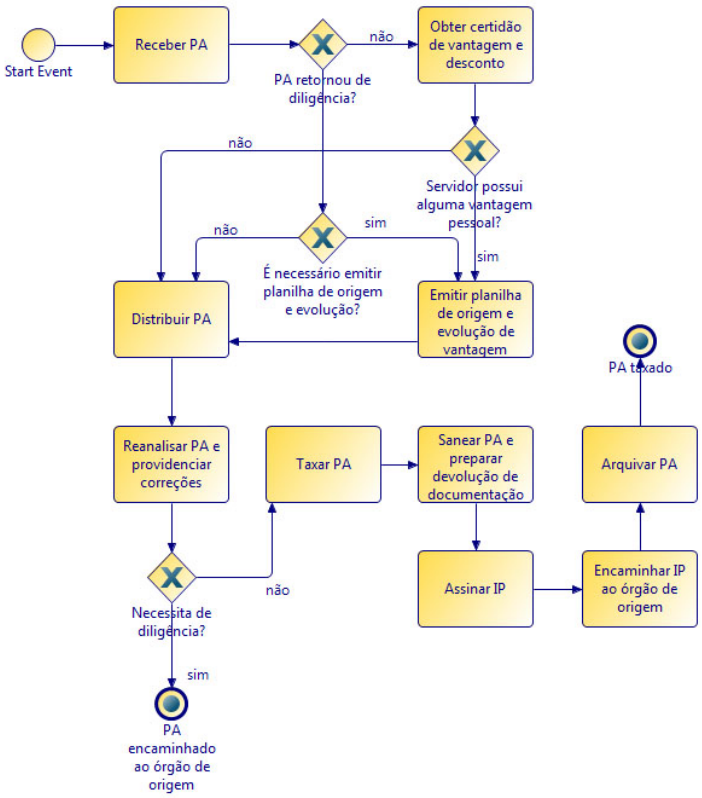

Figura 26: Fluxo "Taxar e sanear PA"

somente uma decisão do subfluxo "Taxar e sanear PA" (Figura 26) possui tal característica. Os subfluxos "Receber e distribuir PA", "Analisar e conferir PA" e "Emitir e publicar ato de aposentadoria" da Figura 23 não poderiam ser usados para este teste pois estão envolvidos em dois ciclos. Já no subfluxo "Preparar e enviar PA ao TCEMG" não existem objetos de decisão. Mesmo no subfluxo restante, "Taxar e sanear PA", somente a decisão "Necessita de diligência" possui caminhos excludentes não inseridas em ciclos. Testou-se tal hipótese no *software* e a resposta obtida foi positiva.

5.2.3 A execução de uma atividade ou decisão significa a possibilidade da execução de outra

Para esta validação, obtém-se verdadeiro se existir ao menos uma possibilidade de que ambas atividades possam ser executadas em uma sequência. Como consequência lógica, sempre que no primeiro tipo de validação a resposta for positiva, nesta também será. Testou-se a hipótese que obteve tal resposta na primeira validação e o *software* informou corretamente a validade da regra. Porém não é possível concluir que, sempre que se obtiver resposta negativa para o segundo tipo de validação, será obtida a resposta positiva neste tipo.

Ainda por conclusão lógica, quando se compara esta validação com as anteriores, obtém-se respostas negativas se for obtidas resposta positivas para a segunda. Quando utilizadas as mesmas validações do tipo anterior com as respostas apontada acima, obteve-se respostas negativas. No entanto, respostas negativas para o primeiro tipo não significa a obtenção de respostas negativas para este.

5.2.4 A execução de uma atividade ou decisão depende da prévia execução de outra

Este tipo de validação pode ser considerado como um detalhamento da primeira validação. Para obter respostas positivas, não somente é necessário obter positivo para a primeira validação, mas também é preciso que os elementos avaliados não sejam executados simultaneamente. Dessa forma, garante-se que uma determinada atividade só será executada se outra for executada e concluída anteriormente.

O primeiro teste realizado, entre as atividades "Receber PA" e "Distribuir PA", foi refeito para este tipo de validação. Foi testado se esta segunda atividade depende da execução prévia da primeira. O *software* obteve a resposta positiva, como esperado.

Uma hipótese de obtenção de resposta negativa para este caso se dá quando é obtido falso para a avaliação do primeiro tipo. Testou-se as três hipóteses que obtiveram falso como resposta para o primeiro tipo e, em todos os casos, obteve-se falso.

Outra possibilidade para a obtenção de falso se refere ao caso em que as atividades são executadas simultaneamente. Isso poderia ocorrer no caso de caminhos paralelos. Nesse caso identificou-se um erro na modelagem original do estudo de caso. O fluxo "Emitir e publicar ato de aposentadoria" (Figura 27) foi modelado de forma a possibilitar a publicação de ato de aposentadoria sem a assinatura prévia do processo. Tanto a atividade "Publicar ato de aposentadoria" quanto a atividade "Obter assinatura do ato de aposentadoria" são executadas paralelamente ao invés de especificar que a segunda precede a primeira.

Cabe ressaltar que a ausência de mais erros não pode ser considerada um indicativo de inadequação do *software*, mas sim indicativo que a modelagem do processo foi bem realizada. A identificação de um erro na lógica do processo após um considerável esforço de análise, revisão e correção por parte da equipe responsável pelo projeto não era esperada. Porém, mesmo após tal esforço, foi possível identificar uma incorreção por meio de testes realizados com o *software* no caso estudado. Ressalta-se também que este erro não decorreu da alteração do fluxo original para um modelo automatizável, promovida

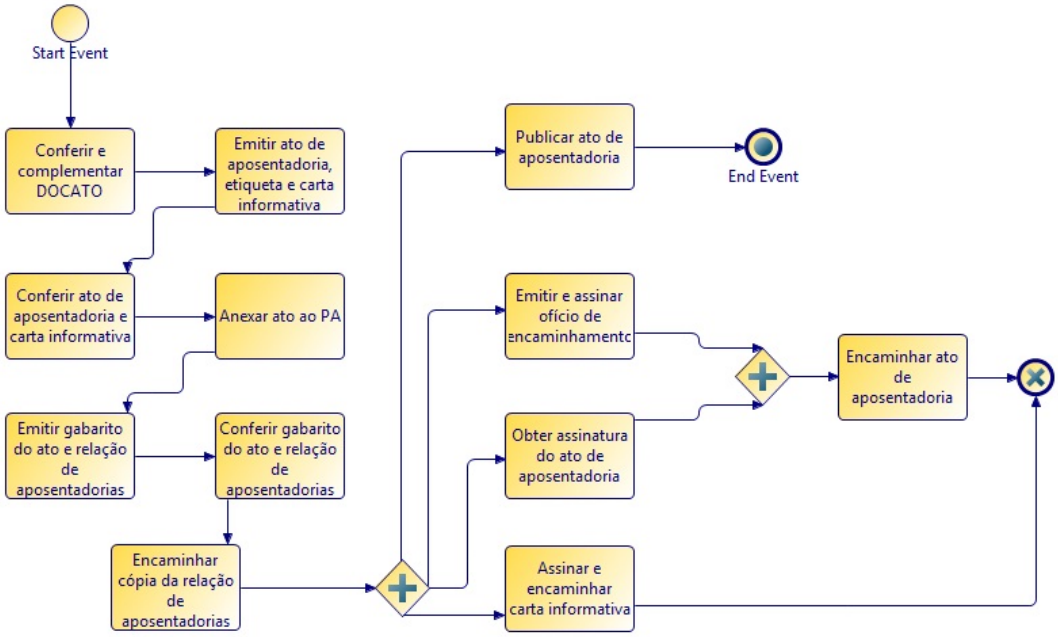

Figura 27: Fluxo "Emitir e publicar ato de aposentadoria"

neste trabalho. Identificou-se que o problema remonta ao fluxo original.

5.2.5 É possível que uma determinada atividade não seja executada

Para este tipo de validação, assim como para o próximo, nos quais os elementos são avaliados isoladamente, não é possível testar objetos de definição de condicional ou paralelismo. Tais elementos, apesar de serem avaliados pelo *software* e terem suas lógicas traduzidas, não são transformados em variáveis pelo simples fato de não representarem ações passíveis de execução mas sim definições de lógica do processo.

Obtém-se respostas positivas para esta validação quando o elemento analisado encontra-se em um caminho derivado de um objeto de decisão ou após ciclos ininterruptos. Como teste desta ocorrência, verificou-se qual seria a resposta obtida pelo *software* para as atividades "Receber PA" e "Arquivar PA" do subfluxo contido na Figura 20. Obteve-se corretamente a resposta positiva para ambas pois todo o subfluxo se encontra dependente da decisão inicial "Processo retornado de diligência".

Testou-se também se o *software* interpreta corretamente as regras de paralelismo, não aplicando a mesma lógica de objetos de decisão. Para isso utilizou-se o subfluxo da Figura 27 e os quatro elementos dos caminhos paralelos. Obteve-se positivo, ao contrário do que se inicialmente esperava. No entanto, percebeu-se que o subfluxo como um todo se

encontra em um dos caminhos afetados pela decisão "A qual setor deve ser encaminhado?", como pode ser observado na Figura 23. Dessa forma, a avaliação foi corretamente avaliada pelo *software*. Ao testar os mesmos elementos, isolando o fluxo "Emitir e publicar ato de aposentadoria", obteve-se desta vez a resposta negativa para os quatro testes.

Para a obtenção de resposta negativa é necessário que o elemento avaliado seja executado em todas as hipóteses de caminhamento do fluxo. No fluxo estudado, essa regra somente se aplica ao nó inicial geral. Todos os outros podem não ser executados, seja pela existência dos ciclos, seja pela dependência de condicionais, como pode ser observado no fluxo da Figura 23.

5.2.6 Uma determinada atividade nunca é executada

Nesse caso, somente se o elemento avaliado estiver desconexo (grau de entrada zero), é obtido verdadeiro como resposta. Para testar tal possibilidade foi necessário novamente lançar mão da inserção de erros controlados. Testou-se a atividade "Anexar ato ao PA" do fluxo da Figura 27, antes e após a remoção da transição de entrada. Antes da remoção a resposta obtida foi negativa. Após a remoção obteve-se positiva, conforme esperado. A mesma validação foi executada para a atividade em paralelismo "Publicar ato de aposentadoria" do mesmo fluxo e para a atividade "Arquivar PA" em caminho condicional da Figura 20. Em ambos os casos, antes e após a remoção das transições, obteve-se os resultados esperados, conforme exemplo anterior.

5.2.7 É sempre possível atingir o final do fluxo

Para obter uma resposta positiva para esta validação é necessário que exista um nó final do fluxo geral, que este não esteja desconexo e que não haja ciclos ininterruptos no fluxo. Testado o fluxo do estudo de caso, obteve-se a resposta negativa pois existem ciclos ininterruptos no mesmo. Para obter a resposta positiva, foi necessário alterar o fluxo para incorporar os ciclos em atividades do tipo *"Standard Loop"* , impondo um limite de execuções para estes. Com esta modificação foi possível obter a resposta desejada.

Testou-se também as outras hipóteses de resposta negativa. Ao remover o nó final, o *software* apresenta uma mensagem de erro acusando a inexistência de tal tipo de nó, ficando impossibilitada a execução desta validação. Se, depois de restaurado o nó final, forem removidas as transições para este, a validação é realizada mas obtém-se a resposta negativa, como esperado.

5.2.8 Conclusão do estudo de caso

Por meio dos testes descritos anteriormente foi possível determinar a viabilidade da abordagem na realização de validações em fluxos de trabalho e a capacidade do *software* no tratamento dos vários elementos de *workflow* e na tradução da lógica dos mesmos. A seguir apresenta-se um quadro comparativo que resume os resultados dos testes. Apresenta-se no próximo capítulo a conclusão desta dissertação, assim com a sugestão de trabalhos futuros.

Tabela 1: Quadro de resultados dos testes - Estudo de caso

Validação	Atividade/decisão 1	Fluxo	Atividade/decisão 2	Fluxo	Resultado
A execução de uma atividade ou decisão implica na execução de outra	Receber PA	Receber e distribuir PA	Distribuir PA	Receber e distribuir PA	Verdadeiro
	Receber PA	Receber e distribuir PA	Arquivar PA	Receber e distribuir PA	Falso
	-	Taxar e sanear PA	-	Receber e distribuir PA	Falso
	-	Receber e distribuir PA	-	Taxar e sanear PA	Falso
A execução de uma atividade ou decisão implica na não execução de outra	Receber PA	Receber e distribuir PA	Arquivar PA	Receber e distribuir PA	Falso
	Taxar PA	Taxar e sanear	PA encaminhado ao órgão de origem	Taxar e sanear	Verdadeiro
	Receber PA	Receber e distribuir PA	Distribuir PA	Receber e distribuir PA	Falso
A execução de uma ativ. ou decisão significa a possib. de exec. de outra	Receber PA	Receber e distribuir PA	Distribuir PA	Receber e distribuir PA	Verdadeiro
	Taxar PA	Taxar e sanear	PA encaminhado ao órgão de origem	Taxar e sanear	Falso
A execução de uma atividade ou decisão depende da prévia execução de outra	Receber PA	Receber e distribuir PA	Distribuir PA	Receber e distribuir PA	Verdadeiro
	Receber PA	Receber e distribuir PA	Arquivar PA	Receber e distribuir PA	Falso
	-	Taxar e sanear PA	-	Receber e distribuir PA	Falso
	-	Receber e distribuir PA	-	Taxar e sanear PA	Falso
	Obter assinatura do ato de aposentadoria	Emitir e publicar ato de aposentadoria	Publicar ato de aposentadoria	Emitir e publicar ato de aposentadoria	Falso
É possível que uma determinada atividade não seja executada	Receber PA	Receber e distribuir PA	-	-	Verdadeiro
	Arquivar PA	Receber e distribuir PA	-	-	Verdadeiro
	Publicar ato de aposentadoria	Emitir e publicar ato de aposentadoria	-	-	Verdadeiro
	Emitir e assinar ofício de encaminhamento	Emitir e publicar ato de aposentadoria	-	-	Verdadeiro
	Obter assinatura do ato de aposentadoria	Emitir e publicar ato de aposentadoria	-	-	Verdadeiro
	Assinar e encaminhar carta informativa	Emitir e publicar ato de aposentadoria	-	-	Verdadeiro
	Publicar ato de aposentadoria	Emitir e publicar ato de aposentadoria (isolado)	-	-	Falso
	Emitir e assinar ofício de encaminhamento	Emitir e publicar ato de aposentadoria (isolado)	-	-	Falso
	Obter assinatura do ato de aposentadoria	Emitir e publicar ato de aposentadoria (isolado)	-	-	Falso
	Assinar e encaminhar carta informativa	Emitir e publicar ato de aposentadoria (isolado)	-	-	Falso
	Nó inicial	Principal	-	-	Falso
Uma determinada atividade nunca é executada	Anexar ato ao PA	Emitir e publicar ato de aposentadoria	-	-	Falso
	Publicar ato de aposentadoria	Emitir e publicar ato de aposentadoria	-	-	Falso
	Arquivar PA	Receber e distribuir PA	-	-	Falso
	Anexar ato ao PA	Emitir e publicar ato de aposentadoria (alterado)	-	-	Verdadeiro
	Publicar ato de aposentadoria	Emitir e publicar ato de aposentadoria (alterado)	-	-	Verdadeiro
	Arquivar PA	Receber e distribuir PA (alterado)	-	-	Verdadeiro
É sempre possível atingir o final do fluxo	-	-	-	-	Falso

6 CONCLUSÃO / TRABALHOS FUTUROS

O tema da verificação de modelagens de negócio vem adquirindo importância na última década e se mostrado eficaz na resolução antecipada de inconsistências que podem representar custos de solução crescentes ao longo dos projetos de desenvolvimento e ao longo da vida útil dos *softwares*. Detectar erros nesse estágio também representa oportunidades de revisão e otimização dos próprios processos de negócio, assim como a redução de riscos em casos de processos críticos. A técnica de *model checking* se mostra adequada para o propósito de validação de regras de negócio, conforme pode ser observado neste e nos outros trabalhos relacionados.

O presente trabalho apresenta uma abordagem e uma ferramenta para a realização de verificações concernentes a regras de execução de atividades, seu sequenciamento e interdependências, podendo ser automatizado por meio de uma interface que abstrai grande parte da complexidade do processo, aumentando assim a aplicabilidade da técnica. Tendo como público-alvo da ferramenta os profissionais especialistas em análise de negócio, foi importante que essa não exigisse conhecimentos específicos de verificação de modelos, linguagens de programação ou lógica temporal. Também foi importante que ela não exigisse esforços de adaptação das modelagens já existentes e utilizasse como base uma linguagem difundida e adequada aos propósitos do *software*. Nesse sentido, mesmo com objetivos próximos aos de outras pesquisas, foi possível inovar ao trazer esses vários aspectos conjugados a uma ferramenta que estará disponível em código aberto.

Também se destaca a identificação dos casos de tradução, representando as possibilidades de variação de comportamento da execução de fluxos de trabalho e as validações possíveis. Para o objetivo do presente trabalho, estes se mostraram suficientes e adequados. Toda a lógica de tradução apresentada pela abordagem foi considerada no desenvolvimento do *software*, demonstrando a viabilidade desta abordagem e sua capacidade de automatização.

Por meio dos testes realizados e por meio da validação do estudo de caso foi pos-

sível testar a eficácia da ferramenta e o alcance dos objetivos propostos. Foi possível não somente detectar os erros propositalmente simulados no fluxo, mas também encontrar inconsistência na lógica do processo que se manteve mesmo após uma série de revisões pelos responsáveis. Percebeu-se também que o *software* pode ser usado para a verificação de adequação entre modelagens e suas respectivas alterações/traduções para fins de orquestração ou automatização.

A maior ressalva quanto à aplicabilidade da ferramenta se refere à possibilidade de execução infinita de ciclos não encapsulados. Não se impôs um padrão de modelagem específico para a abordagem mas adotou-se na verdade um método já existente nos padrões de notação e especificação de fluxos utilizados neste trabalho. Apesar desse recurso não estar presente no caso estudado, existe uma forma padronizada de lidar com essa interpretação promovida pela abordagem deste trabalho. A não utilização do recurso pode ser atribuída a uma incorreção da modelagem ou até mesmo à não necessidade de interrupção dos ciclos do fluxo.

Uma outra forma de tratar a limitação seria a definição de um elemento específico nas linguagens envolvidas para a representação de objetos de decisão com contadores de ciclo que limitem ciclos não encapsulados. Essa possibilidade demandaria o reconhecimento da aplicabilidade da criação de tal elemento considerando o uso geral das linguagens. Além disso, a adoção de tal elemento representaria no curto prazo o esforço de adaptação das modelagens legadas.

Uma limitação da abordagem é a inviabilidade de incorporação de outras validações usando a estrutura proposta, como descrito na seção 4.2. Outra limitação nesse sentido é a restrição do número de relações testáveis entre os elementos dos fluxos na ferramenta. Esta decorre da própria criação da interface, voltada para simplificar o uso pelos usuários não especialistas. Não são necessários conhecimentos de lógica temporal e verificação de modelos para obter a validação das regras de execução de fluxos de trabalho pela ferramenta, o que se mostra como um dos benefícios da abordagem.

De modo a ampliar a contribuição deste trabalho, a ferramenta será disponibilizada de maneira irrestrita para que análises para a execução de possíveis correções e melhorias na mesma sejam possíveis assim como o próprio uso da mesma. Alguns trabalhos futuros se mostram viáveis:

- Estudo da viabilidade de utilização de outras linguagens de especificação de fluxo de trabalho ou orquestração de processos e também de outros verificadores;

- Inclusão de tradução para elementos do tipo complexo, limitação identificada na seção 4.1;
- Inclusão de validações não incluídas neste trabalho, identificadas na seção 4.2;
- Estudo da viabilidade de transformação da ferramenta em um *plugin* para ferramentas de modelagem de fluxos que permita a exibição dos resultados de validações (mais precisamente os contraexemplos) por meio de diagramas;
- Incorporação de métodos de simulação de tempo de execução de fluxos, de forma a possibilitar a realização de cálculos de tempo previsto ou consumo de outro tipo de recursos para a execução de vários processos concorrentes;
- Aplicação de testes de usabilidade do *software* em especialistas de negócios para detecção de sua viabilidade prática e possíveis melhorias, assim como testes de correção de *software*.

REFERÊNCIAS

AALST, W. M. P. van der. Verification of workflow nets. In: AZéMA, G. B. P. (Ed.). *Application and theory of Petri Nets 1997, LNCS volume 1248*. [S.l.]: Springer, 1997, (LNCS, v. 1248).

AGUILAR-SAVéN, R. S. Business process modelling: Review and framework. *International Journal of Production Economics - Volume 90 - Issue 2 - 28 July 2004*, v. 90, p. 129–149, 2004.

AN, L.; JENG, J.-J. On developing system dynamics model for business process simulation. *Proceedings of the 2005 Winter Simulation Conference*, 2005.

ANDRADE, A. et al. Um estudo de aplicação de modelagem de processo de negócio para apoiar a especificação de requisitos de um sistema. *VI Simpósio Internacional de Melhoria de Processos de Software, São Paulo - SP, Brasil. 24-26/11/2004*, 2004.

BAKER, B. *Business Modeling with UML: The Light at the End of the Tunnel*. http://www.ibm.com/developerworks/rational/library/content/RationalEdge/archives/dec01.html: [s.n.], December 2001.

CHAN, W. et al. Model checking large software specifications. *IEEE Transactions on software engineering, vol. 24, no. 7, July 1998*, 1998.

CLARKE, E. M. *Model checking*. [S.l.]: MIT Press, 1999.

CONERY, J. S.; CATCHEN, J. M.; LYNCH, M. Rule-based workflow management for bioinformatics. *The VLDB Journal - The International Journal on Very Large Data Bases, Volume 14, Issue 3 - September 2005*, 2005.

ERIKSSON, H.-E.; PENKER, M. *Business Modeling with UML: Business Patterns at Work*. [S.l.]: John Wiley & Sons©, 2000.

ESHUIS, H. *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*. Tese (CTIT Ph.D.-thesis series No. 02-44) — University of Twente, 2002. Disponível em: <http://www.ctit.utwente.nl/library/phd/2002/>.

ESHUIS, H.; WIERINGA, R. *A formal semantics for UML Activity Diagrams - Formalising workflow models*. [S.l.], 2001.

FISTEUS, J. A.; LOPEZ, A. M. Verbus: A formal model for business process verification. *Information Resources Management Association International Conference. New Orleans, Louisiana, USA*, 2004.

GODEFROID, P.; HUTH, M. Model checking vs. generalized model checking: semantic minimizations for temporal logics. *Procedings of the 20th annual symposium on logic in Computer Science (LICS'05) - IEEE*, 2005.

GRUHN, V.; LAUE, R. Using timed model checking for verifying workflows. *CSAC 2006 - Computer Supported Activity Coordination*, 2006.

GUELFI, N.; MAMMAR, A. *A Formal Approach for the Verification of E-Business Processes with PROMELA*. [S.l.], 2004.

GUELFI, N.; MAMMAR, A. A formal framework to generate xpdl specifications from uml activity diagrams. *21st Annual ACM Symposium on Applied Computing*, 2006.

HALLER, A.; GAALOUL, W.; MATEUSZ, M. Towards an xpdl compliant process ontology. *2008 IEEE Congress on Services*, 2008.

HAPPEL, H.-J.; STOJANOVIC, L. Ontoprocess - a prototype for semantic business process verification using swrl rules. *3rd European Semantic Web Conference 2006 - ESWC 2006*, 2006.

HUANG, Y.-H. *Using model checking to verify the consistency between business process models*. [S.l.], 2006.

INTERNATIONAL INSTITUTE OF BUSINESS ANALYSIS. *A guide to the Business Analysis Body of Knowledge (BABOK Guide) - Version 2.0*. [S.l.], 2009.

JANSSEN, W. et al. Model checking for managers. *Proceedings of the 5th and 6th International SPIN Workshops on Theoretical and Practical Aspects of SPIN Model Checking*, 1999.

JIANG, P.; MAIR, Q.; NEWMAN, J. Using uml to design distributed collaborative workflows: from uml to xpdl. *Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE-03)*, 2003.

KEEN, P.; QURESHI, S. Organizational transformation through business models: a framework for business model design. *HICSS39 - Procedings of the 39th Hawaii International Conference on System Sciences - IEEE*, 2006.

KHöLER, J.; TIRENNI, G.; KUMARAN, S. From business process model to consistent implementation: A case for formal verification methods. *6th International Enterprise Distributed Object Computing Conference (EDOC 2002)*, 2002.

MARKOSIAN, L. Z.; MANSOURI-SAMANI, M.; MEHLITZ, P. C. Program model checking using design-for-verification: Nasa flight software case study. *IEEEAC 06 - IEEE Guide for AC Motor Protection*, 2006.

MATOUSEK, P. *Verification of business process models*. [S.l.], 2003.

MUEHLEN, M. zur; INDULSKA, M.; KAMP, G. Business process and business rule modeling languages for compliance management: A representational analysis. *Twenty-Sixth International Conference on Conceptual Modeling - ER*, 2007.

OMG. *Business Process Modeling Notation (BPMN) Specification, version 1.2*. [S.l.], 01 2009. Especificação da notação BPMN. Disponível em: <www.bpmn.org>.

PETRUCCI, V. T.; CARVALHO, R. A. de; FELIX, M. F. Verificação formal de workflow. *Revista eletrônica de Iniciação Científica, vol. 6, no. 2, Junho 2006*, 2006.

PFEIFFER, J.-H.; ROSSAK, W. R.; SPECK, A. Applying model checking to workflow verification. *Proceedings of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, 2004.

RUSSEL, N. et al. On the suitability of uml 2.0 activity diagrams for business process modelling. *3rd Asia-Pacific Conference on Conceptual Modelling (APCCM2006)*, 2006.

SANTOS, C.; FERREIRA, C.; TRIBOLET, J. Modelação e verificação formal de processos de negócio em ambiente hospitalar. *7ª Conferência da Associação Portuguesa de Sistemas de Informação, Jan. 2007*, 2007.

SASAKI, S.; IIJIMA, J. Verification of invariant properties of business process based on formal approach. *Wireless Communications, Networking and Mobile Computing*, 2007.

SCHAAD, A.; SOHR, K.; DROUINEAUD, M. A workflow-based model-checking approach to inter and intra-analysis of organisational controls in service-oriented bussiness processes. *Journal of Information Assurance and Security, vol. 2, iss. 1*, 2007.

SREEMANI, T.; ATLEE, J. M. Feasibility of model checking software requirements: a case study. *COMPASS'96, Proceedings of the 11th Annual Conference on Computer Assurance*, 1996.

TAKEMURA, T. Formal semantics and verification of bpmn transaction and compensation. *2008 IEEE Asia-Pacific Services Computing Conference*, 2008.

TSAI, C.-H.; LUO, H.-J.; WANG, F.-J. Constructing a bpm environment with bpmn*. *Proceedings of the 11th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'07)*, 2007.

WFMC. *XPDL 2.1 Specification*. Wfmc-tc-1025-oct-10-08-a. [S.l.], 10 2008. Disponível em: <http://www.wfmc.org/View-document-details/WFMC-TC-1025-Oct-10-08-A-Final-XPDL-2.1-Specification.html>.

WHITE, S. A. *Process Modeling Notations and Workflow Patterns*. [S.l.], January 2004. Disponível em: <http://www.bpmn.org>.

ZSIFKOV, N.; CAMPEANU, R. Business rules domains and business rules modeling. *International Symposium on Information and Communication Technologies*, 2004.

ANEXO A – FLUXOS DO ESTUDO DE CASO

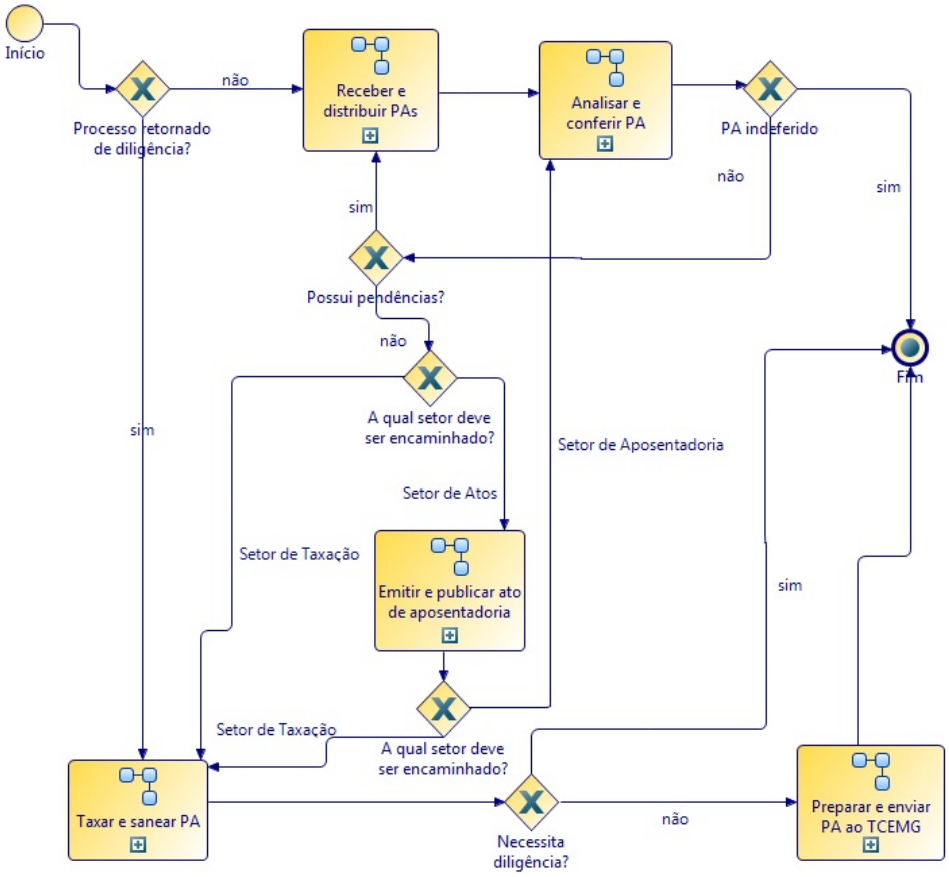

Figura 28: Fluxo de trabalho mais abstrato do estudo de caso

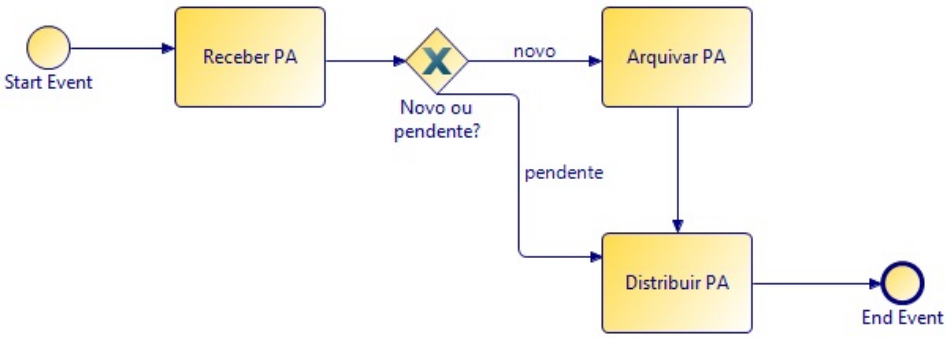

Figura 29: Fluxo "Receber e distribuir PA"

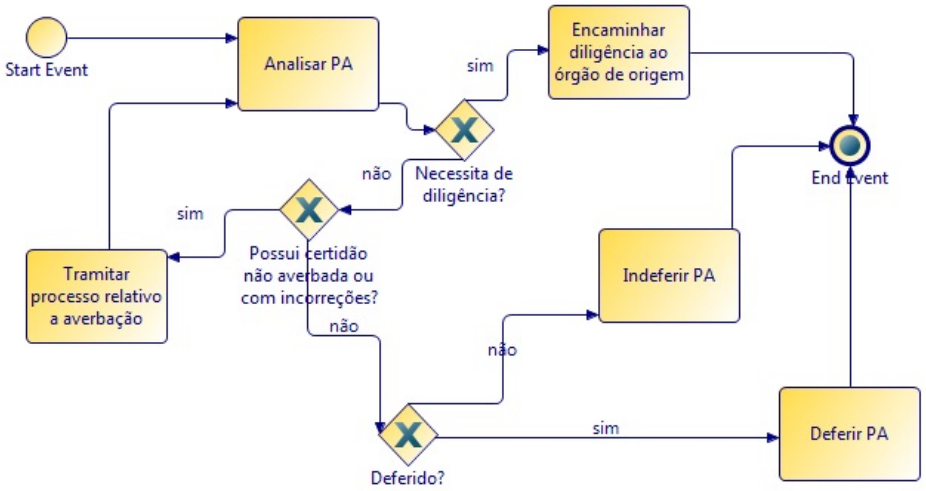

Figura 30: Fluxo "Analisar e conferir PA"

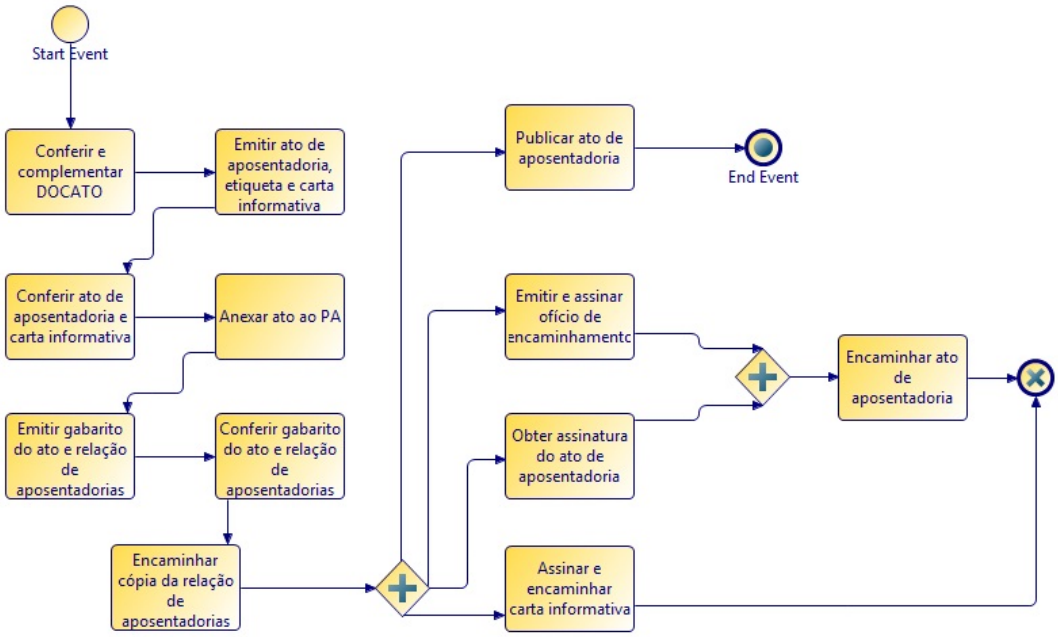

Figura 31: Fluxo "Emitir e publicar ato de aposentadoria"

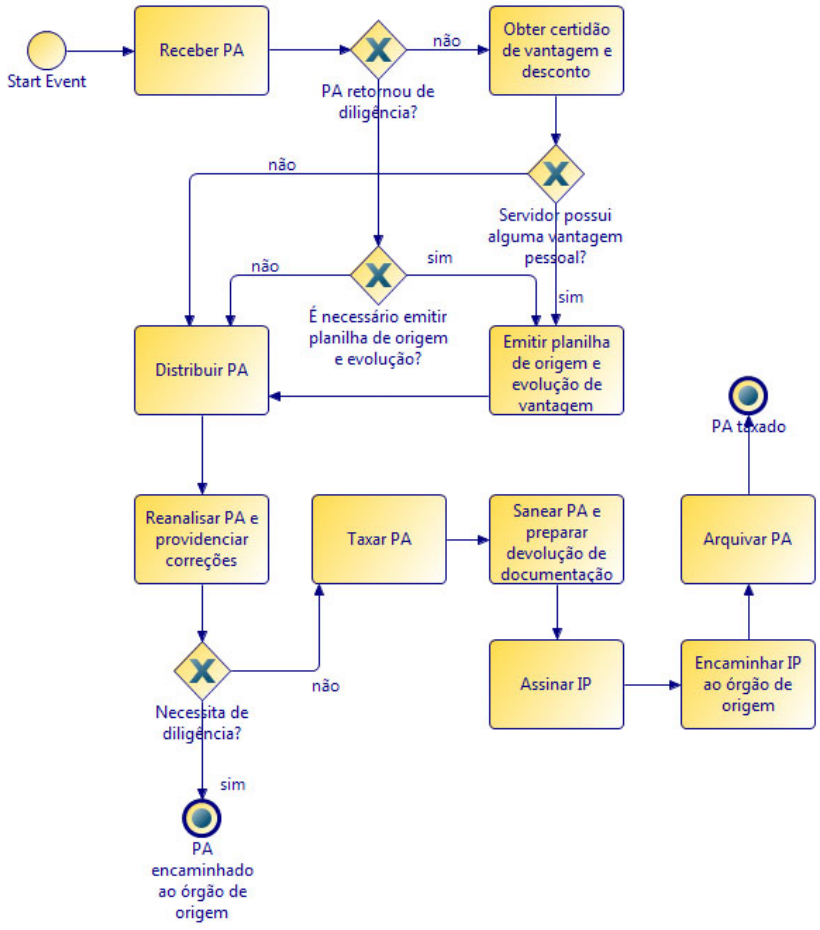

Figura 32: Fluxo "Taxar e sanear PA"

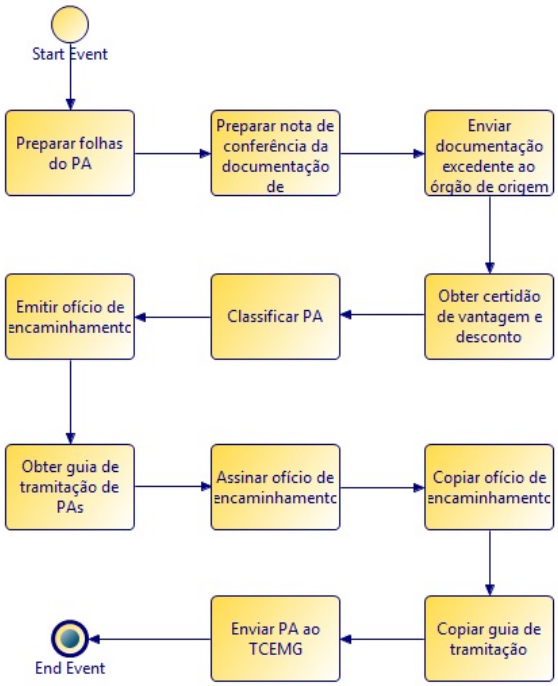

Figura 33: Fluxo “Preparar e enviar PA ao TCEMG”

ANEXO B – TRADUÇÃO COMPLETA DO FLUXO DO ESTUDO DE CASO

```
1  MODULE main
2  VAR
3        atividade0: boolean; -- Nó inicial (_ebgdotQ4Ed6XPO7ah7EKDA) - processo 'Principal'
             (_d5umQNQ4Ed6XPO7ah7EKDA)
4        atividade1: boolean; -- Nó final - terminate (_ebgdo9Q4Ed6XPO7ah7EKDA) - processo '
             Principal' (_d5umQNQ4Ed6XPO7ah7EKDA)
5        -- atividade2: Receber e distribuir PA (_j2Yx8NQ4Ed6XPO7ah7EKDA) - subflow (
             _YEo60NQ7Ed66BtkiytxsJQ) - processo 'Principal' (_d5umQNQ4Ed6XPO7ah7EKDA)
6        -- atividade3: Analisar e conferir PA (_m8qmYNQ4Ed6XPO7ah7EKDA) - subflow (_A5MnkNQ-
             Ed66BtkiytxsJQ) - processo 'Principal' (_d5umQNQ4Ed6XPO7ah7EKDA)
7        -- atividade4: Emitir e publicar ato de aposentadoria (_pTf64NQ4Ed6XPO7ah7EKDA) -
             subflow (_dAQjcNQ_Ed66BtkiytxsJQ) - processo 'Principal' (
             _d5umQNQ4Ed6XPO7ah7EKDA)
8        -- atividade5: Taxar e sanear PA (_s1QKYNQ4Ed6XPO7ah7EKDA) - subflow (
             _HX7eMNRBEd66BtkiytxsJQ) - processo 'Principal' (_d5umQNQ4Ed6XPO7ah7EKDA)
9        -- atividade6: Preparar e enviar PA ao TCEMG (_vKaD4NQ4Ed6XPO7ah7EKDA) - subflow (
             _Sz0rANRHEd66BtkiytxsJQ) - processo 'Principal' (_d5umQNQ4Ed6XPO7ah7EKDA)
10       -- atividade7: Processo retornado de diligência? (_4vMz4NQ4Ed6XPO7ah7EKDA) -
             processo 'Principal' (_d5umQNQ4Ed6XPO7ah7EKDA)
11       -- atividade8: A qual setor deve ser encaminhado? (_RQ2kwNQ5Ed6XPO7ah7EKDA) -
             processo 'Principal' (_d5umQNQ4Ed6XPO7ah7EKDA)
12       -- atividade9: A qual setor deve ser encaminhado? (_dwCCQNQ5Ed6XPO7ah7EKDA) -
             processo 'Principal' (_d5umQNQ4Ed6XPO7ah7EKDA)
13       -- atividade10: Possui pendências? (_fKbQwNRGEd66BtkiytxsJQ) - processo 'Principal'
             (_d5umQNQ4Ed6XPO7ah7EKDA)
14       -- atividade11: Necessita diligência? (_t6ImcNRGEd66BtkiytxsJQ) - processo '
             Principal' (_d5umQNQ4Ed6XPO7ah7EKDA)
15       -- atividade12: PA indeferido (_TN8s0NqKEd6Hr6WkTj-iWw) - processo 'Principal' (
             _d5umQNQ4Ed6XPO7ah7EKDA)
16       atividade13: boolean; -- Nó inicial (_Yb5kQdQ7Ed66BtkiytxsJQ) - processo 'Receber e
             distribuir PA' (_YEo60NQ7Ed66BtkiytxsJQ)
17       atividade14: boolean; -- Nó final - terminate (_Yb5kQtQ7Ed66BtkiytxsJQ) - processo '
             Receber e distribuir PA' (_YEo60NQ7Ed66BtkiytxsJQ)
18       atividade15: boolean; -- Receber PA (_e2fpINQ7Ed66BtkiytxsJQ) - processo 'Receber e
             distribuir PA' (_YEo60NQ7Ed66BtkiytxsJQ)
19       atividade16: boolean; -- Arquivar PA (_gW32wNQ7Ed66BtkiytxsJQ) - processo 'Receber e
              distribuir PA' (_YEo60NQ7Ed66BtkiytxsJQ)
20       atividade17: boolean; -- Distribuir PA (_ir83wNQ7Ed66BtkiytxsJQ) - processo 'Receber
              e distribuir PA' (_YEo60NQ7Ed66BtkiytxsJQ)
21       -- atividade18: Novo ou pendente? (_sbzBMNQ7Ed66BtkiytxsJQ) - processo 'Receber e
             distribuir PA' (_YEo60NQ7Ed66BtkiytxsJQ)
```

```
22      atividade19: boolean; -- Nó inicial (_BMk2ktQ-Ed66BtkiytxsJQ) - processo 'Analisar e
            conferir PA' (_A5MnkNQ-Ed66BtkiytxsJQ)
23      atividade20: boolean; -- Nó final - terminate (_BMk2k9Q-Ed66BtkiytxsJQ) - processo '
            Analisar e conferir PA' (_A5MnkNQ-Ed66BtkiytxsJQ)
24      atividade21: boolean; -- Analisar PA (_CbQQMNQ-Ed66BtkiytxsJQ) - processo 'Analisar
            e conferir PA' (_A5MnkNQ-Ed66BtkiytxsJQ)
25      atividade22: boolean; -- Encaminhar diligência ao órgão de origem (_EwQYsNQ-
            Ed66BtkiytxsJQ) - processo 'Analisar e conferir PA' (_A5MnkNQ-Ed66BtkiytxsJQ)
26      atividade23: boolean; -- Indeferir PA (_CSDbcNQ_Ed66BtkiytxsJQ) - processo 'Analisar
            e conferir PA' (_A5MnkNQ-Ed66BtkiytxsJQ)
27      atividade24: boolean; -- Deferir PA (_EJsSgNQ_Ed66BtkiytxsJQ) - processo 'Analisar e
            conferir PA' (_A5MnkNQ-Ed66BtkiytxsJQ)
28      atividade25: boolean; -- Tramitar processo relativo a averbação (
            _J7yM8NQ_Ed66BtkiytxsJQ) - processo 'Analisar e conferir PA' (_A5MnkNQ-
            Ed66BtkiytxsJQ)
29      -- atividade26: Possui certidão não averbada ou com incorreções? (_O-7
            LINRJEd66BtkiytxsJQ) - processo 'Analisar e conferir PA' (_A5MnkNQ-
            Ed66BtkiytxsJQ)
30      -- atividade27: Necessita de diligência? (_ptHsENRJEd66BtkiytxsJQ) - processo '
            Analisar e conferir PA' (_A5MnkNQ-Ed66BtkiytxsJQ)
31      -- atividade28: Deferido? (_axpp8NRXEd66BtkiytxsJQ) - processo 'Analisar e conferir
            PA' (_A5MnkNQ-Ed66BtkiytxsJQ)
32      atividade29: boolean; -- Nó inicial (_dWG3otQ_Ed66BtkiytxsJQ) - processo 'Emitir e
            publicar ato de aposentadoria' (_dAQjcNQ_Ed66BtkiytxsJQ)
33      atividade30: boolean; -- Nó final - terminate (_dWG3o9Q_Ed66BtkiytxsJQ) - processo '
            Emitir e publicar ato de aposentadoria' (_dAQjcNQ_Ed66BtkiytxsJQ)
34      atividade31: boolean; -- Conferir e complementar DOCATO (_eCFz4NQ_Ed66BtkiytxsJQ) -
            processo 'Emitir e publicar ato de aposentadoria' (_dAQjcNQ_Ed66BtkiytxsJQ)
35      atividade32: boolean; -- Emitir ato de aposentadoria, etiqueta e carta informativa (
            _jTTO0NQ_Ed66BtkiytxsJQ) - processo 'Emitir e publicar ato de aposentadoria' (
            _dAQjcNQ_Ed66BtkiytxsJQ)
36      atividade33: boolean; -- Conferir ato de aposentadoria e carta informativa (
            _o54c0NQ_Ed66BtkiytxsJQ) - processo 'Emitir e publicar ato de aposentadoria' (
            _dAQjcNQ_Ed66BtkiytxsJQ)
37      atividade34: boolean; -- Anexar ato ao PA (_r9zg0NQ_Ed66BtkiytxsJQ) - processo '
            Emitir e publicar ato de aposentadoria' (_dAQjcNQ_Ed66BtkiytxsJQ)
38      atividade35: boolean; -- Emitir gabarito do ato e relação de aposentadorias (
            _4JmIQNQ_Ed66BtkiytxsJQ) - processo 'Emitir e publicar ato de aposentadoria' (
            _dAQjcNQ_Ed66BtkiytxsJQ)
39      atividade36: boolean; -- Conferir gabarito do ato e relação de aposentadorias (_-
            DGSQNQ_Ed66BtkiytxsJQ) - processo 'Emitir e publicar ato de aposentadoria' (
            _dAQjcNQ_Ed66BtkiytxsJQ)
40      atividade37: boolean; -- _FLyMMNRAEd66BtkiytxsJQ (nó final - cancel) (
            _FLyMMNRAEd66BtkiytxsJQ) - cancel - processo 'Emitir e publicar ato de
            aposentadoria' (_dAQjcNQ_Ed66BtkiytxsJQ)
41      atividade38: boolean; -- Encaminhar cópia da relação de aposentadorias (
            _IIZHQNRAEd66BtkiytxsJQ) - processo 'Emitir e publicar ato de aposentadoria' (
            _dAQjcNQ_Ed66BtkiytxsJQ)
42      -- atividade39: _LnY9sNRAEd66BtkiytxsJQ (_LnY9sNRAEd66BtkiytxsJQ) - and - processo '
            Emitir e publicar ato de aposentadoria' (_dAQjcNQ_Ed66BtkiytxsJQ)
43      atividade40: boolean; -- Assinar e encaminhar carta informativa (
            _OgQBMNRAEd66BtkiytxsJQ) - processo 'Emitir e publicar ato de aposentadoria' (
            _dAQjcNQ_Ed66BtkiytxsJQ)
```

```
44      atividade41: boolean; -- Obter assinatura do ato de aposentadoria (
            _Ss9SMNRAEd66BtkiytxsJQ) - processo 'Emitir e publicar ato de aposentadoria' (
            _dAQjcNQ_Ed66BtkiytxsJQ)
45      atividade42: boolean; -- Publicar ato de aposentadoria (_cqYkoNRAEd66BtkiytxsJQ) -
            processo 'Emitir e publicar ato de aposentadoria' (_dAQjcNQ_Ed66BtkiytxsJQ)
46      atividade43: boolean; -- Emitir e assinar ofício de encaminhamento (
            _qhNxkNRAEd66BtkiytxsJQ) - processo 'Emitir e publicar ato de aposentadoria' (
            _dAQjcNQ_Ed66BtkiytxsJQ)
47      -- atividade44: _twqJENRAEd66BtkiytxsJQ (_twqJENRAEd66BtkiytxsJQ) - and - processo '
            Emitir e publicar ato de aposentadoria' (_dAQjcNQ_Ed66BtkiytxsJQ)
48      atividade45: boolean; -- Encaminhar ato de aposentadoria (_11ZbkNRAEd66BtkiytxsJQ) -
             processo 'Emitir e publicar ato de aposentadoria' (_dAQjcNQ_Ed66BtkiytxsJQ)
49      atividade46: boolean; -- Nó inicial (_HyzcUtRBEd66BtkiytxsJQ) - processo 'Taxar e
            sanear PA' (_HX7eMNRBEd66BtkiytxsJQ)
50      atividade47: boolean; -- PA taxado (nó final - terminate) (_HyzcU9RBEd66BtkiytxsJQ)
            - processo 'Taxar e sanear PA' (_HX7eMNRBEd66BtkiytxsJQ)
51      atividade48: boolean; -- Receber PA (_M2nUcNRBEd66BtkiytxsJQ) - processo 'Taxar e
            sanear PA' (_HX7eMNRBEd66BtkiytxsJQ)
52      atividade49: boolean; -- Obter certidão de vantagem e desconto (
            _Oi86cNRBEd66BtkiytxsJQ) - processo 'Taxar e sanear PA' (_HX7eMNRBEd66BtkiytxsJQ
            )
53      atividade50: boolean; -- Distribuir PA (_WRWwkNRBEd66BtkiytxsJQ) - processo 'Taxar e
             sanear PA' (_HX7eMNRBEd66BtkiytxsJQ)
54      atividade51: boolean; -- Emitir planilha de origem e evolução de vantagem pessoal (
            _QbVtgNRDEd66BtkiytxsJQ) - processo 'Taxar e sanear PA' (_HX7eMNRBEd66BtkiytxsJQ
            )
55      atividade52: boolean; -- Reanalisar PA e providenciar correções (
            _T7kTANRDEd66BtkiytxsJQ) - processo 'Taxar e sanear PA' (_HX7eMNRBEd66BtkiytxsJQ
            )
56      atividade53: boolean; -- Taxar PA (_W0avcNRDEd66BtkiytxsJQ) - processo 'Taxar e
            sanear PA' (_HX7eMNRBEd66BtkiytxsJQ)
57      atividade54: boolean; -- Sanear PA e preparar devolução de documentação excedente (
            _Yo2hgNRDEd66BtkiytxsJQ) - processo 'Taxar e sanear PA' (_HX7eMNRBEd66BtkiytxsJQ
            )
58      -- atividade55: PA retornou de diligência? (_sstRYNRDEd66BtkiytxsJQ) - processo '
            Taxar e sanear PA' (_HX7eMNRBEd66BtkiytxsJQ)
59      -- atividade56: É necessário emitir planilha de origem e evolução? (_5-
            w9UNRDEd66BtkiytxsJQ) - processo 'Taxar e sanear PA' (_HX7eMNRBEd66BtkiytxsJQ)
60      -- atividade57: Servidor possui alguma vantagem pessoal? (_GTpPUNREEd66BtkiytxsJQ) -
             processo 'Taxar e sanear PA' (_HX7eMNRBEd66BtkiytxsJQ)
61      atividade58: boolean; -- PA encaminhado ao órgão de origem (nó final - terminate) (
            _mz2LINREEd66BtkiytxsJQ) - processo 'Taxar e sanear PA' (_HX7eMNRBEd66BtkiytxsJQ
            )
62      atividade59: boolean; -- Assinar IP (_A919YNRGEd66BtkiytxsJQ) - processo 'Taxar e
            sanear PA' (_HX7eMNRBEd66BtkiytxsJQ)
63      atividade60: boolean; -- Encaminhar IP ao órgão de origem (_ETUrcNRGEd66BtkiytxsJQ)
            - processo 'Taxar e sanear PA' (_HX7eMNRBEd66BtkiytxsJQ)
64      atividade61: boolean; -- Arquivar PA (_Gqwc4NRGEd66BtkiytxsJQ) - processo 'Taxar e
            sanear PA' (_HX7eMNRBEd66BtkiytxsJQ)
65      -- atividade62: Necessita de diligência? (_QbcT0NRGEd66BtkiytxsJQ) - processo 'Taxar
             e sanear PA' (_HX7eMNRBEd66BtkiytxsJQ)
66      atividade63: boolean; -- Nó inicial (_THGzZ9RHEd66BtkiytxsJQ) - processo 'Preparar e
             enviar PA ao TCEMG' (_Sz0rANRHEd66BtkiytxsJQ)
```

```
67      atividade64: boolean; -- Nó final - terminate (_THGzaNRHEd66BtkiytxsJQ) - processo '
            Preparar e enviar PA ao TCEMG' (_Sz0rANRHEd66BtkiytxsJQ)
68      atividade65: boolean; -- Preparar folhas do PA (_a7FtENRHEd66BtkiytxsJQ) - processo
            'Preparar e enviar PA ao TCEMG' (_Sz0rANRHEd66BtkiytxsJQ)
69      atividade66: boolean; -- Preparar nota de conferência da documentação de
            aposentadoria (_cc5eENRHEd66BtkiytxsJQ) - processo 'Preparar e enviar PA ao
            TCEMG' (_Sz0rANRHEd66BtkiytxsJQ)
70      atividade67: boolean; -- Enviar documentação excedente ao órgão de origem (
            _f7ZlQNRHEd66BtkiytxsJQ) - processo 'Preparar e enviar PA ao TCEMG' (
            _Sz0rANRHEd66BtkiytxsJQ)
71      atividade68: boolean; -- Obter certidão de vantagem e desconto (
            _lvYWENRHEd66BtkiytxsJQ) - processo 'Preparar e enviar PA ao TCEMG' (
            _Sz0rANRHEd66BtkiytxsJQ)
72      atividade69: boolean; -- Classificar PA (_oWTVANRHEd66BtkiytxsJQ) - processo '
            Preparar e enviar PA ao TCEMG' (_Sz0rANRHEd66BtkiytxsJQ)
73      atividade70: boolean; -- Emitir ofício de encaminhamento (_p6j9ENRHEd66BtkiytxsJQ) -
             processo 'Preparar e enviar PA ao TCEMG' (_Sz0rANRHEd66BtkiytxsJQ)
74      atividade71: boolean; -- Obter guia de tramitação de PAs (_tg9cANRHEd66BtkiytxsJQ) -
             processo 'Preparar e enviar PA ao TCEMG' (_Sz0rANRHEd66BtkiytxsJQ)
75      atividade72: boolean; -- Assinar ofício de encaminhamento (_1ehIcNRHEd66BtkiytxsJQ)
            - processo 'Preparar e enviar PA ao TCEMG' (_Sz0rANRHEd66BtkiytxsJQ)
76      atividade73: boolean; -- Copiar ofício de encaminhamento (_4At_cNRHEd66BtkiytxsJQ) -
             processo 'Preparar e enviar PA ao TCEMG' (_Sz0rANRHEd66BtkiytxsJQ)
77      atividade74: boolean; -- Enviar PA ao TCEMG (_DKNcYNRIEd66BtkiytxsJQ) - processo '
            Preparar e enviar PA ao TCEMG' (_Sz0rANRHEd66BtkiytxsJQ)
78      atividade75: boolean; -- Copiar guia de tramitação (_DpQv4NRIEd66BtkiytxsJQ) -
            processo 'Preparar e enviar PA ao TCEMG' (_Sz0rANRHEd66BtkiytxsJQ)
79
80  ASSIGN
81      init(atividade0) := 1;
82      init(atividade1) := 0;
83      init(atividade13) := 0;
84      init(atividade14) := 0;
85      init(atividade15) := 0;
86      init(atividade16) := 0;
87      init(atividade17) := 0;
88      init(atividade19) := 0;
89      init(atividade20) := 0;
90      init(atividade21) := 0;
91      init(atividade22) := 0;
92      init(atividade23) := 0;
93      init(atividade24) := 0;
94      init(atividade25) := 0;
95      init(atividade29) := 0;
96      init(atividade30) := 0;
97      init(atividade31) := 0;
98      init(atividade32) := 0;
99      init(atividade33) := 0;
100     init(atividade34) := 0;
101     init(atividade35) := 0;
102     init(atividade36) := 0;
103     init(atividade37) := 0;
104     init(atividade38) := 0;
105     init(atividade40) := 0;
```

```
106     init(atividade41) := 0;
107     init(atividade42) := 0;
108     init(atividade43) := 0;
109     init(atividade45) := 0;
110     init(atividade46) := 0;
111     init(atividade47) := 0;
112     init(atividade48) := 0;
113     init(atividade49) := 0;
114     init(atividade50) := 0;
115     init(atividade51) := 0;
116     init(atividade52) := 0;
117     init(atividade53) := 0;
118     init(atividade54) := 0;
119     init(atividade58) := 0;
120     init(atividade59) := 0;
121     init(atividade60) := 0;
122     init(atividade61) := 0;
123     init(atividade63) := 0;
124     init(atividade64) := 0;
125     init(atividade65) := 0;
126     init(atividade66) := 0;
127     init(atividade67) := 0;
128     init(atividade68) := 0;
129     init(atividade69) := 0;
130     init(atividade70) := 0;
131     init(atividade71) := 0;
132     init(atividade72) := 0;
133     init(atividade73) := 0;
134     init(atividade74) := 0;
135     init(atividade75) := 0;
136
137     next(atividade0) := 1;
138     next(atividade13) :=
139                 case
140                         atividade13=1: 1;
141                         atividade0=1 & ( atividade46=1 ): 0;
142                         atividade0=1: {0,1};
143                         1: 0;
144     esac;
145     next(atividade46) :=
146                 case
147                         atividade46=1: 1;
148                         atividade0=1 & ( atividade13=1 | next(atividade13)=1 ): 0;
149                         atividade0=1 & atividade13=0: 1;
150                         1: 0;
151     esac;
152     next(atividade48) :=
153                 case
154                         atividade46=1: 1;
155                         1: 0;
156     esac;
157     next(atividade49) :=
158                 case
159             atividade49=1: 1;
```

```
160                         atividade48=1: {0,1};
161                         1: 0;
162         esac;
163         next(atividade50) :=
164                     case
165                             atividade50=1: 1;
166                         (atividade48=1): 1;
167                             1: 0;
168         esac;
169         next(atividade52) :=
170                     case
171                             atividade52=1: 1;
172                             atividade50=1: 1;
173                             1: 0;
174         esac;
175         next(atividade51) :=
176                     case
177                atividade51=1: 1;
178                             atividade48=1: {0,1};
179                             1: 0;
180         esac;
181         next(atividade53) :=
182                     case
183                             atividade53=1: 1;
184                             atividade52=1 & ( atividade58=1 ): 0;
185                             atividade52=1: {0,1};
186                             1: 0;
187         esac;
188         next(atividade58) :=
189                     case
190                             atividade58=1: 1;
191                             atividade52=1 & ( atividade53=1 | next(atividade53)=1 ): 0;
192                             atividade52=1 & atividade53=0: 1;
193                             1: 0;
194         esac;
195         next(atividade1) :=
196                     case
197                             atividade1=1: 1;
198                             atividade47=1 & atividade58=1 & atividade64=1 & ( atividade63=1 )
                                    : 0;
199                             atividade47=1 & atividade58=1 & atividade64=1: {0,1};
200                             1: 0;
201         esac;
202         next(atividade63) :=
203                     case
204                             atividade63=1: 1;
205                             atividade47=1 & atividade58=1 & atividade64=1 & ( atividade1=1 |
                                    next(atividade1)=1 ): 0;
206                             atividade47=1 & atividade58=1 & atividade64=1 & atividade1=0: 1;
207                             1: 0;
208         esac;
209         next(atividade65) :=
210                     case
211                             atividade63=1: 1;
```

```
212                         1: 0;
213         esac;
214         next(atividade66) :=
215                     case
216                         atividade65=1: 1;
217                         1: 0;
218         esac;
219         next(atividade67) :=
220                     case
221                         atividade66=1: 1;
222                         1: 0;
223         esac;
224         next(atividade68) :=
225                     case
226                         atividade67=1: 1;
227                         1: 0;
228         esac;
229         next(atividade69) :=
230                     case
231                         atividade68=1: 1;
232                         1: 0;
233         esac;
234         next(atividade70) :=
235                     case
236                         atividade69=1: 1;
237                         1: 0;
238         esac;
239         next(atividade71) :=
240                     case
241                         atividade70=1: 1;
242                         1: 0;
243         esac;
244         next(atividade72) :=
245                     case
246                         atividade71=1: 1;
247                         1: 0;
248         esac;
249         next(atividade73) :=
250                     case
251                         atividade72=1: 1;
252                         1: 0;
253         esac;
254         next(atividade75) :=
255                     case
256                         atividade73=1: 1;
257                         1: 0;
258         esac;
259         next(atividade74) :=
260                     case
261                         atividade75=1: 1;
262                         1: 0;
263         esac;
264         next(atividade64) :=
265                     case
```

```
266                         atividade74=1: 1;
267                         1: 0;
268         esac;
269         next(atividade54) :=
270                     case
271                         atividade53=1: 1;
272                         1: 0;
273         esac;
274         next(atividade59) :=
275                     case
276                         atividade54=1: 1;
277                         1: 0;
278         esac;
279         next(atividade60) :=
280                     case
281                         atividade59=1: 1;
282                         1: 0;
283         esac;
284         next(atividade61) :=
285                     case
286                         atividade60=1: 1;
287                         1: 0;
288         esac;
289         next(atividade47) :=
290                     case
291                         atividade61=1: 1;
292                         1: 0;
293         esac;
294         next(atividade15) :=
295                     case
296                         atividade13=1: 1;
297                         1: 0;
298         esac;
299         next(atividade16) :=
300                     case
301               atividade16=1: 1;
302                         atividade15=1: {0,1};
303                         1: 0;
304         esac;
305         next(atividade17) :=
306                     case
307                         atividade17=1: 1;
308                     (atividade15=1): 1;
309                         1: 0;
310         esac;
311         next(atividade14) :=
312                     case
313                         atividade14=1: 1;
314                         atividade17=1: 1;
315                         1: 0;
316         esac;
317         next(atividade21) :=
318                     case
319                         atividade19=1: 1;
```

```
320                         1: 0;
321         esac;
322         next(atividade24) :=
323                     case
324                 atividade24=1: 1;
325                         atividade21=1 & ( atividade23=1 | atividade25=1 | atividade22=1 )
                                : 0;
326                         atividade21=1: {0,1};
327                         1: 0;
328         esac;
329         next(atividade23) :=
330                     case
331                 atividade23=1: 1;
332                         atividade21=1 & ( atividade24=1 | atividade25=1 | atividade22=1 )
                                : 0;
333                         atividade21=1 & ( atividade24=0 & atividade25=0 & atividade22=0 )
                                : 1;
334                         1: 0;
335         esac;
336         next(atividade25) :=
337                     case
338                 atividade25=1: 1;
339                         atividade21=1 & ( atividade24=1 | atividade23=1 | atividade22=1 )
                                : 0;
340                         atividade21=1 & ( atividade24=0 & atividade23=0 & atividade22=0 )
                                : 1;
341                         1: 0;
342         esac;
343         next(atividade22) :=
344                     case
345                 atividade22=1: 1;
346                         atividade21=1 & ( atividade24=1 | atividade23=1 | atividade25=1 )
                                : 0;
347                         atividade21=1 & ( atividade24=0 & atividade23=0 & atividade25=0 )
                                : 1;
348                         1: 0;
349         esac;
350         next(atividade31) :=
351                     case
352                         atividade29=1: 1;
353                         1: 0;
354         esac;
355         next(atividade32) :=
356                     case
357                         atividade31=1: 1;
358                         1: 0;
359         esac;
360         next(atividade33) :=
361                     case
362                         atividade32=1: 1;
363                         1: 0;
364         esac;
365         next(atividade34) :=
366                     case
```

```
367                         atividade33=1: 1;
368                         1: 0;
369         esac;
370         next(atividade35) :=
371                     case
372                         atividade34=1: 1;
373                         1: 0;
374         esac;
375         next(atividade36) :=
376                     case
377                         atividade35=1: 1;
378                         1: 0;
379         esac;
380         next(atividade38) :=
381                     case
382                         atividade36=1: 1;
383                         1: 0;
384         esac;
385         next(atividade40) :=
386                     case
387                         atividade38=1: 1;
388                         1: 0;
389         esac;
390         next(atividade41) :=
391                     case
392                         atividade38=1: 1;
393                         1: 0;
394         esac;
395         next(atividade42) :=
396                     case
397                         atividade38=1: 1;
398                         1: 0;
399         esac;
400         next(atividade43) :=
401                     case
402                         atividade38=1: 1;
403                         1: 0;
404         esac;
405         next(atividade19) :=
406                     case
407                         atividade19=1: 1;
408                         atividade30=1 & atividade37=1 & atividade14=1 & ( atividade46=1 )
                                : 0;
409                         atividade30=1 & atividade37=1 & atividade14=1: {0,1};
410                         1: 0;
411         esac;
412         next(atividade30) :=
413                     case
414                         atividade42=1: 1;
415                         1: 0;
416         esac;
417         next(atividade45) :=
418                     case
419                         atividade41=1 & atividade43=1: 1;
```

```
420                                 1: 0;
421         esac;
```

Código B.1: Tradução completa do estudo de caso

ANEXO C – FLUXO DE TRABALHO ADICIONAL - SISTEMA DE CAPACITAÇÃO

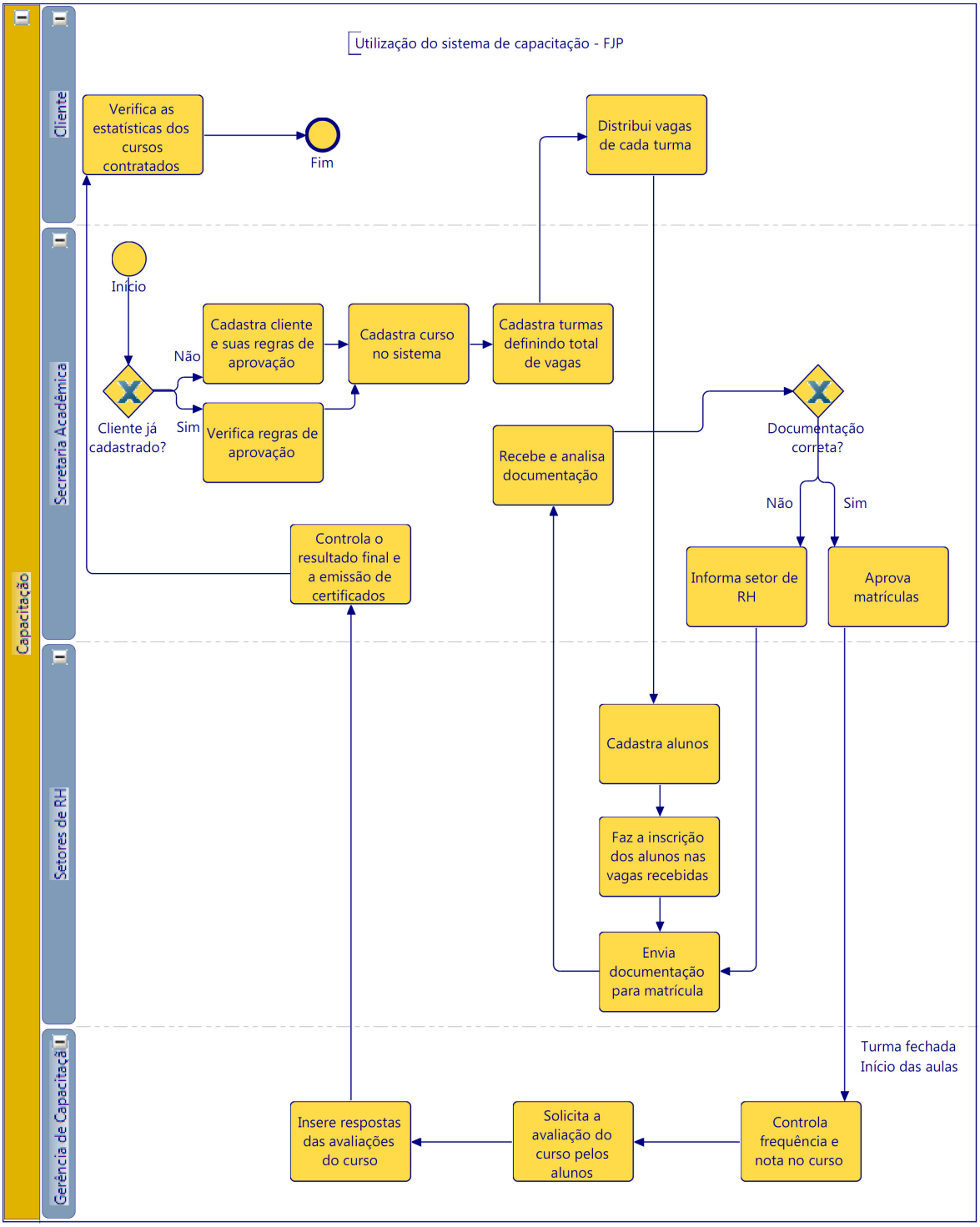

Figura 34: Fluxo “Utilização do sistema de capacitação - Fundação João Pinheiro”

```
1  MODULE main
2  VAR
3        atividade0: boolean; -- Nó inicial (_FQL7opivEd-nyvvtUKOMaQ)
4        atividade1: boolean; -- Fim (nó final) (_FQL7o5ivEd-nyvvtUKOMaQ)
5        atividade2: boolean; -- Cadastra curso no sistema (_nqiP8JivEd-nyvvtUKOMaQ)
6        atividade3: boolean; -- Cadastra cliente e suas regras de aprovação (_4HfKkJivEd-
             nyvvtUKOMaQ)
7        atividade4: boolean; -- Cadastra turmas definindo total de vagas (_HAyhIJiwEd-
             nyvvtUKOMaQ)
8        atividade5: boolean; -- Distribui vagas de cada turma (_Lmp5AJiwEd-nyvvtUKOMaQ)
9        atividade6: boolean; -- Cadastra alunos (_c2paIJiwEd-nyvvtUKOMaQ)
10       atividade7: boolean; -- Faz a inscrição dos alunos nas vagas recebidas (_qILhoJiwEd-
             nyvvtUKOMaQ)
11       atividade8: boolean; -- Envia documentação para matrícula (_uZoIsJiwEd-nyvvtUKOMaQ)
12       atividade9: boolean; -- Aprova matrículas (_07pEMJiwEd-nyvvtUKOMaQ)
13       atividade10: boolean; -- Controla frequência e nota no curso (_5NxLQJixEd-
             nyvvtUKOMaQ)
14       atividade11: boolean; -- Controla o resultado final e a emissão de certificados (
             _DPRscJiyEd-nyvvtUKOMaQ)
15       atividade12: boolean; -- Solicita a avaliação do curso pelos alunos (_Uni_0JiyEd-
             nyvvtUKOMaQ)
16       atividade13: boolean; -- Insere respostas das avaliações do curso (_hA_owJiyEd-
             nyvvtUKOMaQ)
17       atividade14: boolean; -- Verifica as estatísticas dos cursos contratados (
             _wB7RUJiyEd-nyvvtUKOMaQ)
18       atividade15: boolean; -- Recebe e analisa documentação (_FwGYgJi0Ed-nyvvtUKOMaQ)
19       -- atividade16: Documentação correta? (_ISOXAJi0Ed-nyvvtUKOMaQ)
20       atividade17: boolean; -- Informa setor de RH (_I-kDIJi1Ed-nyvvtUKOMaQ)
21       atividade18: boolean; -- Verifica regras de aprovação (_uh6vwJi1Ed-nyvvtUKOMaQ)
22       -- atividade19: Cliente já cadastrado? (_zwl0EJi1Ed-nyvvtUKOMaQ)
23
24 ASSIGN
25       init(atividade0) := 1;
26       init(atividade1) := 0;
27       init(atividade2) := 0;
28       init(atividade3) := 0;
29       init(atividade4) := 0;
30       init(atividade5) := 0;
31       init(atividade6) := 0;
32       init(atividade7) := 0;
33       init(atividade8) := 0;
34       init(atividade9) := 0;
35       init(atividade10) := 0;
36       init(atividade11) := 0;
37       init(atividade12) := 0;
38       init(atividade13) := 0;
39       init(atividade14) := 0;
40       init(atividade15) := 0;
41       init(atividade17) := 0;
42       init(atividade18) := 0;
43
44       next(atividade0) := 1;
45       next(atividade3) :=
```

```
46                  case
47                          atividade3=1: 1;
48                          atividade0=1 & ( atividade18=1 ): 0;
49                          atividade0=1: {0,1};
50                          1: 0;
51          esac;
52          next(atividade18) :=
53                  case
54                          atividade18=1: 1;
55                          atividade0=1 & ( atividade3=1 | next(atividade3)=1 ): 0;
56                          atividade0=1 & atividade3=0: 1;
57                          1: 0;
58          esac;
59          next(atividade2) :=
60                  case
61                          (atividade3=1 | atividade18=1): 1;
62                          1: 0;
63          esac;
64          next(atividade4) :=
65                  case
66                          atividade2=1: 1;
67                          1: 0;
68          esac;
69          next(atividade5) :=
70                  case
71                          atividade4=1: 1;
72                          1: 0;
73          esac;
74          next(atividade6) :=
75                  case
76                          atividade5=1: 1;
77                          1: 0;
78          esac;
79          next(atividade7) :=
80                  case
81                          atividade6=1: 1;
82                          1: 0;
83          esac;
84          next(atividade8) :=
85                  case
86                          (atividade7=1 | atividade17=1): 1;
87                          1: 0;
88          esac;
89          next(atividade15) :=
90                  case
91                          atividade8=1: 1;
92                          1: 0;
93          esac;
94          next(atividade9) :=
95                  case
96                          atividade9=1: 1;
97                          atividade15=1 & ( atividade17=1 ): 0;
98                          atividade15=1: {0,1};
99                          1: 0;
```

```
100     esac;
101     next(atividade17) :=
102                 case
103                         atividade17=1: 1;
104                         atividade15=1 & ( atividade9=1 | next(atividade9)=1 ): 0;
105                         atividade15=1 & atividade9=0: 1;
106                         1: 0;
107     esac;
108     next(atividade10) :=
109                 case
110                         atividade9=1: 1;
111                         1: 0;
112     esac;
113     next(atividade12) :=
114                 case
115                         atividade10=1: 1;
116                         1: 0;
117     esac;
118     next(atividade13) :=
119                 case
120                         atividade12=1: 1;
121                         1: 0;
122     esac;
123     next(atividade11) :=
124                 case
125                         atividade13=1: 1;
126                         1: 0;
127     esac;
128     next(atividade14) :=
129                 case
130                         atividade11=1: 1;
131                         1: 0;
132     esac;
133     next(atividade1) :=
134                 case
135                         atividade14=1: 1;
136                         1: 0;
137     esac;
```

Código C.1: Tradução completa do fluxo do sistema de capacitação

Tabela 2: Quadro de resultados dos testes - Fluxo do sistema de capacitação

Validação	Atividade/decisão 1	Atividade/decisão 2	Resultado
A execução de uma atividade ou decisão implica na execução de outra	Cadastra curso no sistema	Distribui vagas de cada turma	Verdadeiro
	Recebe e analisa documentação	Aprova matrículas	Falso
A execução de uma atividade ou decisão implica na não execução de outra	Cadastra curso no sistema	Distribui vagas de cada turma	Falso
	Recebe e analisa documentação	Aprova matrículas	Falso
	Cadastra cliente e suas regras de aprov.	Verifica regras de aprovação	Verdadeiro
A execução de uma atividade ou decisão implica na possibilidade de execução de outra	Cadastra curso no sistema	Distribui vagas de cada turma	Verdadeiro
	Recebe e analisa documentação	Aprova matrículas	Verdadeiro
	Cadastra cliente e suas regras de aprov.	Verifica regras de aprovação	Falso
A execução de uma atividade depende da prévia execução de outra	Cadastra curso no sistema	Distribui vagas de cada turma	Verdadeiro
	Cadastra cliente e suas regras de aprov.	Verifica regras de aprovação	Falso
	Envia documentação para matrícula	Informa setor de RH	Falso
É possível que a atividade não seja executada	Informa setor de RH	-	Verdadeiro
	Cadastra curso no sistema	-	Falso
A atividade nunca é executada	Informa setor de RH	-	Falso
	Cadastra curso no sistema	-	Falso
É sempre possível atingir o final do fluxo	-	-	Falso

ANEXO D – FLUXO DE TRABALHO ADICIONAL - ATENDIMENTO À DENGUE

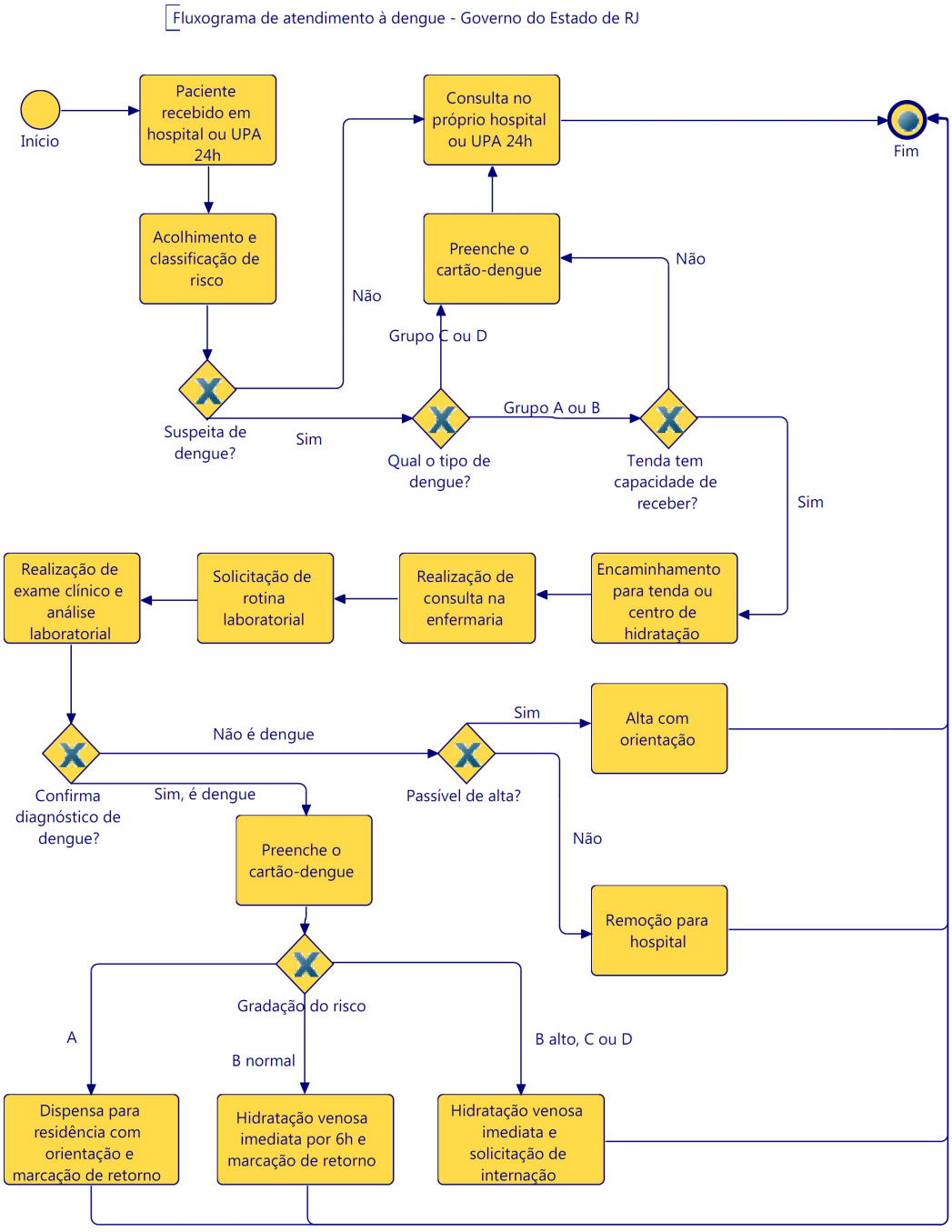

Figura 35: Fluxo "Processo de atendimento à dengue - Governo do Estado do Rio de Janeiro"

```
1  MODULE main
2  VAR
3        atividade0: boolean; -- Nó inicial (_iJlrUZ-_Ed-7W5P-0ye_lQ)
4        atividade1: boolean; -- Fim (nó final - terminate) (_iJlrUp-_Ed-7W5P-0ye_lQ)
5        atividade2: boolean; -- Paciente recebido em hospital ou UPA 24h (_mFaAoJ-_Ed-7W5P-0
             ye_lQ)
6        atividade3: boolean; -- Acolhimento e classificação de risco (_qYltEJ-_Ed-7W5P-0
             ye_lQ)
7       -- atividade4: Suspeita de dengue? (_t0U0IJ-_Ed-7W5P-0ye_lQ)
8        atividade5: boolean; -- Consulta no próprio hospital ou UPA 24h (_z6vYwJ-_Ed-7W5P-0
             ye_lQ)
9        atividade6: boolean; -- Preenche o cartão-dengue (_B8jtsJ_AEd-7W5P-0ye_lQ)
10      -- atividade7: Qual o tipo de dengue? (_LElCIJ_AEd-7W5P-0ye_lQ)
11      -- atividade8: Tenda tem capacidade de receber? (_gYazMJ_AEd-7W5P-0ye_lQ)
12       atividade9: boolean; -- Encaminhamento para tenda ou centro de hidratação (
             _N9_CwJ_BEd-7W5P-0ye_lQ)
13       atividade10: boolean; -- Realização de consulta na enfermaria (_ekilUJ_BEd-7W5P-0
             ye_lQ)
14       atividade11: boolean; -- Solicitação de rotina laboratorial (_iGN8UJ_BEd-7W5P-0ye_lQ
             )
15       atividade12: boolean; -- Realização de exame clínico e análise laboratorial (
             _k2nnYJ_BEd-7W5P-0ye_lQ)
16       atividade13: boolean; -- Preenche o cartão-dengue (_q_UD8J_BEd-7W5P-0ye_lQ)
17      -- atividade14: Confirma diagnóstico de dengue? (_ssjo4J_BEd-7W5P-0ye_lQ)
18       atividade15: boolean; -- Alta com orientação (_xfHY8J_BEd-7W5P-0ye_lQ)
19       atividade16: boolean; -- Remoção para hospital (_208OIJ_BEd-7W5P-0ye_lQ)
20      -- atividade17: Passível de alta? (_5v3uMJ_BEd-7W5P-0ye_lQ)
21       atividade18: boolean; -- Dispensa para residência com orientação e marcação de
             retorno (_q4rbAJ_CEd-7W5P-0ye_lQ)
22       atividade19: boolean; -- Hidratação venosa imediata por 6h e marcação de retorno (
             _5BZpMJ_CEd-7W5P-0ye_lQ)
23       atividade20: boolean; -- Hidratação venosa imediata e solicitação de internação (
             __g41IJ_CEd-7W5P-0ye_lQ)
24      -- atividade21: Gradação do risco (_DDEicJ_DEd-7W5P-0ye_lQ)
25
26 ASSIGN
27       init(atividade0) := 1;
28       init(atividade1) := 0;
29       init(atividade2) := 0;
30       init(atividade3) := 0;
31       init(atividade5) := 0;
32       init(atividade6) := 0;
33       init(atividade9) := 0;
34       init(atividade10) := 0;
35       init(atividade11) := 0;
36       init(atividade12) := 0;
37       init(atividade13) := 0;
38       init(atividade15) := 0;
39       init(atividade16) := 0;
40       init(atividade18) := 0;
41       init(atividade19) := 0;
42       init(atividade20) := 0;
43
```

```
44      next(atividade0) := 1;
45      next(atividade2) := 1;
46      next(atividade3) :=
47                case
48                        atividade2=1: 1;
49                        1: 0;
50      esac;
51      next(atividade5) :=
52                case
53                        atividade5=1: 1;
54                    (atividade3=1): 1;
55                        1: 0;
56      esac;
57      next(atividade1) :=
58                case
59                        (atividade5=1 | atividade15=1 | atividade16=1 | atividade18=1 |
                              atividade19=1 | atividade20=1): 1;
60                        1: 0;
61      esac;
62      next(atividade6) :=
63                case
64              atividade6=1: 1;
65                        atividade3=1 & ( atividade9=1 | atividade5=1 ): 0;
66                        atividade3=1 & ( atividade9=0 & atividade5=0 ): 1;
67                        1: 0;
68      esac;
69      next(atividade9) :=
70                case
71              atividade9=1: 1;
72                        atividade3=1 & ( atividade6=1 | atividade5=1 | atividade6=1 ): 0;
73                        atividade3=1 & ( atividade6=0 & atividade5=0 & atividade6=0 ): 1;
74                        1: 0;
75      esac;
76      next(atividade10) :=
77                case
78                        atividade9=1: 1;
79                        1: 0;
80      esac;
81      next(atividade11) :=
82                case
83                        atividade10=1: 1;
84                        1: 0;
85      esac;
86      next(atividade12) :=
87                case
88                        atividade11=1: 1;
89                        1: 0;
90      esac;
91      next(atividade15) :=
92                case
93              atividade15=1: 1;
94                        atividade12=1 & ( atividade16=1 | atividade13=1 ): 0;
95                        atividade12=1: {0,1};
96                        1: 0;
```

```
97      esac;
98      next(atividade16) :=
99                case
100             atividade16=1: 1;
101                       atividade12=1 & ( atividade15=1 | atividade13=1 ): 0;
102                       atividade12=1 & ( atividade15=0 & atividade13=0 ): 1;
103                       1: 0;
104     esac;
105     next(atividade13) :=
106               case
107             atividade13=1: 1;
108                       atividade12=1 & ( atividade15=1 | atividade16=1 ): 0;
109                       atividade12=1 & ( atividade15=0 & atividade16=0 ): 1;
110                       1: 0;
111     esac;
112     next(atividade18) :=
113               case
114                       atividade18=1: 1;
115                       atividade13=1 & ( atividade19=1 | atividade20=1 ): 0;
116                       atividade13=1: {0,1};
117                       1: 0;
118     esac;
119     next(atividade19) :=
120               case
121                       atividade19=1: 1;
122                       atividade13=1 & ( atividade18=1 | atividade20=1 ): 0;
123                       atividade13=1 & atividade18=0 & atividade20=0: 1;
124                       1: 0;
125     esac;
126     next(atividade20) :=
127               case
128                       atividade20=1: 1;
129                       atividade13=1 & ( atividade18=1 | next(atividade18)=1 |
                              atividade19=1 | next(atividade19)=1 ): 0;
130                       atividade13=1 & atividade18=0 & atividade19=0: 1;
131                       1: 0;
132     esac;
```

Código D.1: Tradução completa do fluxo de atendimento à dengue

Tabela 3: Quadro de resultados dos testes - Fluxo de atendimento à dengue

Validação	Atividade/decisão 1	Atividade/decisão 2	Resultado
A execução de uma atividade ou decisão implica na execução de outra	Paciente recebido em hospital...	Acolhimento e classificação de risco	Positivo
	Paciente recebido em hospital...	Preenche o cartão-dengue (1)	Falso
A execução de uma atividade ou decisão implica na não execução de outra	Paciente recebido em hospital...	Acolhimento e classificação de risco	Falso
	Paciente recebido em hospital...	Preenche o cartão-dengue (1)	Falso
	Alta com orientação	Preenche o cartão-dengue (2)	Positivo
A execução de uma atividade ou decisão implica na possibilidade de execução de outra	Paciente recebido em hospital...	Preenche o cartão-dengue (1)	Positivo
	Paciente recebido em hospital...	Encaminhamento para tenda...	Positivo
	Consulta no próprio hospital...	Encaminhamento para tenda...	Falso
A execução de uma atividade depende da prévia execução de outra	Paciente recebido em hospital...	Consulta no próprio hospital...	Positivo
	Preenche o cartão-dengue (1)	Consulta no próprio hospital...	Falso
	Alta com orientação	Preenche o cartão-dengue (2)	Falso
É possível que a atividade não seja executada	Preenche o cartão-dengue (1)	-	Positivo
	Paciente recebido em hospital...	-	Falso
A atividade nunca é executada	Preenche o cartão-dengue (1)	-	Falso
	Paciente recebido em hospital...	-	Falso
É sempre possível atingir o final do fluxo	-	-	Positivo