UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

LINA RAYLLY MOREIRA GOMES

CONSTRUÇÃO AUTOMÁTICA DE ONTOLOGIA EM PROLOG A PARTIR DE MODELOS DE PROCESSOS EM BPMN

QUIXADÁ

2016

LINA RAYLLY MOREIRA GOMES

CONSTRUÇÃO AUTOMÁTICA DE ONTOLOGIA EM PROLOG A PARTIR DE MODELOS DE PROCESSOS EM BPMN

Trabalho de Conclusão de Curso submetido à Coordenação do Curso de Graduação em Sistemas de Informação da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: Computação.

Orientador: Prof. Dr. Davi Romero de Vasconcelos.
Co-orientador: Prof. Msc. Camilo Camilo Almendra.

QUIXADÁ

2016

Dados Internacionais de Catalogação na Publicação

Universidade Federal do Ceará

Biblioteca do Campus de Quixadá

G615c Gomes, Lina Raylly Moreira

Construção automática de ontologia em Prolog a partir de modelos de processos em BPMN / Lina Raylly Moreira Gomes. – 2016.

87 f. : il. color., enc. ; 30 cm.

Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Bacharelado em Sistemas de Informação, Quixadá, 2016.

Orientação: Prof. Dr. Davi Romero de Vasconcelos

Área de concentração: Computação

1. Linguagem de programação lógica 2. XML (Linguagem de marcação de documento) 3. Gerenciamento de processos I. Título.

CDD 005.1

LINA RAYLLY MOREIRA GOMES

CONSTRUÇÃO AUTOMÁTICA DE ONTOLOGIA EM PROLOG A PARTIR DE MODELOS DE PROCESSOS EM BPMN

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Bacharelado em Sistemas de Informação da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: computação

Aprovado em: _____ / _____ / _____.

BANCA EXAMINADORA

_____

Prof. Dr. Davi Romero de Vasconcelos
(Orientador)

Universidade Federal do Ceará-UFC

_____

Prof. MSc. Camilo Camilo Almendra

(Co- orientador)

Universidade Federal do Ceará-UFC

_____

Prof. MSc. Lucas Ismaily Bezerra Freitas

Universidade Federal do Ceará-UFC

À Deus.

Única razão da minha existência.

AGRADECIMENTOS

À Deus, impossível descrever com palavras toda a sua grandeza, amor e poder. Obrigada Senhor pelo dom da vida, pela sua permissão para que eu chegasse até aqui, por tudo que o Sr. tem me proporcionado em minha vida e por todos os teus cuidados durante todos esses anos.

Aos meus pais Antônio Moacir (*in memorian*) e Eufêmia Moreira, pelo amor, ensinamentos, e todos os cuidados para mim proporcionados durante toda minha vida. Mãe muito obrigada, essa graduação teria sido bem mais difícil sem a Sra. ao meu lado.

A minha querida irmã, Ana Raquel, ela é sim o melhor presente que os meus pais já me deram ☺. Pela cumplicidade, pelo incentivo e pelos momentos de descontração juntas, por todo amor para mim demostrado em todos esses anos de convivência.

A todos os meus familiares em especial a minha querida tia Vilani todo seu amor, carinho e cuidado, tias sabem amar como se fossem mães. A minha tia Mariana por tudo que fez por mim, e a todas as minhas demais tias: Ana Maria, Sônia Viana, Fátima, Euridéia e Francisca e Terezinha esta última a considero como minha tia. A minha prima Meiriane Moreira, ao qual estou torcendo para que se forme muito em breve. E a todos os demais primos e primas, tios, tia e minha querida vozinha.

Ao meu professor, tutor, e orientador Prof. Davi Romero de Vasconcelos, pela paciência, pela tutoria no PET durante cinco anos, por todas as lições que eu vou levar por toda minha vida, pela excelente orientação e constante ajuda no desenvolvimento deste trabalho, e por todo tempo dedicado para esta orientação. Muito obrigada prof. Davi.

Ao meu co-orientador Camilo Camilo Almendra pelos conhecimentos compartilhados que foram indispensáveis para aperfeiçoar este trabalho. Ao convidado professor Lucas Ismaily Bezerra Freitas que completou a banca julgadora desta monografia, disponibilizando seu tempo e oferecendo valiosas sugestões para melhoria do trabalho.

A professora Tânia Pinheiro por todas os conselhos, dicas, e ensino durante a disciplina de Projeto de Pesquisa que foram indispensáveis para a criação deste trabalho.

A todos os técnicos administrativos do *Campus* Quixadá, em especial, a secretária do campus, Evalda Rodrigues, que se disponibilizou para me ensinar todo o trâmite do processo de afastamento de professores e a revisar meu diagrama em BPMN diversas vezes. Obrigada Evalda sua ajuda foi de extrema importância para desenvolvimento deste trabalho. Não

poderia esquecer das bibliotecárias da UFC que me ajudaram muito com as referências da ABNT. Obrigada meninas.

A todos os professores do ensino médio.

A todo o corpo docente que faz ou fez parte da UFC Quixadá que contribuíram para minha formação no decorrer de todos esses anos. Em especial ao professor Wladimir Araújo que me proporcionou momentos de alegria durante os estudos, sempre me contagiando com seu entusiasmo, tornavam o aprender sempre mais agradável; agradeço a todas as horas que prontamente se dispôs para sanar minhas dúvidas, responder e-mails, corrigir e auxiliar em trabalhos. Obrigada Wladimir, você fez enorme diferença na minha vida durante essa fase universitária.

Ao mestre Ricardo Reis em todas as minhas conversas com ele sempre teve algo a me ensinar, e ao ilustre Professor Aragão por todos os valiosos ensinamentos dados com seu exemplo de vida.

A todos os colegas da minha turma de graduação e a todos os que conheci no grupo PET, pelos momentos de estudo juntos e de descontração. Em especial minha grande amiga, Leonara Braz, pela sua amizade durante todo esse tempo, pelas suas palavras sinceras e pela ajuda que me proporcionou em momentos difíceis da vida. A minha amiga Thalita Maria, ao meu amigo Bruno Leandro, Pedro Henrique, José Diógenes e Alexandra Valesca, e por fim a minha amiga de adolescência Patrícia Maia.

Aos demais estudantes que fizeram laços comigo, mas que não são da minha turma incluindo amigos que fiz no curso de inglês da UECE.

A todos os meus irmãos em Cristo que conheci na Igreja de Cristo em Quixadá na Igreja Presbiteriana e na Igreja Internacional da graça de Deus. Em especial a minha irmã em Cristo, Francisca das Dores e ao seu esposo Ednei Lourenço. Obrigada Dorinha pelas palavras de força, por sempre acreditar muito em mim. E eu não poderia esquecer da Vandilma Diniz, que me ensinou a dar meus primeiros passos na fé, e teve um papel muito importante na minha vida espiritual. Obrigada Vandilma.

“O meu povo foi destruído porque lhe faltou conhecimento”. (Oséias 4:6a).

RESUMO

A *Business Process Management Notation* (BPMN) permite a modelagem de processos de negócios. Essa modelagem é feita para alguma finalidade especifica, padronizar, simular, documentar processos de negócios, identificar problemas, implementar soluções entre outros. Contudo analistas por descuidos ou por não conhecerem bem a notação podem desenvolver inconsistências. Essas, por sua vez são um problema, pois dificultam a leitura da modelagem, travam o processo e podem gerar erros sintáticos e semânticos. Para solucionar este problema propomos uma solução usando programação em lógica e JAVA; utilizaremos a programação lógica para fazer consultas que são respondidas através de inferências e deduções, e JAVA para ler e transformar o *eXtensible Markup Language* (XML) da modelagem ao seu respectivo código em Prolog. Assim, desenvolvemos um aplicativo em JAVA Desktop que usa a API DOM W3C, e recebe como entrada o código em XML de uma modelagem BPMN e retorna o código em Prolog referente a esse código XML. Por fim, apresentamos um escopo de consultas que podem identificar algumas inconsistências em modelos de processos. Com essas inconsistências identificadas o processo de melhoria torna-se mais fácil e analistas e desenvolvedores podem entender melhor os modelos de processos.

Palavras-chave: BPMN. Prolog. Tradutor.

ABSTRACT

The *Business Process Management Notation* (BPMN) enables the modeling of business processes. This modeling is done for some specific purpose, standardize, simulate, document business processes, identify problems, implement solutions among others. However, analysts by carelessness or not well known notation can develop inconsistencies. These, in turn are a problem because difficult to read modeling, lock the process, and can generate syntactic and semantic errors. To solve this problem we propose a solution using logic programming and JAVA; we use logic programming to queries that to answer through inferences and deductions, and JAVA to read and transform the eXtensible Markup Language (XML) from the model to its corresponding code in Prolog. Thus, we developed a Java Desktop in application that uses the API DOM W3C, and receives as input the XML code in a BPMN modeling and returns the code in Prolog regarding this XML code. Finally, we present a query scope that can identify some inconsistencies in process models. With these inconsistencies identified, the improvement process becomes easier and analysts and developers can better understand the process models.

Keywords: BPMN. Prolog. Translator.

LISTA DE ILUSTRAÇÕES

Figura 1: fluxograma similar a modelagem visual do BPMN. 14
Figura 2: Metodologia para o desenvolvimento de ontologia. 16
Figura 3: Fases de um Processo. 19
Figura 4: Organização do gerenciamento de processos de negócio. 20
Figura 5: Linhas de conexão. 32
Figura 6: Artefatos. 33
Figura 7: Piscinas do BPMN. 34
Figura 8: Lane do BPMN. 35
Figura 9: Operadores Lógicos. 41
Figura 10: Processo em BPMN incompleto. 45
Figura 11: Erros em modelagem, processo termina sem início. 45
Figura 12: Símbolos soltos no diagrama. 46
Figura 13: símbolos mal usados. 46
Figura 14: ciclos em modelagens de processos. 47
Figura 15: Desenho ilustrativo das fases deste trabalho. 51
Figura 16: Desenho ilustrativo do processo de tradução. 51
Figura 17: Fluxo simples em BPMN. 52
Figura 18: delineações do espaço. 59
Figura 19: Fluxo com gateway em BPMN. 59
Figura 20: Fluxo com dois gateways em BPMN. 60
Figura 21: Fluxo com um gateway do tipo divergente e outro do tipo convergente em BPMN. 61
Figura 22: Fluxo apresentando processo completo mais simples em BPMN. 62
Figura 23: fluxo quebrado. 68
Figura 24: Saída de gateways sem nomes de rótulos. 72
Figura 25: Processo de Afastamento de Professores em BPMN. 85

LISTA DE QUADROS

Quadro 1: As categorias de eventos. 23
Quadro 2: Tipos de eventos de início. 24
Quadro 3: Tipos de eventos intermediários. 27
Quadro 4: Tipos de eventos intermediários. 28
Quadro 5: Tipos de Gateways. 30
Quadro 6: Tipos de Atividades. 32

LISTA DE SIGLAS E ABREVIATURAS

BPM – Business Process Management

BPMN – Business Process Management Notation

BPD – Business Process Diagram

BPMI - Business Process Managent Initiative

XML – eXtensible Markup Language

HTML – HyperText Markup Language

BPMS - Business Process Management System

FGCS - Fifth Generation Computing Systems

SUMÁRIO

1 INTRODUÇÃO 13

2 TRABALHOS RELACIONADOS 16

3 FUNDAMENTAÇÃO TEÓRICA 18

3.1 Processo 18
3.2 Gestão de processos de negócio. 19
3.3 Elementos BPMN 22
3.3.1 Objetos de fluxo 22
3.3.2 Objetos de Conexão 32
3.3.3 Artefatos 33
Objeto de Dados 33
Agrupamentos 34
3.3.4 Piscina (Pool) 34
3.3.5 Raias (Lanes) 34
3.4 Representação do Conhecimento 35
3.4.1 Programação Lógica 35
3.5 Problemas típicos em modelagem de processos 43

4 PROCEDIMENTOS METODOLÓGICOS 49

4.1 Escolha da ferramenta a ser utilizada para modelar processos 49
4.2 Modelagem do processo de afastamento de Professores em BPMN 49
4.2.1 Tradução do Processo de Afastamento de Professores para Programação Lógica 49
4.2.2 Consultas em Programação Lógica 49
4.3 Mapeamento geral que traduz qualquer processo em BPMN para Prolog 49

5 MAPEAMENTO GERAL (Diagramas BPMN para Programação em Lógica) 50

5.1 Tipos de Consultas 64
5.1.1 Consultas para descobrir se fluxos terminam ou seja se eles têm no mínimo um evento de fim 65
5.1.2 Consultas para descobrir se o fluxo inicia com um StartEvent 66
5.1.3 Consultas para descobrir se um fluxo está desconectado pela origem ou destino. 67
5.1.4 Consultas para descobrir se existe fluxo perdido sem origem nem destino. 69
5.1.5 Consultas para descobrir se existe Task e subProcessos perdidos sem entrada e saída 70
5.1.6 Consultas para descobrir se existe gateway sem rótulos. 71
5.1.7 Consultas para descobrir se existem ciclos no programa 72

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS 74

REFERÊNCIAS 76

APÊNDICE A – Caso de estudo: Processo de afastamento de Professores 82
Descrição do Processo em Linguagem Natural 82
Representação do Processo em Programação Lógica. 86

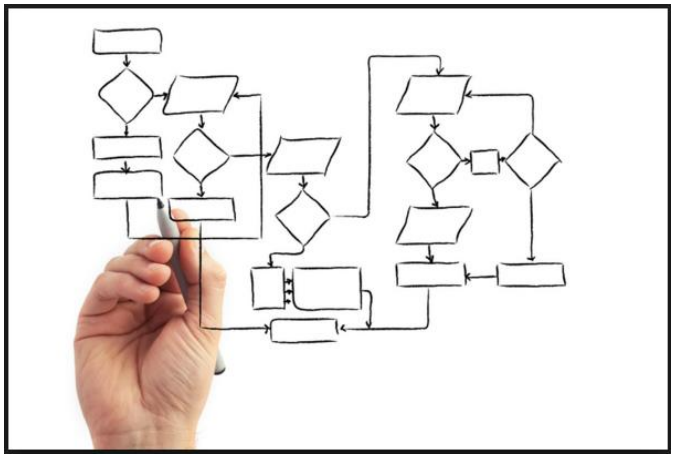
1 INTRODUÇÃO

A gestão de processos de negócio ou em inglês *Business Process Management* (BPM) é “um conjunto de métodos, técnicas e ferramentas computacionais desenvolvidas para amparar as diferentes fases do ciclo de vida de um processo de negócio, ou seja, o projeto, a configuração, a implantação/execução e a análise”. (VAN DER AALST, 2003).

Segundo Ferreira (2013), o BPM surgiu devido uma reorganização horizontal das empresas em torno dos processos, o qual fez com que o foco que era no fluxo de trabalho de áreas funcionais, fosse substituído pelo foco nos processos de negócio da organização, com isso o foco que antigamente não era processos passou a ser.

O BPM tem sido utilizado pelas empresas para agregar valor à organização (VALE, 2013) e gerar valor para os seus clientes (FERREIRA, 2013). Além desses, existem outros benefícios que podem ser alcançados com a gestão de processos de negócio, dentre eles temos a simulação, a documentação, a execução, o controle e por fim a compreensão dos processos por parte do leitor.

O BPM possui uma notação própria para fazer modelagem de processos, o *Business Process Management Notation* (BPMN), que é um padrão que foi desenvolvido visando fornecer as empresas uma notação gráfica mais facilmente compreendida, que apoia o gerenciamento e monitoramento dos processos (BPMI/OMG, 2006). A Figura abaixo ilustra um fluxograma similar a um diagrama feito usando BPMN.



Fonte: IEL(2016)
Figura 1: fluxograma similar a modelagem visual do BPMN.

Aos diagramas que usam o BPMN, chamamos de *Business Process Diagram* **(**BPD). Eles são um tipo de modelagem visual de um processo em BPM e servem para várias finalidades: padronizar processos, melhorar a qualidade do processo executado na empresa, facilitar a identificação de problemas e soluções em processos (VALE, 2013), tornar conhecido um processo a determinado analista ou desenvolvedor, entre outros.

A modelagem visual "busca definir o processo de uma forma intuitiva e não ambígua" (FRANCO, 2013), para que todas as partes da organização compreendam os processos da mesma forma. Essa compreensão idêntica dos processos entre os envolvidos é necessária; porque quando isso acontece, cada parte da organização age para atingir seus objetivos com a mesma visão do todo, e cada parte atingindo seus objetivos a organização como um todo atinge os seus, gerando assim maior crescimento (VALE, 2013).

O BPD facilita o entendimento dos processos de negócio para analistas, desenvolvedores, e todos os interessados no processo de forma geral. Esse entendimento de forma fácil ocorre por causa da própria notação do BPM, que devido ser gráfica, é facilmente compreendida e assimilada. No entanto, essa compreensão e assimilação pode ser perdida quando diagramas forem mal elaborados. Dúvidas podem surgir no momento da leitura visual, quando isso ocorre um dos sentidos da modelagem é perdida, pois o esperado com diagramas é que eles sejam autoexplicativos.

Uma solução para resolver esse tipo de problema é fazer uma análise de diagramas, descobrir erros, inconsistências, falhas, ambiguidades e refazer os desenhos

colocando informações que sanam quaisquer dúvidas no momento da leitura visual. Após os desenhos refeitos é necessário que a análise seja refeita, e esse ciclo de refazer desenhos e refazer análise deve continuar até o momento em que não existir nenhuma inconsistência claramente identificada.

Para identificamos essas inconsistências usaremos uma “técnica de organização da informação" (AMBROSIO, 2007) ou técnica de organização do ser (GUIMARÃES, 2002) denominada ontologia. A literatura mostra que existem vários tipos de ontologias e vários conceitos para esse termo dentro da ciência da computação, provindo dos vários pontos de vista diferentes. Sabe-se que o termo vem da filosofia que lida com a natureza e a organização do ser. Neste trabalho, usaremos programação em lógica, Prolog, como linguagem para representar ontologias.

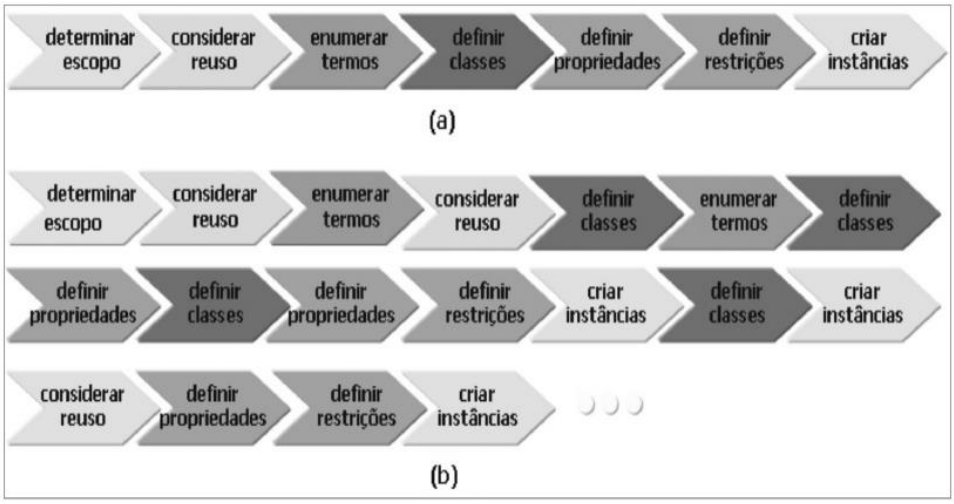
Este trabalho se propõe a identificar inconsistências em modelagens BPMN, mapeando automaticamente processos em BPMN para programação em lógica e fazendo consultas em Prolog. Para fazer esse mapeamento automático usamos como tradutor um aplicativo que desenvolvemos em Java que usa a API DOM do tipo W3C. Este recebe um código em XML referente a uma modelagem em BPMN e o transforma em um código usando Programação Lógica.

O restante deste trabalho está dividido em 5 seções. Na seção 2, apresentamos alguns trabalhos relacionados. Na seção 3, apresentamos a fundamentação teórica usada neste trabalho. Na seção 4, apresentamos os procedimentos metodológicos utilizados para realização deste trabalho. Na seção 5, mostramos como elaboramos um construtor automático que recebe um código *eXtensible Markup Language* (XML) de uma modelagem e o transforma em um código em Prolog. E por fim na seção 6 tecemos conclusões e trabalhos futuros.

2 TRABALHOS RELACIONADOS

Na literatura é possível encontrar alguns trabalhos que fazem uso de ontologias e outros trabalhos que fazem uso BPMN. Entretanto, encontrar os dois temas em conjunto é mais difícil, ainda assim encontramos dois com essas características o de Ligeza et al. (2012) e o de Androcec (2010), que apresentamos mais à frente.

Rautemberg et al. (2008) propõe uma metodologia para o desenvolvimento de ontologias, mostrando uma série de passos utilizados na elaboração de uma ontologia, independentemente se esta vem ou não com uma representação em BPMN, esses passos, se seguidos produzem uma ontologia. A Figura 2 apresenta os passos necessários para se fazer uma ontologia.



Fonte: Rautemberg et. al.(2008).
Figura 2: Metodologia para o desenvolvimento de ontologia.

Com essa metodologia podemos fazer uma ontologia (programa em Prolog), definindo o escopo, no escopo precisamos saber o que pretendemos atingir com a representação no Prolog, que nesse caso é: fazer uma verificação formal de diagramas BPMN. Após definir escopo devemos observar na literatura os trabalhos relacionados para decidir se vamos reusar alguma ideia. Em seguida devemos enumerar termos, classificar ou usar termos existentes do BPMN para fazer a representação em Prolog. Logo após devemos definir classes e propriedades utilizadas no programa assim como os relacionamentos entre objetos. E por fim deve-se definir as restrições e criar instâncias. Aproveitamos neste trabalho apenas a metodologia apresentada na Figura 2(a).

Devido a carência existente de verificação formal em diagramas BPMN, Ligeza et al. (2012) investigam como construir um modelo lógico de BPMN usando lógica de programação e regras. Após criado um processo em BPMN é feito um modelo em Prolog para análise lógica do diagrama.

Em Androcec (2010) é feito uma simulação utilizando programação lógica de um modelo BPMN. No artigo, Androcec tem o objetivo de mostrar que os mesmos resultados obtidos usando IBM *WebSphere Business Modeler* podem ser obtidos utilizando programação lógica (Prolog).

Neste trabalho exploramos a característica declarativa do Prolog e construímos de forma automática o mapeamento geral BPMN para Prolog, buscando tirar a carga manual do usuário que deseja realizar a tradução de modelos de Processos BPMN para Programação Lógica.

3 FUNDAMENTAÇÃO TEÓRICA

Esta seção aborda algumas informações relevantes assim como conceitos de: Processo e seu gerenciamento, Elementos BPMN, Representação do conhecimento e Programação em lógica (Prolog). Escolhemos para cada tópico a parte essencial para apresentarmos neste trabalho, por isso, para mais informações sobre o assunto indicamos Pain et. al (2009), Model (2011), Instituto (2013) e Vale (2013).

3.1 Processo

De acordo com o dicionário Aurélio da língua portuguesa existem várias definições para a expressão *processo,* dentre as quais, a que mais se aproxima de ser a definição dentro da ciência da computação é a seguinte: "um conjunto de manipulações para obter um resultado". Na literatura encontramos outros conceitos de alguns autores da área. Na sequência temos:

1- Hammer e Champy (1994) que define processo como "um grupo de atividades realizadas numa sequência lógica com o objetivo de produzir um bem ou um serviço que tem valor para um grupo específico de clientes".
2- E Vale et. al (2013) que define processo como: "um encadeamento de atividades executadas dentro de uma companhia ou organização, que transformam entradas em saídas".

Os processos dentro do contexto computacional geralmente apresentam uma estrutura comum, tendo uma entrada, um processamento e uma saída. A entrada geralmente são os dados ou as informações necessárias para que o processamento seja computado. O processamento são os passos feitos com esses dados para que tenha uma saída para o usuário. A saída é uma informação, um documento ou um material obtido através dos dados processadas. Veja os três passos na Figura 3.

Fonte: Ramos(2003).
Figura 3: Fases de um Processo.

Ao tomarmos processos como exemplos podemos identificar as três fases da Figura 3, pois o padrão é que processos sejam eles administrativos ou operacionais possuam as três fases, mesmo que implícitas.

3.2 Gestão de processos de negócio.

Alguns desconhecem e acreditam que gerenciamento de processos de negócio é desperdício de tempo, acreditam que pode-se saltar esta etapa, o que não é uma realidade (VALE, 2013). De acordo com Vale (2013), antes de elaborar qualquer modelagem é necessário determinar a finalidade de se modelar determinado processo. Algumas finalidades apontadas por Vale são:

1 - Entender o negócio através do comportamento dos processos, permitindo a identificação de seus requisitos retrabalhos, gargalos e ineficiências.

2 – Analisar oportunidades de melhoria e monitoramento em processos de negócio.

3 – Melhorar a qualidade e produtividade.

4 - Facilitar a identificação e solução de Problemas em Processos de negócios. Entre outros.

A área pode ser organizada e/ou dividida em cinco fases principais que são: modelagem, análise, desenho, gerenciamento de desempenho e transformação de processos de negócio. Veja a Figura 4 abaixo:

Fonte: PIRES (200-?).
Figura 4: Organização do gerenciamento de processos de negócio.

Gerenciar processos de negócios é importante porque esses precisam ser projetados, documentados e desenhados visando seu aperfeiçoamento. Além disso, os processos precisam ser compreendidos por todo o público que tem interesse no assunto.

Segundo Pain et. al (2009), as pessoas estudam e entendem processos de negócio para diminuir a perda de tempo na identificação de um problema de desempenho nos processos. Uma das formas encontradas para reduzir esse tempo foi a análise de processos e algumas ações de modelagem. Contudo, para que o tempo seja realmente reduzido, essas ações devem estar bem estruturadas de forma que seja permitido os processos serem facilmente diagnosticados e as soluções sejam facilmente identificadas, permitindo assim uma implantação no menor intervalo de tempo e custos.

No dia-a-dia as organizações passam a internalizar a gestão de processos melhorando o trabalho que vem sendo realizado e projetado na empresa. Tudo isso vem contribuindo para a incorporação de práticas que melhoram o desempenho desse trabalho ao longo do tempo. Quando entendemos a importância de gestão de processos para uma organização, percebemos o quanto é essencial trabalharmos esses processos para a melhoria de uma empresa ou organização.

Giannini (2013) mostra 50 motivos pelos quais devemos usar BPM em uma organização, extraímos os 10 que acreditamos ser mais relevantes. Os motivadores estão escritos e enumerados a seguir:

1. Reorganizações: transição de papéis e de responsabilidades. Uma vez que os papéis e responsabilidades estejam representados podem ser compreendidos por outros funcionários da empresa.
2. Crescimento elevado: dificuldade em lidar com um alto nível de crescimento, ou de fazer um planejamento proativo para uma elevada taxa de crescimento.
3. A necessidade de agilidade do negócio, permitindo que a organização responda às oportunidades conforme surjam.
4. Necessidade de prover aos gestores um maior controle sobre seus processos.
5. Alta rotatividade dos funcionários, talvez devido à natureza do trabalho ou do grau de pressão e expectativas sobre pessoas sem o suporte adequado. Com os processos mapeados esses problemas de treinamento dos novos funcionários podem ser amenizados.
6. Baixa satisfação com os serviços que podem estar sendo causando pela alta rotatividade do pessoal e/ou pela incapacidade dos funcionários de responder de forma adequada dentro do tempo esperado.
7. Prazos de entrega inaceitáveis para o mercado (falta de agilidade do negócio, uma forma de agilizar é mapeando processos em BPM, isso pode fazer uma grande diferença).
8. Papéis e responsabilidades mal definidos do ponto de vista do processo.
9. Falta de padronização dos processos.
10. Falta de comunicação e entendimento do processo de ponta-a-ponta pelas partes que participam do processo.

Arantes (2014) afirma que:

> "O BPMN permite descrever a lógica dos passos de um processo. Com a modelagem é possível ter uma notação gráfica que expressa para o usuário de forma clara determinado processo de negócio, onde, mesmos processos complexos se tornam de fácil compreensão/visualização para os donos do negócio facilitando tanto em análise de melhoria quanto de automatização deste processo".

Para fazermos uma melhoria de um determinado processo é importante elaboramos uma modelagem, pois a partir dessa modelagem são descobertas inconsistências e essas quando descobertas podem ser tratadas. Esses ajustes e correções trazem benefícios para a empresa e para todos os envolvidos.

3.3 Elementos BPMN

A notação BPMN utiliza sinais para modelar processos de negócios (HEREDIA, 2012), facilitando a comunicação das áreas gerenciais de uma organização. Foi desenvolvida pela *Business Process Managent Initiative* (BPMI), e atualmente é mantida pelo *Object Management Group* (PIZZA, 2012).

A notação BPMN permite a representação gráfica e a documentação de processos em forma de fluxogramas. Os elementos BPMN são divididos em cinco categorias básicas, que são: objetos de fluxo, objetos de conexão, dados, agrupamentos e artefatos. Nas próximas subseções apresentamos um pouco sobre cada um deles.

3.3.1 Objetos de fluxo

São elementos responsáveis por definir o comportamento de um processo. Podem ser divididas em outras três categorias: evento, atividade e gateway.

Eventos:

São objetos em formato de círculo que servem para identificar ou informar algo que ocorre no processo. Por exemplo, para marcar ou informar o início de um processo usamos o evento do tipo início. O início de um processo pode iniciar de várias formas diferentes, por isso existe no BPMN diversos tipos de eventos de início. Existem três categorias de eventos e cada uma se divide em diversos tipos. As três categorias são: eventos de início, eventos intermediários e eventos de fim. Mais adiante explicaremos cada categoria e cada tipo de forma detalhada. Observe o Quadro 1, com as três primeiras divisões.

	**Eventos de Início**: indicam o início de um processo. Ao ler um mapa de processo, comece por ele!
	**Eventos Intermediários**: ocorrem durante o transcurso de um processo, ou seja, entre o início e o fim.
	**Eventos de Fim**: indicam onde um processo é finalizado.

Fonte: INSTITUTO (2013).
Quadro 1: As categorias de eventos.

Eventos de Início

Os eventos de início ao qual são apresentados no Quadro 2 tem o objetivo de marcar o início de um processo, eles podem ser facilmente identificados pela sua cor que é verde. Observe o Quadro 2, que mostra os elementos e a especificação ao lado de cada um deles.

TIPOS DE EVENTOS DE INÍCIO	DESCRIÇÃO
	O Evento genérico serve para representar um início de um processo que não apresenta nenhum comportamento dos posteriores.
	O evento de início de *timer* significa que existe um tempo estimulado para o início do processo.
	O evento de início de *conditional* indica que o processo só iniciará quando uma condição for satisfeita.

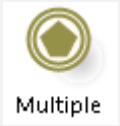
	O evento de início de *message* é usado para indicar que é necessário o recebimento de uma mensagem para iniciar o processo.
	O evento de *multiple* indica que existem muitas formas de iniciar o processo e que ao se cumprir uma das mesmas o processo iniciará.
	O evento de início *parallel multiple* indica que para iniciar o processo é necessário esperar que ocorram múltiplos eventos, e todos esses múltiplos eventos precisam ocorrer para o processo iniciar.
	O evento de início de *signal* indica que um processo começa quando um sinal de outro processo é capturado.

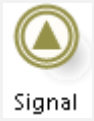
Fonte: BPMN (200-?) com adaptações.
Quadro 2: Tipos de eventos de início.

Para este trabalho utilizamos apenas o evento de início genérico. No processo de afastamento de professores que está nos Apêndices e demais exemplos utilizados no trabalho adotaremos somente o evento genérico.

Eventos Intermediários

Os eventos intermediários ao qual são apresentados no Quadro 3 tem o objetivo de marcar algo que ocorre durante o processo, depois do início e antes do fim. O Quadro 3 mostra os elementos e a especificação ao lado de cada um deles.

Tipos De Eventos Intermediários	Descrição
	O evento intermediário genérico indica que algo pode ocorrer durante o processo. Ele só pode ser usado dentro da sequência do fluxo.
	O evento intermediário de *timer* indica uma espera dentro do processo. Este tipo de evento pode utilizar-se dentro do fluxo de sequência indicando uma espera entre as atividades ou ligado aos limites de uma atividade para indicar um fluxo de exceção.
	O evento intermediário *conditional* é utilizado quando é necessário esperar que uma condição de negócio se cumpra.
	O evento intermediário de *message* indica que uma mensagem pode ser enviada ou recebida. Se o evento de mensagem é de recepção indica que o processo não continua basta que a mensagem seja recebida.
	O evento intermediário de *multiple* indica que é esperado a ocorrência de uma dentre os múltiplos eventos.

	O evento intermediário *parallel multiple* indica que são esperados vários eventos para continuar o processo e todos devem ocorrer para o fluxo prosseguir.
	O evento intermediário de *signal* é utilizado para enviar e receber sinais. Só pode ser utilizado dentro do fluxo de sequência para enviar ou receber sinais ou ligado a limites de uma atividade indicando fluxo de exceção que se ativara quando o sinal for capturado.
	O evento intermediário de *link* é utilizado para permitir conectar duas seções do processo.
	O evento intermediário de *compensate* permite manejar compensações. Quando se utiliza dentro do fluxo de sequência de um processo indica que se lançará uma compensação. Quando se utiliza ligado aos limites de uma atividade (sempre de captura) indica que essa atividade se compensará quando o evento se ative.

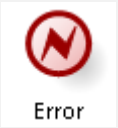
	O evento intermediário de *escalation* indica que o processo está passando por exceção de negócio, lançando o evento para ser capturado pelo processo que está em um nível acima.

Fonte: INSTITUTO (2013), BPMN (200-?) com adaptações.
Quadro 3: Tipos de eventos intermediários.

Eventos de Fim

Os eventos de fim ao qual são apresentados logo em seguida tem o objetivo de marcar o fim de um processo. Seguindo o padrão o Quadro 4 apresenta os elementos na primeira coluna e a especificação ao lado de cada um deles na segunda coluna.

Tipos de Eventos de Fim	**Descrição**
	O evento de fim genérico indica que aquele fluxo chegou ao fim.
	O evento de fim *terminate* indica que o processo é terminado, é dizer que algum caminho do fluxo chega ao fim e o processo termina completamente sem se importar que existam mais fluxos do processo pendente.
	O evento de fim de *message* permite enviar uma mensagem para finalizar determinado fluxo.

	O evento de fim *multiple* indica que vários resultados podem ocorrer ao final de um fluxo.
	O evento de fim *signal* permite enviar um sinal para finalizar o fluxo.
	O evento de fim *compensate* indica que é necessária uma compensação ao finalizar o fluxo.
	O evento de fim de *error* é usado para enviar uma exceção de erro para finalizar o fluxo.
	O evento de fim de *escalation* é usado para finalizar o fluxo com uma exceção de negócio lançando esse evento para ser capturado pelo processo que está em um nível acima.
	O evento de fim de *cancel* permite enviar uma exceção de cancelamento ao finalizar o fluxo. Obs.: Só é utilizado em sub processos.

Fonte: INSTITUTO (2013), BPMN (200-?) com adaptações.

Quadro 4: Tipos de eventos intermediários.

Gateways

Os gateways são elementos em formato de losango e representam uma divergência ou uma convergência. Esses elementos BPMN tem a finalidade de fazer divergir um fluxo em dois ou mais fluxos, ou convergir dois ou mais fluxos em um fluxo. É muito utilizado quando é necessário sair de um fluxo normal que pode divergir em sim ou não, aprovado ou reprovado, A ou B, nesses casos é possível usá-lo para representar esses diferentes fluxos. O Quadro 5, apresenta seus tipos com suas descrições ao lado.

Tipos de Gateways	Descrição
	**Divergência:** Ocorre quando um ponto do fluxo baseado nos dados do processo escolhe um só caminho dos vários disponíveis. **Convergência:** como ponto de convergência é usado para convergir caminho exclusivo.
	**Divergência:** Se utiliza quando várias atividades podem se realizar concorrentemente ou em paralelo. **Convergência:** permite sincronizar vários caminhos paralelos em um só.
	**Divergência:** Se utiliza quando um ponto se ativa em um ou mais caminhos de vários caminhos disponíveis baseados nos dados do processo. **Convergência:** Se utiliza para sincronizar caminhos ativados previamente por uma comporta inclusiva usada como ponto de divergência.

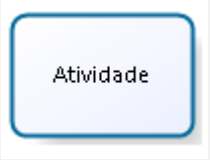
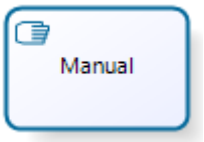
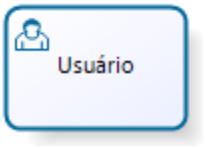
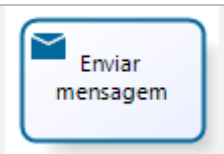
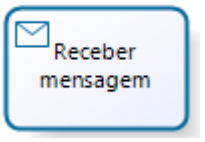
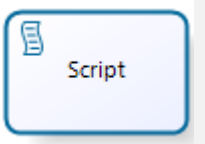
	**Apenas de divergência:** usado para iniciar um processo baseado na ocorrência de múltiplos eventos.
	**Apenas de divergência:** inicia um processo baseado na ocorrência de múltiplos eventos.
	**Apenas de divergência:** é usado quando a condição de escolha do caminho a ser seguido está vinculada a ocorrência exclusiva de um dos eventos.
	**Divergência:** É utilizada para controlar pontos de decisão complexos. **Convergência:** Permite continuar o seguinte ponto do processo quando uma condição de negócio se cumpre.

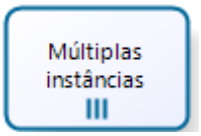
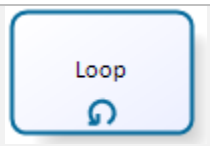
Fonte: BPMN (200-?) e IPROCESS (2014).
Quadro 5: Tipos de Gateways.

Atividades

As atividades são elementos que identificam uma ação que pode ser executada por alguém ou por um sistema. Geralmente as atividades são identificadas por um verbo, ou seja, quando utilizamos um verbo para expressar algo feito conseguimos identificar a presença de uma atividade. Por exemplo, na sentença, *um professor solicita afastamento*, podemos facilmente identificar que existe uma atividade de solicitar feita

por um professor. Uma atividade é um passo dentro de um processo (INSTITUTO, 2013).

Tipos de Atividades	Descrição
	Esse é o tipo genérico de atividade, normalmente utilizado nos estágios iniciais do processo.
	É uma atividade não-automática, realizada por uma pessoa, sem uso do sistema.
	É uma atividade que ocorre automaticamente, ligado a algum tipo de serviço, sem necessidade de inferência humana.
	Usado quando a atividade é realizada por uma pessoa com o auxílio de um sistema.
	É uma atividade de envio de mensagem a um participante externo. É similar ao evento intermediário de mensagem**.**
	É uma atividade de recebimento de mensagens de um participante externo. Possui características semelhante ao evento intermediário de chegada de mensagem.
	Usado quando o desempenho de uma atividade existe em um *check list* a ser adotado.

	Indica que a atividade possui vários dados a serem verificados e deve ser especificado o número de vezes que a atividade se repetirá. Exemplo: Se a matriz de uma empresa for verificar os resultados financeiros das filiais, a quantidade de vezes que a atividade se repetirá será a quantidade de filiais existentes.
	O loop (expressão booleana) indica que uma atividade deverá ser repetida até que uma condição estabelecida anteriormente seja cumprida.

Fonte: Ministério Público Federal (2010 - 2014) com adaptações.
Quadro 6: Tipos de Atividades.

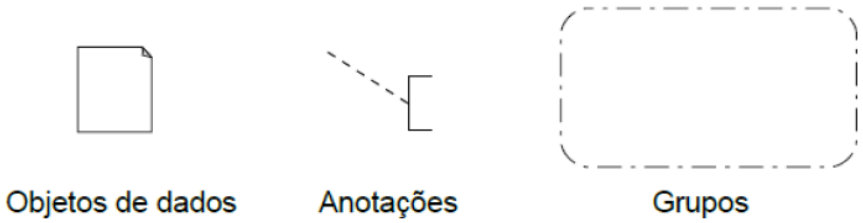
3.3.2 Objetos de Conexão

Os objetos de conexão são elementos que tem a principal função de ligar elementos. Eles determinam a direção e o sentido do fluxo, sem eles o processo ficaria em desordem, ficando impossibilitado compreender o caminho desde a origem ao seu destino. As ligações podem ser entre atividades, gateways e eventos.

Fonte: Autora (2015).
Figura 5: Linhas de conexão.

3.3.3 Artefatos

São elementos que nos ajudam a adicionar informações importantes na modelagem do processo de negócio.



Fonte: Ferreira (2013).
Figura 6: Artefatos.

Objeto de Dados

Representam os dados que serão tratados e manipulados durante o processo de execução. Por exemplo, quando modelamos o processo de afastamento de Professores percebemos que existe um formulário, que é preenchido pelo professor, revisado pela secretária, assinado pelo diretor. Esse formulário influencia diretamente no fluxo, e pode ser representado por um símbolo chamado, objeto de dados (data object). Ele possui um formato de uma folha sem pauta com a ponta dobrada na diagonal.

Anotações

As anotações são elementos que servem para acrescentar informações ao processo. Isso acontece geralmente quando é necessário adicionar algo a modelagem, mas não existe representação direta no BPMN para isso, então para informar por exemplo, a existência de algum produto, material que é produzido por uma atividade, usamos uma anotação. Vale informar que anotações também servem para adicionar notas explicativas, avisos e outros escritos que consideramos importantes para a boa compreensão da modelagem.

Agrupamentos

Para organizar e categorizar as atividades que pertencem a um processo é feito o agrupamento de elementos. O objeto de agrupamento ou grupo é um símbolo caracterizado por traços separado por curtos espaço formando um quadrilátero. Ele pode ser observado na Figura 6.

3.3.4 Piscina (Pool)

As piscinas são elementos BPMN que ficam no fundo do processo e servem para identificar o processo. É nesse elemento que o processo inteiro está contido. Dentro dele colocamos todo o fluxo, com eventos, gateways, atividades, papeis, documentos entre outros. Também com as piscinas é possível identificar um processo, se houver mais de um em um mesmo documento. Na Figura 7 podemos substituir o nome ***Process1***, por afastamento de professores identificando assim o processo.

Fonte: Autora (2015).
Figura 7: Piscinas do BPMN.

3.3.5 Raias (Lanes)

As raias são elementos semelhante as piscinas em aparência visual e em função. Enquanto as piscinas têm a função de identificar o processo, as raias têm a função de identificar quem executa as atividades que estão dentro de determinada raia. A falta das raias dificultaria a leitura da modelagem para o usuário, pois o mesmo não saberia quem estaria executando determinada atividade. As raias são colocadas dentro de uma piscina pois para identificar os diversos papéis no processo. A Figura 8 apresenta duas raias dentro de uma piscina.

Fonte: Autora (2015).
Figura 8: Lane do BPMN.

3.4 Representação do Conhecimento

"O termo inteligência artificial nasceu em 1956 no famoso encontro de Dartmouth. No final dos anos 50 e início dos anos 60, os cientistas Newell, Simon, e J. C. Shaw introduziram o processamento simbólico". (CAMARA, 2000 - 2001).

A representação do conhecimento é uma subárea da inteligência artificial que estuda a forma de como se representa o conhecimento. Existem grandes questões que impulsionaram a criação dessa área como, por exemplo: como representamos o conhecimento para se comunicar com as máquinas de forma que as mesmas derivem, deduzam e infiram novos conhecimentos? Será que todas as pessoas representam o conhecimento da mesma maneira? É possível representar tudo o que existe no mundo de maneira formal?

Segundo Prati (200-?) Uma boa representação deve ser: transparente, rápida e computável, composta por quatro partes: léxica, estrutural, procedural e semântica.

3.4.1 Programação Lógica

De acordo com Sebesta (2003) a lógica simbólica é usada para expressar proposições e relações assim como descrever novas proposições. Prolog é uma linguagem de programação que usa o paradigma declarativo ou descritivo para "expressar programas em fórmula de lógica simbólica e usar um processo de inferência lógica para produzir resultados (SEBESTA, 2003)".

O Prolog foi desenvolvido na Universidade de Aix-Marseille por Alian Colmerauer e Phillippe Roussel com o auxílio e colaboração de Robert Kowalski da universidade de Edinburgh. O propósito de Colmerauer e Roussel era desenvolver a linguagem para processamento de linguagem natural e o de Kowalski era usá-la para demonstração de teoremas. “Nos anos 70 os estudos passaram a ser realizados de forma independente nos dois lugares recebendo limitada atenção” (SEBESTA, 2003). A linguagem conseguiu avançar, mas somente quando foi escolhida para ser a linguagem base de um projeto chamado *Fifth Generation Computing Systems* (FGCS) cujo objetivo era o desenvolvimento de máquinas inteligentes. A partir daí sugiram vários interesses de países da Europa e dos Estados Unidos na inteligência artificial e na programação lógica, o que motivou seu crescimento e expansão (SEBESTA, 2003).

a) Como Funciona?

Para entendermos melhor como funciona o Prolog necessitamos conhecer sua base que é a lógica formal e alguns dos termos utilizados (SEBESTA, 2003). O objetivo dessa primeira é verificar a validade das sentenças.

No dia a dia fazemos afirmações que transmitem informações sobre algo ou alguém, a lógica formal representa essas afirmações que fazemos e as chama de declarações ou sentenças. Denominamos proposição uma sentença ou declaração que pode ser verdadeira ou falsa (GERSTING, 2003), já o “fato”, outro termo utilizado na lógica, é uma proposição que tem necessariamente seu valor verdadeiro. Regras por sua vez são sentenças que ao satisfazerem determinadas condições geram novos conhecimentos a partir dos existentes, geralmente são criadas de fatos.

O interpretador do Prolog “é um analisador capaz de ler programas lógicos e ao ser perguntado consegue fazer a derivação da cláusula que foi perguntada” (ABE, 2002), ele é necessário para resolver as consultas.

O Prolog funciona a base de fatos, regras e consultas. Esses três são seus elementos básicos. Os programas em Prolog são um conjunto de declarações (cláusulas), as quais têm o objetivo de descrever problemas, com estes declarados no programa é possível fazer as consultas, que devem ser respondidas pelo interpretador do

Prolog através de inferências, feitas por deduções lógicas das premissas existentes na base de conhecimento.

De acordo com GESTING (2003) denominamos predicado uma propriedade. Por exemplo “x > 0 ”descreve uma propriedade ou o predicado que x é um número positivo. De acordo com Moura (2003), um predicado é uma declaração e cláusula é uma definição.

Em seguida, mostramos fatos, regras e consultas em Prolog.

Definição 1: Fato

Fato é um termo utilizado para expressar *algo que é*, *que ocorreu* ou *que está ocorrendo.*

Exemplos dos três tipos de fato:

➔ TIPO 1 - Fato que representa uma realidade, dizendo aquilo que é:

Ex.: 1 - Na sentença *“Antônio é um homem”* a afirmação está dizendo aquilo que Antônio é, ou seja essa é um tipo de declaração que define algo. Também podemos dizer que Antônio pertence à classe homem. Em Prolog essa sentença pode ser representada da seguinte maneira:

homem (antonio).

Da mesma forma podemos expressar outros fatos que definem algo ou alguém:

anjo(miguel).	*animal (gato).*	*ave(pomba).*
sentimento(amor).	*objeto(carro).*	*livro(bíblia).*
elemento(terra).	*alimento(pao).*	*material(pedra).*

Neste exemplo, “miguel”, “gato”, “pomba”, “carro”, “amor”, “bíblia”, “terra” e “pao” são constantes pois representam elementos específicos e “anjo”, “livro” “animal”, “sentimento”, “objeto”, “alimento” são predicados unários pois só possuem um elemento.

Em Prolog, sempre colocamos um ponto para finalizar uma cláusula, as constantes escrevemos em letra minúscula e as variáveis colocamos a primeira letra maiúscula, e

geralmente o elemento de quem se fala, fica entre o par de parênteses; e a classificação, ou verbo, fora dos parênteses.

➔ TIPO 2 - Fato que representa um acontecimento, dizendo aquilo que ocorreu: Fatos também podem representar algo que ocorreu.

Ex. 2 - Na sentença "*estudantes ingressaram na universidade"*. O verbo "ingressaram" fica com a função de conector entre os elementos "estudantes" e "universidade", esses dois últimos ficam entre um par de parênteses separados por uma vírgula.

Geralmente quando existe um verbo na sentença esse fica do lado de fora do par de parênteses indicando a relação que existe entre os objetos.

Em Prolog a cláusula do exemplo 2, assumiria s seguinte forma:

entraram (estudantes, universidade).

É válido ressaltar que a leitura da cláusula não importa para o Prolog, o interpretador apenas deriva e descobre se é "verdadeiro ou falso", de acordo com a ordem em que os objetos são colocados.

Ex.: Para o Prolog *"mata (gato, rato)."* é diferente de *"mata (rato, gato)."* .

➔ TIPO 3 - Fatos que representam algo que está ocorrendo neste momento.

Ex 3.: Na sentença "*alunos estão estudando*", não representa um acontecimento passado, no entanto não deixa de ser uma declaração que pode ter seu valor verdadeiro.

Em Prolog representamos esse fato da seguinte maneira:

estao(alunos, estudando).

b) Regras

Uma regra é uma ou mais condições que ao serem satisfeitas validam uma afirmação.

Ex. 4: Para expressarmos uma regra precisamos de uma implicação, em Prolog chamamos de implicação a união de dois símbolos ":" e "-" formando ":-". Na sentença: "*mortal(homem):- homem(antonio).*" estamos criando uma regra com pouca utilidade já que essa regra só tem constantes, isto poderia ser declarado como dois fatos *"mortal (antonio)." e "homem(antonio).".* Uma regra em Prolog geralmente vem com uma ou mais variáveis e não com constantes

A regra acima poderia ser escrita com uma variável, dessa maneira: "*mortal(X):- homem(X).",* indicando que se indivíduo é homem ele também é mortal.

Outros exemplos:

"irmao(X,Y) :- pai(W,Z), pai(W,Y), X=\Y."

"irmao(X,Y) :- mae(W,Z), mae(W,Y), X=\Y."

Nesses casos acima estamos afirmando que para um X ser irmão de um Y, é necessário que eles tenham o mesmo pai ou a mesma mãe e que um seja necessariamente diferente do outro, pois ninguém é irmão de si mesmo.

Um exemplo mais complexo seria afirmarmos a paternidade de um indivíduo. Suponha que X é Pai, precisamos saber se X satisfaz algumas outras condições, X precisa ter pelo menos um filho, e ele precisa ser homem. Quando essas duas condições forem satisfeitas a consequência lógica disso é que X é um pai. Essa só é validada, quando as duas condições (X é homem e X tem um filho) forem satisfeitas.

Em Prolog representamos essa regra como: "*pai(X) :- homem(X), filho(X, _).*" Nesse predicado está informado que para uma pessoa ser pai ela precisa necessariamente ser um homem e ter um filho seja ele qual for, por isso usamos "_" que em Prolog significa variável anônima e serve para evitar o *Warning: Singleton variables* que ocorre todas as vezes que uma variável não se repete em uma cláusula.

No exemplo acima “,” significa o conectivo e, e os símbolos “:” e “-” juntos (“:-”) significam consequência lógica que é uma relação entre premissas e argumento válido, ou implicação. Ao que vem depois dos símbolos “:-” denominamos premissa, se esta ocorrer então o que vem antes é consequência lógica, um fato baseado em um raciocínio lógico válido.

c) Consultas

As consultas em Prolog são questões que proporcionam ao programador, conclusões válidas sobre determinada cláusula, Barbosa (2006). As consultas são idênticas aos fatos, porém são colocadas em locais diferentes e por isso o interpretador diferencia uma da outra.

Quando fazemos uma consulta em Prolog o interpretador busca provar um teorema ou realizar uma dedução, tomando os fatos e as regras que estão no programa como base. Quando o interpretador procura em todo o programa e não consegue provar, responde que a cláusula é falsa.

Um exemplo bem simples de consultas em Prolog é a seguinte:

homem(antonio).	*pai(antonio).*
homem(davi).	*orientador(davi).*
homem(josias).	*pastor(josias).*

Neste exemplo estamos definindo que antonio, davi e josias são homens. E que cada um, além de ser homem tem papéis diferentes, antônio é pai, davi é orientador e josias é pastor. Quando consultamos ao Prolog “homem(X)”, o interpretador responde que X = antonio, X = davi e X = josias, pois os três são homens. Mas se perguntássemos se “orientador(josias).”, o interpretador responderia falso pois na base de conhecimento josias é um pastor.

d) Sintaxe e exemplos

Em Prolog existe um conjunto de regras de sintaxe que especificam como devemos programar na linguagem. Essas regras precisam ser seguidas se desejarmos obter resultados. Por exemplo, ao declarar um fato nesta linguagem e logo em seguida compilarmos (arquivo .pl) sem colocar o ponto final para finalizar o fato o programa

apresentará um erro de sintaxe. Nas consultas, por exemplo, se esquecemos de colocar o ponto ao final da cláusula podemos esperar o quanto quisermos a instrução nunca será executada pois a falta do ponto indica ao interpretador que o programador ainda não terminou de escrevê-la.

Para entendimento das cláusulas escritas em Prolog é necessário conhecermos alguns operadores lógicos e o que significam na linguagem, observe a Figura 9.

Símbolo	Conectivo	Operação Lógica
:-	IF	Implicação
,	AND	Conjunção
;	OR	Disjunção
not	NOT	Negação

Fonte: (LIMA, 2012).

Figura 9: Operadores Lógicos.

Recursos do Prolog 01

Prolog possui recursos definidos pela própria linguagem que podem ser usados pelo programador. Apresentamos a seguir alguns exemplos com listas.

Ex. 05 - Esse exemplo mostra um programa que encontra o último elemento de uma lista.

Código:

```
ultimo([X],X).
ultimo([_|L],X) :- ultimo(L,X).
```

Consultas:

```
ultimo([1,2,3,4],X).
X = 4.
ultimo([1,2,3,4],3).
false.
```

Nesta primeira consulta o programador deseja saber quem é o último elemento da lista, o interpretador busca na lista e encontra valor 4. Já na segunda consulta o programador pergunta se o número 3 é o último elemento da lista e o interpretador responde corretamente com “falso” pois o último é o 4.

Ex. 06: O programa a seguir verifica o primeiro elemento de uma lista.

Código:
```
primeiro([P|_],P).
```

Consultas:

```
primeiro([P|_],P).
true.

primeiro([1,2,3,4,5],P).
P = 1.

primeiro([4,5],P).
P = 4.

primeiro([z,a,1,4,d,f,g],P).
P = z.
```

Na primeira consulta o programador deseja saber se P é o primeiro elemento da lista o interpretador responde que sim. Nos demais exemplos o programador pergunta para cada lista dada quem é o primeiro elemento e o interpretador responde corretamente a todas as consultas isso pode ser verificado observando a resposta e a lista. Geralmente o interpretador não erra na lógica se algo está ocorrendo de forma incorreta o programa feito pelo programador está incorreto.

Ex. 07 – Para concatenar duas listas.

Código:
```
conc([],L,L).
conc([X|L1],L2,[X|L3]):- conc(L1,L2,L3).
```

Consultas:
```
conc([1,8,9],[1,2,3,4,5],L).
L = [1, 8, 9, 1, 2, 3, 4, 5].

conc([1,8,9],[0,4,5],L).
```

L = [1, 8, 9, 0, 4, 5].

Neste exemplo o código mostra como concatenar duas listas. A primeira assim como a segunda consulta inserem como entrada duas listas e o interpretador apresenta como saída uma lista que é o resultado das duas concatenadas.

Ex. 08 - Verificar se um elemento pertence a uma lista.

Código:
```
membro(X,Y):- Y =[X|_].
membro(X,[_|Y]):- membro(X,Y).
```

Consultas:
```
membro(3,[1,2,3,4,5]).
true .

membro(8,[1,2,3,4,5]).
false.
```

Nas consultas acima o programador está consultando se um determinado elemento pertence a uma determinada lista. Lembrando que as listas nesse caso estão sendo declaradas dinamicamente.

3.5 Problemas típicos em modelagem de processos

Como informado na introdução a compreensão do processo é um dos benefícios que pode ser alcançado com uma modelagem em BPMN. Contudo, é difícil atingir a qualidade do processo que facilita a compreensão sem que haja antes revisões e melhorias, com isso para que a modelagem seja o mais íntegro e genuíno possível em relação ao processo original, essas revisões e melhorias tornam-se necessárias.

Um processo modelado de forma incorreta pode acarretar uma visão errada do processo original, dúvidas, entre outros. Segundo Franco (2014), “Modelos de processos de negócio que apresentem erros podem influenciar negativamente na compreensão e execução desses processos”. Como o objetivo de modelar e facilitar a comunicação, é praticamente indispensável que o processo esteja bem modelado e seja autoexplicativo.

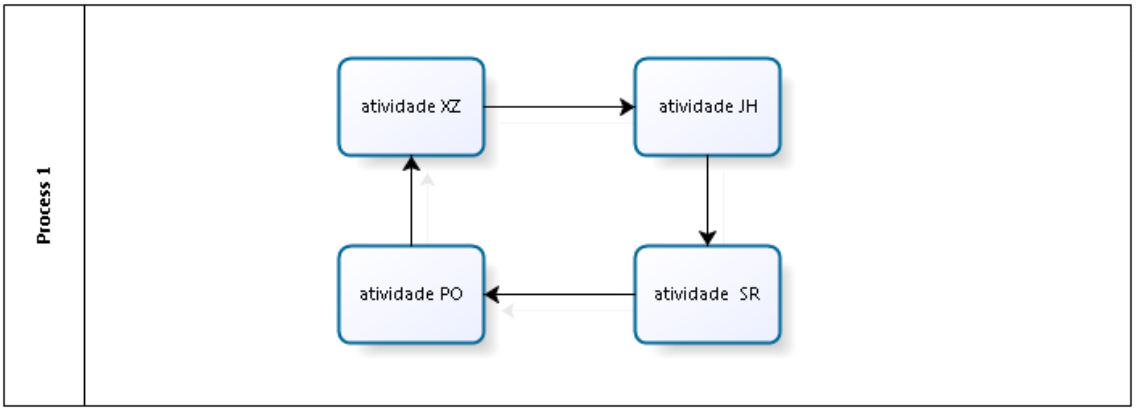
Apresentamos a seguir um escopo de quatro tipos de inconsistências que podem surgir no momento da modelagem de processo em BPMN. E na seção 5, exibimos como identificá-las.

1º Falta de símbolos quando há necessidade:

- É possível criarmos BPDs que não possuem eventos iniciais e finais, contudo o diagrama fica errado, porque esses símbolos são obrigatórios a qualquer diagrama, por mais simples que sejam. Eles têm a função de determinar onde o processo começa e termina.
- Também é possível criar BPDs sem raias que são importantíssimas pois indicam quem está executando o Processo, seja uma pessoa ou um sistema; um processo sem raia gera dúvidas no leitor, e acaba desinformado sobre quem executou a atividade. É necessário que exista um autor para cada ação de um processo, pois sempre que algo é executado, é executado por um executor.
- É possível criar diagramas sem eventos intermediários, temporizadores e outros que muitas vezes passam sutilmente despercebidos, mas sua falta altera o processo genuíno. Um bom exemplo disso é o evento intermediário temporizador, que serve para indicar a existência de uma tolerância de tempo para se executar uma atividade que veio seguida de outra. Sem o evento de tempo a leitura do processo é modificada, pois quem está lendo acredita que após uma determinada atividade A, a atividade B é logo executada, sendo que na realidade a atividade B só pode ser executada 15 dias depois do término da atividade A.

Na Figura 10, que foi elaborada propositalmente com a falta de alguns símbolos, podemos perceber quais dúvidas concebem na mente do leitor:

1- Quem executa as atividades?
2- Onde o processo começa?
3- Onde o processo termina?
4- Qual a condição de parada do ciclo?

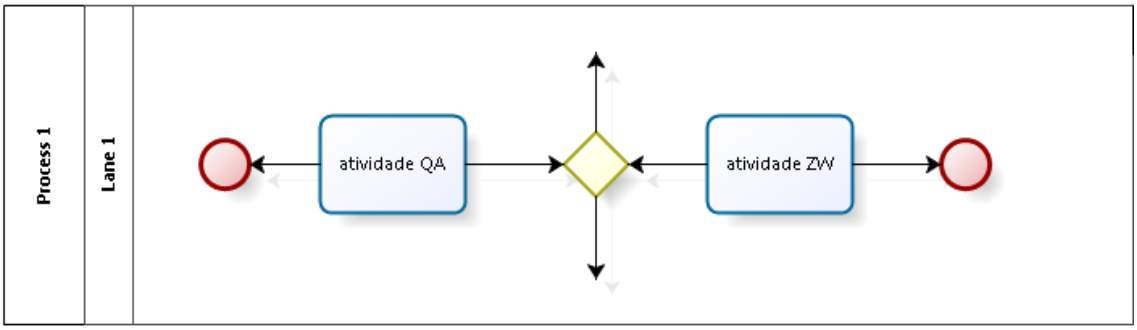


Fonte: Autora (2016).

Figura 10: Processo em BPMN incompleto.

Na Figura 10 não é possível identificar onde o processo começa nem onde ele termina, nem ao menos quem executa cada uma das atividades; assim, fica difícil o entendimento do processo modelado. A Figura em discussão necessita de vários símbolos para facilitar a leitura visual do processo, falta um evento de início, um evento de fim, de pelo menos uma raia e gateways que controlem o ciclo.

As inconsistências podem se agravar, a seguir o exemplo mostra um processo que termina, mas que não começou. Todas essas irregularidades podem ser feitas porque as ferramentas de modelagem permitem a combinação de símbolos sem impor ao analista a atividade obrigatória de colocar um evento inicial toda vez que for modelar um processo, a ferramenta avisa em alguns casos de erros, mas como é possível salvar de qualquer maneira é necessário que as modelagens sejam de alguma forma verificadas depois de feitas para que sejam corrigidas.

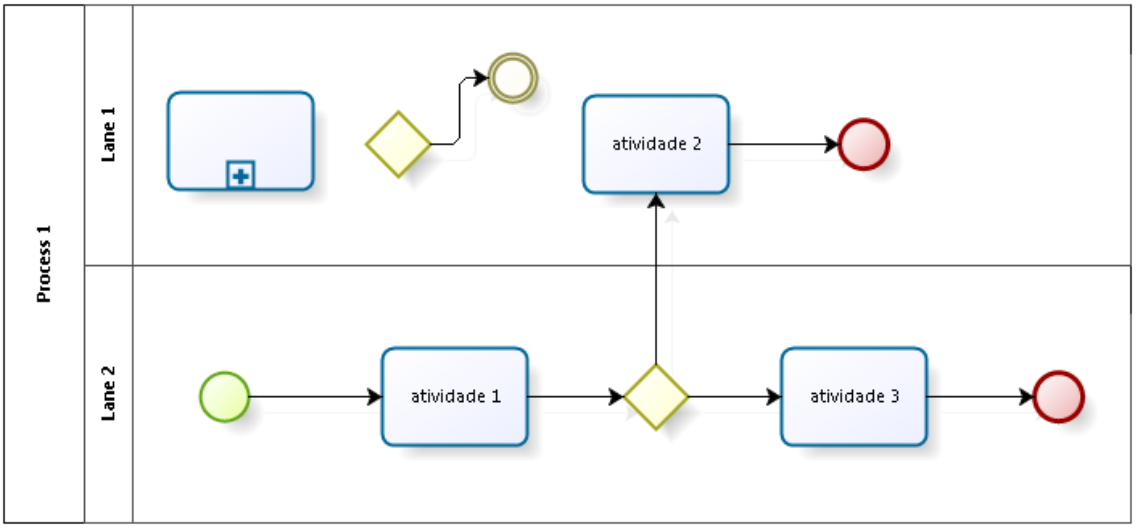


Fonte: Autora (2016).

Figura 11: Erros em modelagem, processo termina sem início.

Além de terminar o que não começou, o que é incoerente, não existe a mínima condição de saber que atividade é a origem do gateway, se é a atividade QA ou a atividade ZW. Além disso, ainda existem dois fluxos de destino saindo de um gateway para o espaço, ao qual não estão ligados a nenhum outro símbolo.

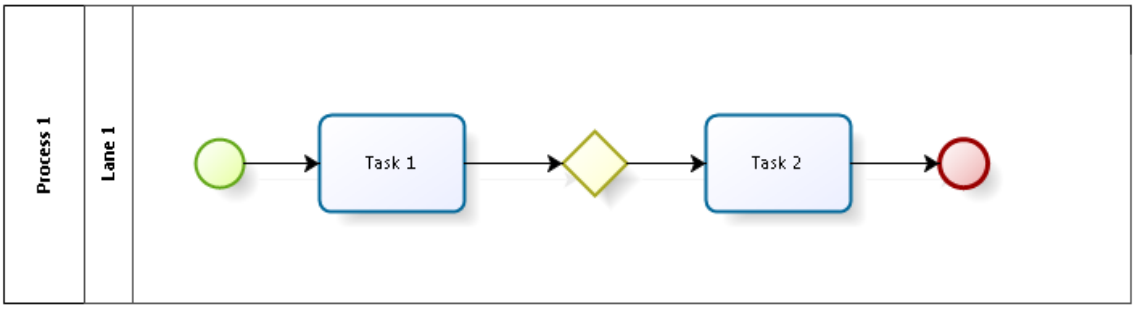
2º Símbolos desconectados do fluxo: É possível colocar símbolos que não estão conectados ao fluxo principal (aqueles que possuem início e fim claramente definidos) e que não tem nenhuma utilidade, deixando o diagrama errado. Observe a Figura 12:



Fonte: Autora (2016).
Figura 12: Símbolos soltos no diagrama.

O Sub-processo e o evento intermediário seguido de um gateway, não têm utilidade no diagrama acima, logo devem ser removidos, uma vez que estão apenas poluindo a modelagem visual do processo.

3º Mau uso de símbolos: É possível também criarmos símbolos, como gateway, com apenas uma saída, o que o deixa sem utilidade.



Fonte: Autora (2016).
Figura 13: símbolos mal usados.

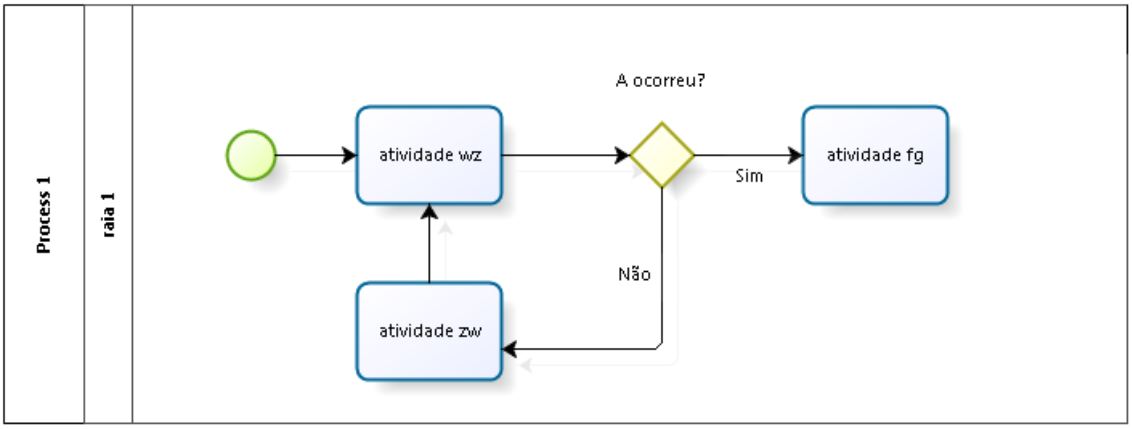
Gateways são utilizados para divergir ou convergir o fluxo, se não existe essa necessidade, o gateway não precisa ser utilizado. Na Figura 13, o gateway tem função de ligar a task 1 para a task 2, essa função não é para esse símbolo, é atribuição das linhas de conexão, o gateway está sendo mal utilizado.

Os símbolos da notação BPM, não podem ser utilizados de qualquer forma, eles sempre se encaixam em situações específicas, veja:

1. Linhas de sequência fazem ligação entre símbolos, com exceção a anotações e objetos de dados que devem ser ligados por linhas de associações.
2. Gateways devem estar ligados a no mínimo 3 linhas de sequências onde elas podem vim como duas entradas e uma saída, ou uma entrada e duas saídas.
3. Eventos intermediários sempre tem entradas e saídas, já eventos de início só possuem saídas e eventos de fim somente entradas.

Colocar um gateway com apenas uma entrada e uma saída em um diagrama, retira sua função primordial que é a convergência ou divergência de dois fluxos, e torna sua presença inútil na modelagem.

4º Ciclos: Os ciclos podem, em alguns casos, ser problemas. Por exemplo, um processo entra em ciclo por decisão automática e esse ciclo nunca rompe porque a condição para rompê-lo nunca é satisfeita. Observe a Figura 14.



Fonte: Autora (2016).

Figura 14: ciclos em modelagens de processos.

Uma solução para problemas como esse é determinar um certo limite para entrar na condição que liberta o processo do ciclo, se a condição não ocorrer até esse limite, alguma providência pode ser tomada, como por exemplo: cancelar o processo ou reiniciá-lo.

4 PROCEDIMENTOS METODOLÓGICOS

Nesta seção apresentamos os procedimentos utilizados durante a execução deste trabalho. Apresentamos desde a fase do estudo dos elementos BPMN até a fase final, que foi elaborar um mapeamento geral automático que traduz modelos de processos em BPMN para Prolog.

4.1 Escolha da ferramenta a ser utilizada para modelar processos

Dentre as diversas ferramentas que existem para modelar um processo na notação BPMN, escolhemos a ferramenta BIZAGI. Marques e Silva (2012) fizeram um estudo comparativo entre ferramentas BPMS e perceberam que BIZAGI além de permitir o usuário gerar relatórios interessantes, a ferramenta também é de fácil aprendizado, utilização e apresenta boa usabilidade.

4.2 Modelagem do processo de afastamento de Professores em BPMN

Após entender o processo de afastamento de professores fizemos a modelagem utilizando a notação do BPM. Para isso, utilizamos a ferramenta BIZAGI Modeler.

4.2.1 Tradução do Processo de Afastamento de Professores para Programação Lógica

Após concluir a modelagem, a próxima etapa foi traduzir o processo de afastamento de professores para Prolog.

4.2.2 Consultas em Programação Lógica

Para finalizar nosso caso de estudo fizemos algumas regras em Prolog para a partir delas realizar consultas em programação lógica e encontrar inconsistências e erros. Essas regras que elaboramos podem ser aplicadas em outros estudos.

4.3 Mapeamento geral que traduz qualquer processo em BPMN para Prolog

A última etapa deste trabalho foi elaborar um programa que faz um mapeamento automático de diagramas em BPMN para Prolog. O programa lê um arquivo XML de um BPD e transforma-o em uma ontologia em Prolog. Com isso foi possível fazermos alguns casos de teste e verificar a qualidade do programa (tradutor).

5 MAPEAMENTO GERAL (Diagramas BPMN para Programação em Lógica)

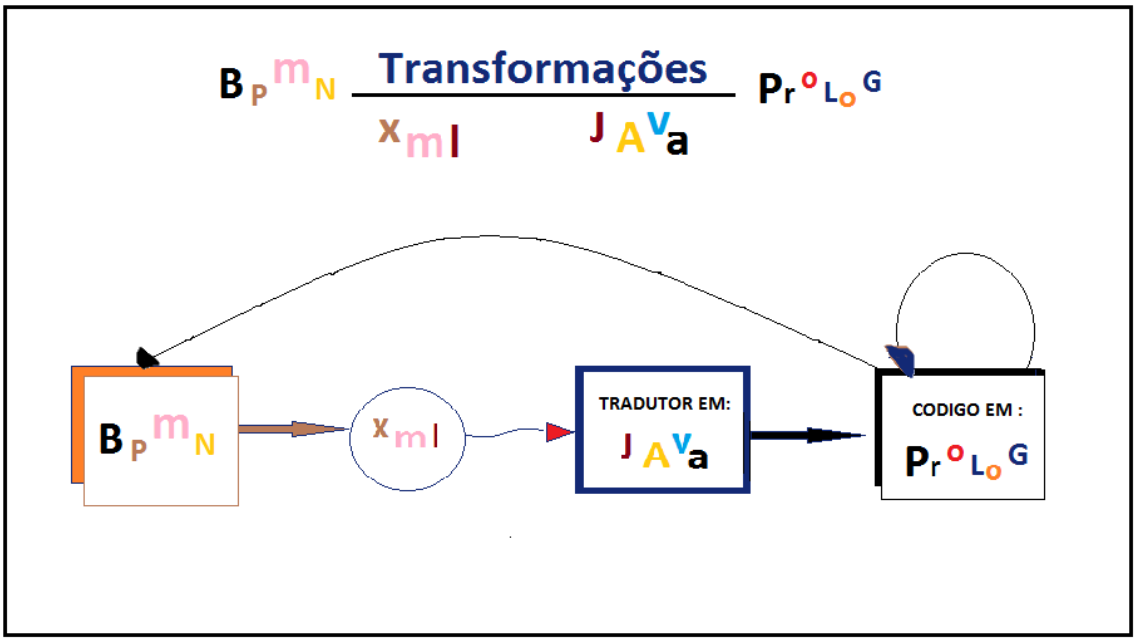
O *eXtensible Markup Language* (XML) assim como o *HyperText Markup Language* (HTML) são linguagens de marcação (ALMEIDA, 2002). O XML é muito utilizado para manipular e estruturar arquivos na internet (ALMEIDA, 2002). No entanto, pode ser usado para outros tipos de aplicações, sendo uma delas o BPMN. Quando geramos código XML de um arquivo em BPMN, possibilitamos a manipulação automática desse arquivo de forma exata para outras linguagens. Essas manipulações podem ser usadas para mapeamentos, edição, leitura, criação de arquivos XML, entre outros.

O BIZAGI e o BONITA são ferramentas usadas para fazer modelos de processos. O BIZAGI, além de permitir a criação de BPDs também disponibiliza uma série de outros documentos que podem ser gerados a partir do diagrama como: código XML, imagens, código em XPDL entre outros.

Neste trabalho, para encontrar inconsistências em BPDs, modelamos processos de negócio em BPMN, em seguida transformamos a modelagem visual, o BPD, em código XML, e enviamos esse último para o tradutor que desenvolvemos, ao qual lê códigos em XML de BPDs e traduz para códigos em Prolog.

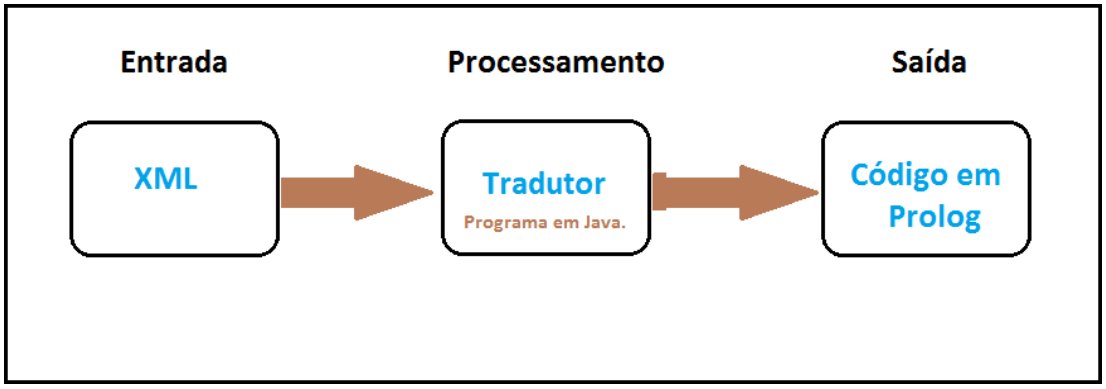
Em seguida explicamos como traduzimos. Elaboramos alguns fluxos que explicam como fazemos a tradução no construtor e depois apresentamos algumas consultas para encontrar inconsistências, erros e falhas, para assim sugerir algumas melhorias nas modelagens de processos de negócio. A seguir como elaboramos:

Podemos visualizar na Figura 15 as fases ou passos necessários para melhorar processos de negócio. Começando da modelagem BPM até o código Prolog referente a essa modelagem. Após chegarmos na fase em que dispomos do código em Prolog, podemos observar que existe um ciclo na Figura 15, indicando que podemos fazer consultas no próprio Prolog e caso encontrarmos pontos de melhoria, esses são enviados para que a modelagem visual seja editada. Após a edição do desenho, todos os passos podem ser refeitos visando uma verificação e procura de outras inconsistências.



Fonte: Autora (2016).
Figura 15: Desenho ilustrativo das fases deste trabalho.

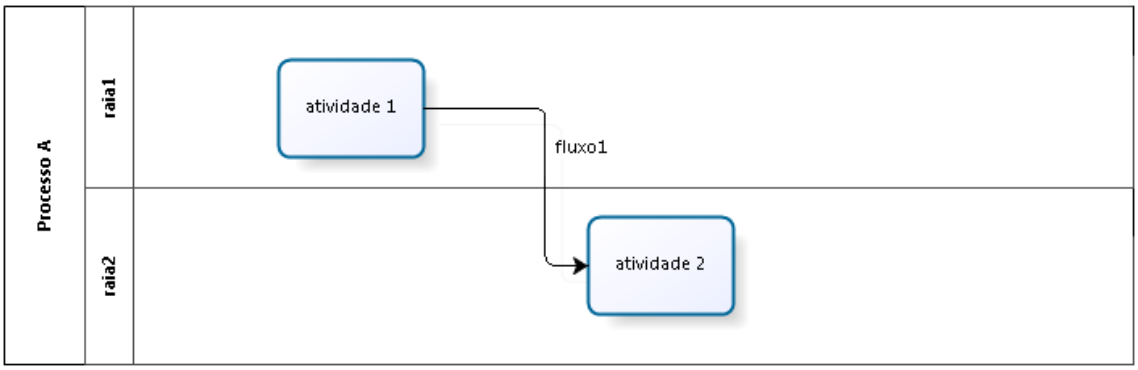
A Figura 16 representa como ocorre o processo de tradução de códigos XML de BPDs para códigos em Prolog. Elaboramos este tradutor usando para isso uma linguagem intermediária, a saber, o JAVA na aplicação do DOM W3C. Esse tradutor recebe um documento XML e a entrega um código em Prolog referente ao XML.



Fonte: Autora(2016).
Figura 16: Desenho ilustrativo do processo de tradução.

Apresentamos a seguir alguns exemplos que mostram como mapeamos BPMN para Prolog.

Ex. 1: O Exemplo 1, é o retrato de um fluxo simples criado na modelagem visual, um BPD. Esse retrata a execução de duas atividades não paralelas.



Fonte: Autora (2016).
Figura 17: Fluxo simples em BPMN.

A Figura 17 tem possui um código XML que a representa unicamente, esse foi transformado em Prolog, apresentamos a seguir os dois seguindo a ordem de conversão mostrada na Figura 15. A tabela abaixo apresenta as partes relevantes que foram consideradas para o processo de tradução de XML para Prolog. Os elementos e sub elementos geralmente possuem um "<" antes do seu nome; os atributos estão sempre dentro dos elementos ou dos sub elementos e os valores dos atributos geralmente vêm logo depois do símbolo "=" e dentro de um par de aspas. Observe o elemento Lane para entender melhor:

```
<lane id="Id_d05be0d1-7d2c-4534-8a9e-cac72dddd9c9" name="raia1">.
```

lane: elemento.

id: atributo.

Id_d05be0d1-7d2c-4534-8a9e-cac72dddd9c9: valor do atributo.

name: atributo.

raia1: valor do atributo.

Observe as linhas 1, 2, 3, 4, 6... os elementos que estão delimitados por < >.

Observe primeiro o código em XML, logo em seguida, na Tabela 1:

Tabela 1 – Partes do XML referente ao exemplo 1. Fonte: adaptado de bizagi (2016).

```
1 <?xml version="1.0"?>
2 <definitions...
3 <process id="Id_bcfcf448-9739-4856-be99-1fca32bf269c">……
4 <lane id="Id_d05be0d1-7d2c-4534-8a9e-cac72dddd9c9" name="raia1"> ….
5 ..... </lane>
6 <lane id="Id_63014e22-6fec-432c-b4c7-8f886e0f8878" name="raia2">….
7 ..... </lane>
8 <task id="Id_f4d65854-cbb8-4886-88b9-847a50be32b3" name="atividade
9 1">…
10.....</task>
11 <task id="Id_fc11f103-9982-4454-ac36-c8fde9bfe300" name="atividade
12 2">…….…..</task>
13 <sequenceFlow id="Id_da4bcf9c-f7c8-40b4-bc24-f690da11a4c9"
14 name="fluxo1" sourceRef="Id_f4d65854-cbb8-4886-88b9-847a50be32b3"
15 targetRef="Id_fc11f103-9982-4454-ac36-c8fde9bfe300">.
16 </process>
17 <participant id="Id_140a43da-3150-45f9-b3b6-703667a92174"
18 name="ProcessoA" processRef="Id_bcfcf448-9739-4856-be99-
19 1fca32bf269c">
20 </participant>
21 <BPMNShape id="DiagramElement_a5b046cb-5aa8-4693-a106-
22 c394427d0cf2" bpmnElement="Id_d05be0d1-7d2c-4534-8a9e-
23 cac72dddd9c9" isHorizontal="true">
24.......<Bounds x="50" y="30" width="650" height="112"…. />
25 </BPMNShape>
26 <BPMNShape id="DiagramElement_e18749d5-434a-4250-be3c-
27 e62f5252f5bd" bpmnElement="Id_63014e22-6fec-432c-b4c7-
28 8f886e0f8878" isHorizontal="true">
29 .......<Bounds x="50" y="142" width="650" height="109"…. />
30 </BPMNShape>
31 <BPMNShape id="DiagramElement_6a94a161-b7a1-4c41-a896-
```

```
32 292a93aac639" bpmnElement="Id_f4d65854-cbb8-4886-88b9-
33 847a50be32b3">
34 .......<Bounds x="201" y="63" width="90" height="60"…. />
35 </BPMNShape>
36 <BPMNShape id="DiagramElement_ef81c2d2-2c8d-4a0d-a12d-
 9f34caf174fc" bpmnElement="Id_fc11f103-9982-4454-ac36-
37 38 c8fde9bfe300">
39........<Bounds x="393" y="160" width="90" height="60"…./>
40 </BPMNShape>
```

Na tabela acima, ao qual está enumerada, o elemento ***process*** na linha 3 que é encerrado na linha 16, é o primeiro elemento depois do elemento raiz (definitions), que é um componente visual e tem um atributo ***id*** (linha 3) que serve para o identificar unicamente, como é possível criarmos mais de um processo em um documento visual, ele também possui um atributo ***nome*** (linha 18)**,** no entanto esse atributo não vem no mesmo elemento ***process*** é possível capturá-lo comparando o ***id*** (linha 3, 18 e 19) do processo em questão com o atributo ***processRef*** (linha 18 e 19) do elemento ***participant*** (linha 17)**,** esse último possui um atributo chamado name (linha 18). Observe essa informação na Tabela 1, acima.

A seguir apresentamos os elementos do XML usados para a conversão para Prolog:

- Elemento **lane** (linhas 4, 5, 6 e 7): possui um id que a identifica unicamente, um nome e uma localização, vele informar que essa localização pode ser buscada em outro elemento comparando-se ids, pois a mesma não vem no elemento lane.
- Elemento **task** (linhas: 8 e 11)**:** possui os mesmos atributos de lane.
- Elemnto **sequenceFlow** (linha 13): possui os atributos: nome, id, sourceRef e targetRef.
- Elemento **BPMNShape** (linhas 21, 26, 31 e 36): possui o atributo id e o atributo bpmnelemnt, também possui um sub elemento chamado ***Bounds*** (linhas: 24, 29, 34 e 39).

- Sub elemento **Bounds**: esse possui 4 atributos principais: x, y, widht e height, que servem para indicar a localização de um símbolo no diagrama.

O elemento ***lane*** (linhas 4, 5, 6 e 7) ou raia, representa o papel de quem executa as atividades que estão dentro dela, observe na Figura 17 os elementos que estão dentro da raia1 e da raia 2.

O elemento ***task*** (linhas: 8 e 11) tem a finalidade de indicar uma ação que é executada por uma pessoa ou por um sistema.

O elemento **sequenceFlow**(linha 13) é uma linha de sequência que estabelece conexão entre dois símbolos esses podem ser **tasks**, **gateways**, **eventos**, entre outros. A ligação pode ser feita entre duas atividades, dois gateways, ou mesmo uma combinação feita como, um gateway e uma atividade, um evento e uma atividade, entre outros.

O elemento ***BPMNShape*** (linha 21) tem a finalidade de situar um elemento dentro da modelagem visual, ou seja, ele mostra exatamente onde está localizado determinado elemento através dos atributos x, y,widht e height do sub elemento: ***Bounds*** (linhas: 24, 29, 34 e 39).

A partir desses elementos e suas informações elaboramos nossa entidade básica em Prolog que será um fato denominado fluxo, que mapeia a sequência de fluxo de BPMN na seguinte forma: “ **`fluxo`** `(X, Y, Z, T, W).` ”, onde:

- **X:** indica a ação inicial executada no processo vindo na maioria das vezes de uma atividade, vale ressaltar que esta ação pode ser oriunda de um evento de início.
- **Y:** indica quem está executando a ação de X. Ou seja, Y é o nome da raia ao qual X está contido.
- **Z:** é uma ação executada posteriormente a X.
- **T:** indica quem está executando a ação de Z. Ou seja, T é o nome da raia ao qual Z está contido.
- E por fim **W:** nome dos rótulos que estão nas linhas de sequência que ligam X a Z. Esses quando obtidos por um gateway, ao qual explicaremos mais adiante como foi representado em Prolog, são mais de um e por isso essa variável W é uma lista.

O fluxo da Figura 17 foi representado em Prolog como:

- ***fluxo*** *(atividade1, raia1, atividade2, raia2, [fluxo1]).*, é uma fato que indica duas ações, uma de origem (atividade 1) e outra de destino (atividade 2) executada por dois agentes (raia 1 e raia 2), onde esses dois últimos podem ser sistemas ou pessoas, e possui um ou mais rótulos ([fluxo1]).

Agora apresentamos como passamos do XML para o Prolog. Nosso fluxo é uma sequenceFlow que denominamos "fluxo", que possui os atributos souceRef e targetRef, esses têm suas localizações indicando em qual lane está, informando quem executa a sourceRef ou a targetRef. Observe abaixo as formas.

- **fluxo**(X,Y,Z,T,rótulo).
- **sequenceFlow**(sourceRef, sourceRef_localizacao, targetRef, targetRef_localizacao, [sequenceFlow.name]).
- **sequenceFlow**(task.name, lane.name, task2.name, lane2.name, [sequenceFlow.name]).

Primeiro analisamos o elemento ***sequenceFlow*** (linha 13) da tabela 1, seu nome (linha 14) é o nosso rótulo último parâmetro do fluxo do Prolog, que pode existir ou não, isso dependerá do analista de processos decidir colocar ou não. Caso ele não coloque o tradutor apenas coloca string vazia ou variável anônima, indicando que não há necessidade daquele rótulo.

O ***sourceRef*** indica a nossa atividade de origem (X), ou seja, de onde nosso o fluxo está saindo e o ***targetRef*** indica nossa atividade de destino (Y), ou seja, aonde o nosso fluxo está chegando.

Para descobrirmos com quais atividades os fluxos estão relacionados comparamos os ***ids*** das ***tasks*** com os atributos ***sourceRef*** e ***targetRef*** do ***sequenceFlow***. Se o ***id*** da ***task*** for igual ao ***souceRef*** do ***sequenceFlow*** então a atividade é de origem e o portanto precisa ser colocada no primeiro parâmetro (X) do fato em Prolog, colocamos o atributo ***name*** como primeiro parâmetro (X) do fluxo. Se o ***id*** da ***task*** for o ***targetRef*** do ***sequenceFlow*** então a atividade será de destino e portanto o seu nome será o terceiro parâmetro (Z) do fluxo. Observe os Ids das linhas 9 e 12 com os Ids das linhas 17 e 18 grifados na tabela seguinte:

```
1 <?xml version=”1.0”?>
2 <definitions...
3 <lane id="Id_d05be0d1-7d2c-4534-8a9e-cac72dddd9c9"
4 name="raia1"> ….
5..... </lane>
6 <lane id="Id_63014e22-6fec-432c-b4c7-8f886e0f8878"
7 name="raia2">….
8..... </lane>
9 <task id="Id_f4d65854-cbb8-4886-88b9-847a50be32b3"
10 name="atividade 1">…
11 .....</task>
12 <task id="Id_fc11f103-9982-4454-ac36-c8fde9bfe300"
13 name="atividade 2">….
14 …..</task>
15 <sequenceFlow id="Id_da4bcf9c-f7c8-40b4-bc24-
16 f690da11a4c9" name="fluxo1" sourceRef="Id_f4d65854-
17 cbb8-4886-88b9-847a50be32b3" targetRef="Id_fc11f103-
18 9982-4454-ac36-c8fde9bfe300">.
19 <BPMNShape id="DiagramElement_a5b046cb-5aa8-4693-a106-
20 c394427d0cf2" bpmnElement="Id_d05be0d1-7d2c-4534-8a9e-
21 cac72dddd9c9" isHorizontal="true">
22 ....<Bounds x="50" y="30" width="650" height="112"…. />
23 </BPMNShape>
24 <BPMNShape id="DiagramElement_e18749d5-434a-4250-be3c-
25 e62f5252f5bd" bpmnElement="Id_63014e22-6fec-432c-b4c7-
26 8f886e0f8878" isHorizontal="true">
27...<Bounds x="50" y="142" width="650" height="109"…. />
28 </BPMNShape>
29 <BPMNShape id="DiagramElement_6a94a161-b7a1-4c41-a896-
30 292a93aac639" bpmnElement="Id_f4d65854-cbb8-4886-88b9-
31 847a50be32b3">
32.....<Bounds x="201" y="63" width="90" height="60"…. />
33 </BPMNShape>
```

```
34 <BPMNShape id="DiagramElement_ef81c2d2-2c8d-4a0d-a12d-
35 9f34caf174fc" bpmnElement="Id_fc11f103-9982-4454-ac36-
35 c8fde9bfe300">
36....<Bounds x="393" y="160" width="90" height="60"….>
37 </BPMNShape>
```

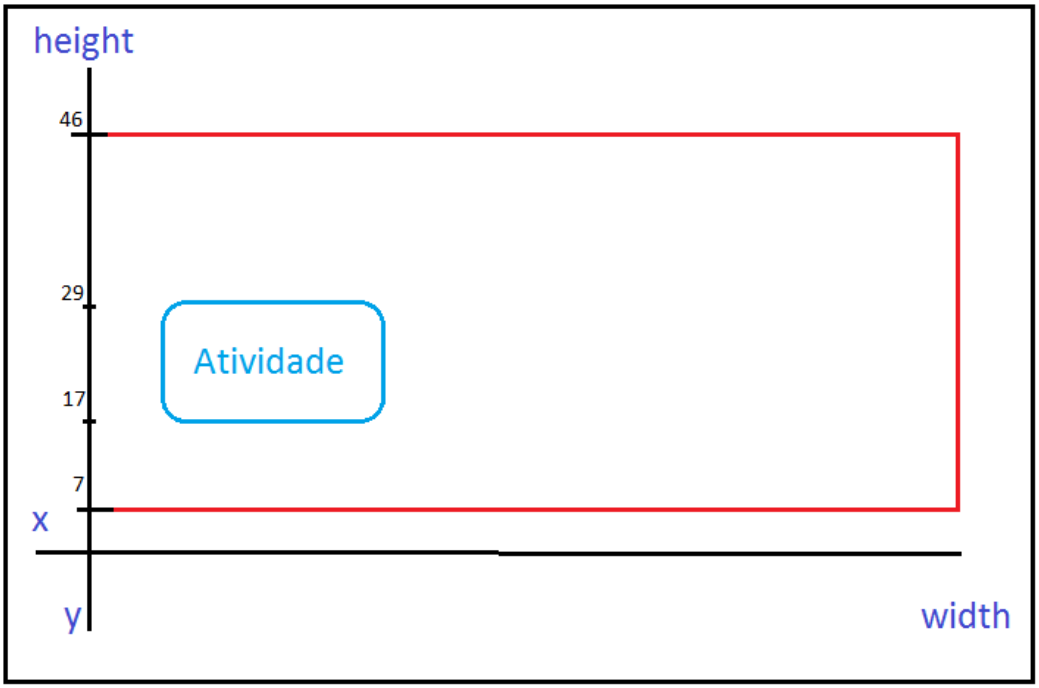
Após calcular as atividades de origem e destino do fluxo precisamos saber onde estão localizadas, em qual raia, pois a mesma indica quem executa a atividade. Para essa informação, o XML do documento nos disponibiliza a localização através do elemento ***BPMNShape*** (linha 18) que em um dos seus sub elementos a informação necessária para nosso trabalho, essa informação vem como pontos de uma coordenada.

O processo de cálculo ocorre da seguinte maneira, primeiro comparamos o atributo ***bpmnElement*** (linha 19 e 20) que traz um id (linha 19 e 20) , esse deve ser comparado com o id (linha 3) do símbolo em análise, se o atributo ***bpmnElement*** (linha 19 e 20) do elemento ***BPMNShape*** for igual ao id de um dos símbolos significa que esse símbolo está delimitado nas coordenas do sub elemento, Bounds (linha 22), de ***BPMNShape*** (linha 19); esses pontos são: ***width, x, height*** e ***y*** e indicam respectivamente começo do comprimento, término do comprimento, começo da altura e término da altura.

Para descobrirmos se determinado símbolo ao qual chamaremos de A está localizado em determinada raia que chamaremos neste caso de R precisamos conferir se o atributo Y de A é maior que o atributo Y de R e se Y de A é menor que a soma do atributo Y e height de R. Assim descobrimos se a atividade A está dentro das coordenadas de R.

- **Cálculo:**
- (Y de A >= Y de R) ^ (Y de A >= soma (Y de R + height de R)).
- Se o retorno desta cláusula for verdadeiro A está dentro de R, caso contrário não está.

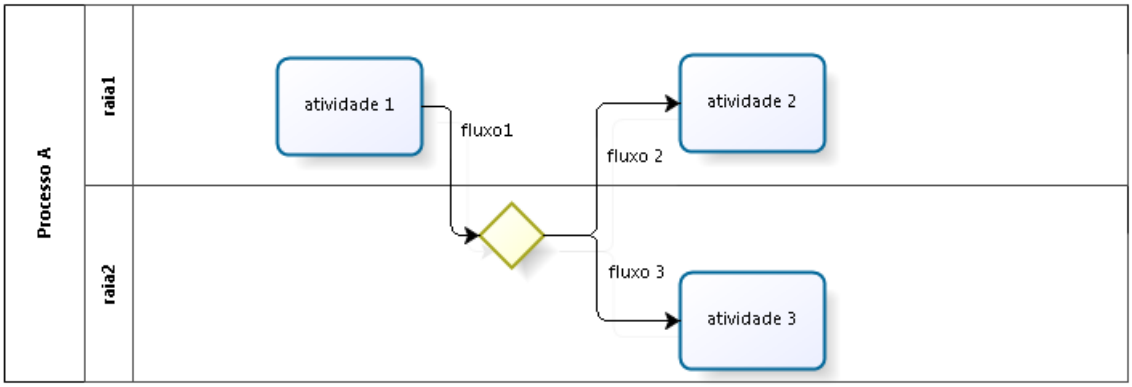
Observe a Figura 18:



Fonte: Autora (2016).
Figura 18: delineações do espaço.

O retângulo delimitado entre os pontos 7 e 46 simboliza a raia e o símbolo que está dento do retângulo que nesse caso é uma atividade simboliza não só um símbolo do tipo atividade, mas qualquer símbolo que esteja dentro de uma raia.

Ex. 2: O Exemplo 2, é um pouco mais complexo que o 1, pois ele traz a presença de um símbolo mais complexo, o gateway.



Fonte: Autora (2016).
Figura 19: Fluxo com gateway em BPMN.

O gateway é um símbolo que não representa uma ação propriamente dita, ele representa uma convergência ou divergência de fluxos. Dado isso optamos por representá-lo como uma regra. Essa regra diz que se dois fluxos possuem a mesma origem ou se dois fluxos possuem o mesmo destino, entre esses está presente pelo menos um gateway.

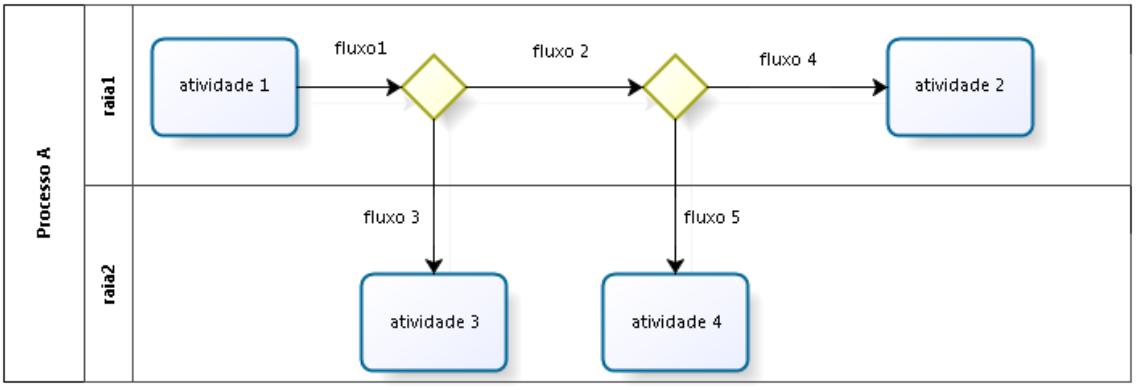
Representamos a regra do gateway da seguinte forma:

```
gateway(O,Y):-fluxo(O,Y,A,B,_),fluxo(O,Y,C,D,_), A\=C.
gateway(O,Y):-fluxo(A,B,O,Y_),fluxo(C,D,O,Y_), A\=C.
```

O programa em Prolog referente ao fluxo da Figura 16 é composto por dois fluxos, que tem mesma origem:

```
o  fluxo (atividade1, raia1, atividade2, raia1, [fluxo1,fluxo2]).
o  fluxo (atividade1, raia1, atividade3, raia2, [fluxo1,fluxo3]).
```

Ex. 3: Para o exemplo 3, definimos uma função recursiva para capturar os fluxos que existem intermediados pelos gateways. Essa função foi necessária porque não é possível capturar as atividades de destino de um fluxo se existem gateways entre elas. No XML e na modelagem visual, o que liga os símbolos são as linhas de conexão, como neste caso não existem conectores diretos entre as atividades, uma função recursiva é necessária para percorrer os gateways até encontrar o destino do fluxo, que no caso é uma atividade. Na figura 20 os destinos são atividades 2, 3 e 4, ou seja, os casos de parada. Observe a Figura 20.



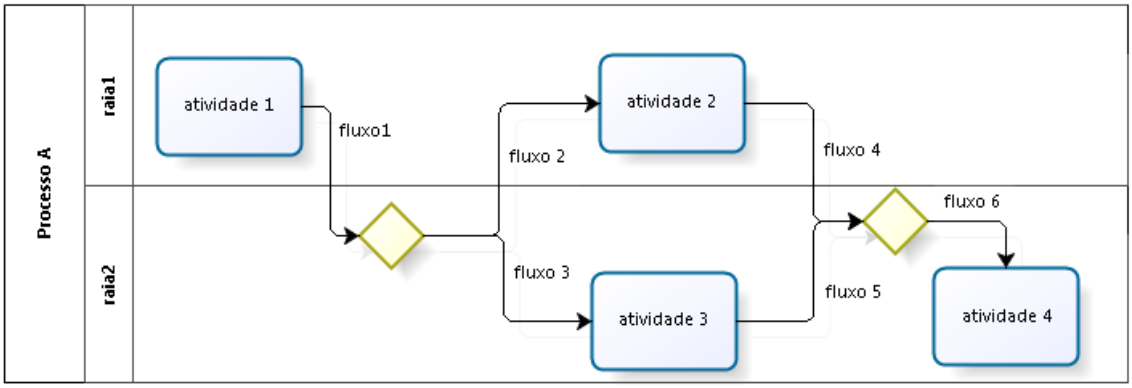
Fonte: Autora (2016).
Figura 20: Fluxo com dois gateways em BPMN.

O programa em Prolog referente a Figura 20 é composto por três fluxos que tem a mesma origem, que no caso é a atividade 1, mas possuem destinos diferentes a saber atividade 2, 3 e 4.

```
o fluxo (atividade1, raia1, atividade2, raia1,
  [fluxo1,fluxo2,fluxo4]).
o fluxo (atividade1, raia1, atividade3, raia2, [fluxo1,fluxo3]).
o fluxo (atividade1, raia1, atividade4, raia2,
  [fluxo1,fluxo2,fluxo5]).
```

A lista de rótulos serve para indicar o caminho que está sendo trilhado da atividade de origem à atividade de destino.

Ex. 4: Este exemplo ilustra um caso que não é comum ocorrer, mas que pode estar presente em alguns casos, ele pode ser utilizado para representar alguma divergência de atividades paralelas e logo em seguida a convergência dessas mesmas.

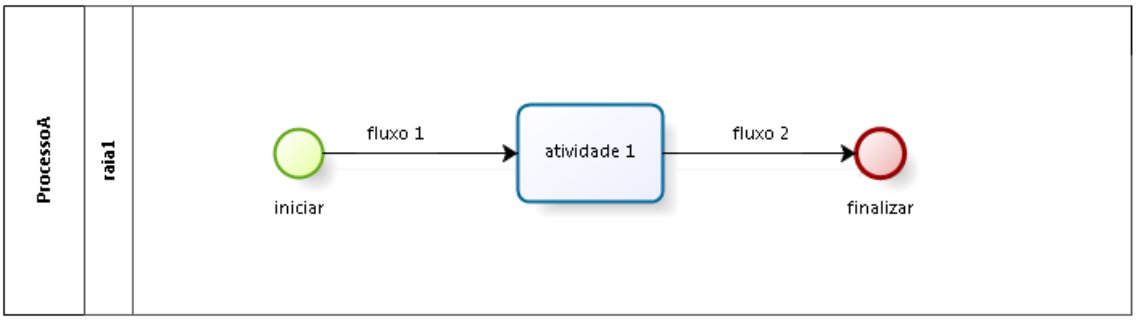


Fonte: Autora (2016).
Figura 21: Fluxo com um gateway do tipo divergente e outro do tipo convergente em BPMN.

O programa em Prolog referente a Figura 21 é composto por quatro fluxos, onde dois possuem a mesma origem, e dois possuem o mesmo destino. Os dois que possuem a mesma origem tem em comum um gateway de divergência e os dois fluxos que possuem o mesmo destino tem em comum um gateway de convergência, os mesmos estão listados a seguir:

```
o fluxo (atividade1, raia1, atividade2, raia1, [fluxo1,fluxo2]).
o fluxo (atividade1, raia1, atividade3, raia2, [fluxo1,fluxo3]).
o fluxo (atividade2, raia1, atividade4, raia2, [fluxo4,fluxo6]).
o fluxo (atividade3, raia2, atividade4, raia2, [fluxo5,fluxo6]).
```

Ex. 5: Todos os exemplos anteriores retratam apenas partes genéricas de um processo, esse porém representa um processo completo genérico. Ou seja, ele possui todos os elementos obrigatórios que um processo por mais simples que seja deve ter: evento de início, evento de fim, atividade, processo, raia e as linhas de conexão.



Fonte: Autora (2016).
Figura 22: Fluxo apresentando processo completo mais simples em BPMN.

Esse processo contém 5 símbolos e um deles ocorre duas vezes. O primeiro é a raia, o segundo é o evento de início, o terceiro e o quinto são duas linhas de ligação, o quarto é uma atividade e o sexto o evento de fim.

A tradução desse processo seguindo a forma apresentada neste trabalho é a composição de dois fluxos:

```
o  fluxo (iniciar, raia1, atividade1, raia1, [fluxo1]).
o  fluxo (atividade1, raia1, finalizar, raia1, [fluxo2]).
```

Os símbolos evento de início e fim foram representados como uma atividade sendo que essa atividade é caracterizada pelo seu tipo de evento.

Para todos os símbolos startEvent reservamos a palavra “iniciarprocesso”, e para o evento de endEvent usamos a palavra “finalizarprocesso”, essas quando levadas ao Prolog são identificadas como constantes. Essas são necessárias porque em algum momento precisaremos identificar se um processo possui esses dois símbolos que caracterizam o início e um fim de um processo. E se deixarmos os diversos nomes que forem colocados pelos analistas não teremos como verificar isso depois. As constantes aqui apresentadas são caso de parada para algumas regras que mais adiante serão apresentadas.

Algumas ações dos símbolos podem ser caracterizadas pelo seu tipo. Existem diferenças básicas para esta categorização. Por exemplo, todo evento de início só tem atributos de ***outgoing***, já o evento de fim só tem atributos de ***incoming,*** se o evento for intermediário ele possuirá tanto atributos de ***outgoing*** como atributos de ***incoming*** pois está localizado entre dois outros símbolos e precisa de um elemento de origem e um de destino.

Na tabela abaixo é possível observar que eventos de início (startEvents) só possui um ou mais ***outgoing,*** e eventos de fim (endEvents) só possui um ou mais ***incoming***.

```
1 <?xml version="1.0"?>
2 <definitions...
3 <task id="Id_f4d65854-cbb8-4886-88b9-847a50be32b3"
4 name="atividade 1">….
5....<incoming>Id_7b919a76-9146-4d21-9709-
6 63b0f7abc7fe</incoming>
7……<outgoing>Id_c7a7a5ac-7c9d-4150-87fe-
8 1013b18d7c81</outgoing>
9 </task>…..
10 <startEvent id="Id_7858ba1c-c440-4c03-8560-
11 00b0db828cbf" name="iniciar">
12…..<outgoing>Id_7b919a76-9146-4d21-9709-
13 63b0f7abc7fe</outgoing>
14 </startEvent>
15 <sequenceFlow id="Id_7b919a76-9146-4d21-9709-
16 63b0f7abc7fe" name="fluxo 1" sourceRef="Id_7858ba1c-
17 c440-4c03-8560-00b0db828cbf" targetRef="Id_f4d65854-
18 cbb8-4886-88b9-847a50be32b3">
19 </sequenceFlow>
20 <endEvent id="Id_ccbec2bb-69e3-4c01-850c-2f655cd201c1"
21 name="finalizar">
22.........<incoming>Id_c7a7a5ac-7c9d-4150-87fe-
23 1013b18d7c81</incoming>
24 </endEvent>
```

Na tabela acima é possível observar que o símbolo task (linhas 3 a 9) é um símbolo intermediário, pois possui pelo menos um atributo de ***incoming*** e pelo menos um atributo ***outgoing***.

No momento em que um analista de processo elabora um BPD, a atribuições de alguns nomes é optativa. Porém, para a nosso mapeamento BPMN para Prolog não é. Para isso criamos algumas constantes para preencher o espaço que seria reservado para esses nomes. A seguir temos:

- "rotulovazio" que é usado toda vez que o atributo *name* de um *sequence flow* estiver vazio.
- "atividadeorigeminexistente" que é usada toda vez existe um sequence flow possui o atributo "target", mas não possui o atributo "source". Em outras palavras toda vez que uma linha de sequência só possuir atividade de destino.
- "raiaorigeminexistente" utilizada toda vez que um sequence flow não possuir atividade de origem, se a atividade não existe ela não pode estar localizada em nenhuma raia.
- "atividadestinoinexistente" que é usada toda vez existe um sequence flow possui o atributo "souce", mas não possui o atributo "target". Em outras palavras, toda vez que uma linha de sequência só possuir atividade de origem.
- "raiadestinoinexistente" utilizada toda vez que um sequence flow não possuir atividade de destino, se a atividade não existe ela não pode estar localizada em nenhuma raia.
- e por fim as constantes "iniciarprocesso" e "finalizarprocesso" para identificar eventos de início e fim, respectivamente.

5.1 Tipos de Consultas

Existem muitas inconsistências que podem ser encontradas usando consultas em Prolog. Neste trabalho, estabelecemos um escopo para encontrar 7 inconsistências. Para encontrarmos essas inconsistências, necessitamos de algumas regras que apresentamos a seguir.

5.1.1 Consultas para descobrir se fluxos terminam ou seja se eles têm no mínimo um evento de fim

Para descobrirmos se existem fluxos que terminam, estabelecemos uma regra recursiva que percorre todo o fluxo desde o início ao fim e nessa busca verifica se no final do fluxo existe um endEvent, se existir então o fluxo tem fim caso contrário não. A palavra que identifica se existe um evento de fim é a constante "finalizarprocesso", observe a regra que estabelecemos abaixo.

Regra a qual estabelecemos:

```
evento_fim(finalizarprocesso,_,_).
evento_fim(A,B,L):- fluxo(A, B, C, D,_),
not(member(fluxo(A,B,C,D,_),
evento_fim(C, D,[fluxo(A,B,C,D,_)|L]).
```

Essa regra ao mesmo tempo em que verifica se um evento chegou ao fim ela trata fluxos que podem vim com ciclos. Isso evita que o programa entre em loop e permaneça executando o ciclo sem responder à consulta.

Como realizar a consulta?

Dado o seguinte código abaixo, apresentamos consultas realizadas no SWI – Prolog.

```
fluxo(iniciarprocesso, lane1,task1,lane1,[rotulovazio]).
fluxo(task1, lane1, task2,lane1,[rotulovazio, rotulovazio]).
fluxo(task1, lane1, task3,lane1,[rotulovazio, rotulovazio]).
fluxo(task2, lane1, finalizarprocesso,lane1,[rotulovazio]).
fluxo(task3, lane1, finalizarprocesso,lane1,[rotulovazio]).
fluxo(task5, lane1, finalizarprocesso,lane1,[rotulovazio]).
fluxo(task5, lane1, task6,lane1,[rotulovazio]).
fluxo(task6, lane1, task7, lane2,[rotulovazio]).
fluxo(task7, lane2, finalizarprocesso,lane2,[rotulovazio]).
fluxo(iniciarprocesso, lane1,task6,lane1,[rotulovazio]).

evento_fim(finalizarprocesso,_,_).
evento_fim(A,B,L):- fluxo(A, B, C, D,_),
      not(member(fluxo(A,B,C,D,_),L)),
      evento_fim(C, D,[fluxo(A,B,C,D,_)|L]).
```

A consulta é feita da seguinte maneira:

```
evento_fim(task6,lane1,[]).
true .
```

A consulta respondeu **true** porque a regra recursiva percorreu até encontrar um final finalizarprocesso que é o caso de parada. Para que a mesma consulta tenha resposta falsa basta retirar a linha: **fluxo(task7, lane2, finalizarprocesso, lane2, [rotulovazio]).** Que o resultado é negativo pois o programa não encontra o caso de parada, ou seja, o evento de fim representado pela atividade "finalizarprocesso".

```
evento_fim(task6,lane1,[]).
false.
```

5.1.2 Consultas para descobrir se o fluxo inicia com um StartEvent

Todo fluxo necessita ter um início, quando modelamos processos em BPMN devemos por convenção utilizar o símbolo chamado *startEvent* para indicar que o processo está iniciando naquele local.

De maneira similar a regra que estabelecemos na seção 5.1.1, definimos uma regra que busca os fatos do fim para o início e encontra se existe naquele determinado fluxo um evento de início.

```
evento_inicio(iniciarprocesso,_,_).
evento_inicio(A,B,L):- fluxo(C, D, A, B,_),
not(member(fluxo(C,D,A,B,_),L)),
evento_inicio(C, D,[fluxo(C,D,A,B,_)|L]).
```

Existe pouca diferença entre a regra da seção 5.1.1 e esta seção. Apenas o que muda é que a função recursiva que chamava o próximo fluxo agora chama o fluxo anterior.

A regra estabelecida tem um caso de parada chamado "**iniciarprocesso**". Quando o programa em Java encontra um startEvent, mesmo que esse símbolo tenha sido batizado com outro nome, na modelagem visual é colocado pelo programa a palavra reservada "**iniciarprocesso**".

Dado o código:

```
fluxo(iniciarprocesso, lane1,task1,lane1,[rotulovazio]).
fluxo(task1, lane1, task2,lane1,[rotulovazio, rotulovazio]).
fluxo(task1, lane1, task3,lane1,[rotulovazio, rotulovazio]).
fluxo(task2, lane1, finalizarprocesso,lane1,[rotulovazio]).
fluxo(task3, lane1, finalizarprocesso,lane1,[rotulovazio]).
fluxo(task5, lane1, finalizarprocesso,lane1,[rotulovazio]).
fluxo(task5, lane1, task6,lane1,[rotulovazio]).
```

```
fluxo(task6, lane1, task7, lane2,[rotulovazio]).
fluxo(task7, lane2, finalizarprocesso,lane2,[rotulovazio]).
fluxo(iniciarprocesso, lane1,task6,lane1,[rotulovazio]).

evento_fim(finalizarprocesso,_,_).
evento_fim(A,B,L):- fluxo(A, B, C, D,_),
      not(member(fluxo(A,B,C,D,_),L)),
      evento_fim(C, D,[fluxo(A,B,C,D,_)|L]).

evento_inicio(iniciarprocesso,_,_).
evento_inicio(A,B,L):- fluxo(C, D, A, B,_),
      not(member(fluxo(C,D,A,B,_),L)),
      evento_inicio(C, D,[fluxo(C,D,A,B,_)|L]).
```

A consulta é feita da seguinte maneira:

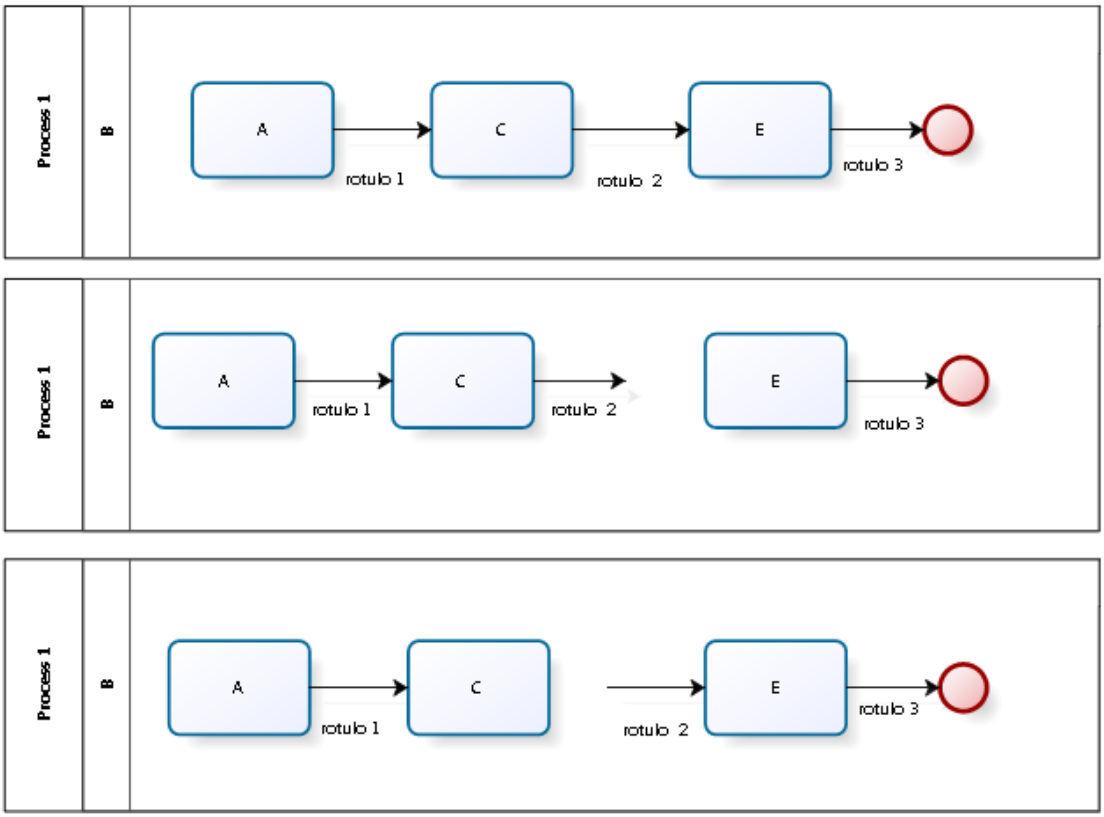
```
evento_fim(task5,lane1,[]).
true.
evento_inicio(task5,lane1,[]).
false.
```

Para todos as atividades podemos fazer consultas. Se, por exemplo, quisermos saber se o startEvent é um evento que tem um fluxo até um EndEvent, basta colocar a palavra reservada iniciarprocesso na consulta.

5.1.3 Consultas para descobrir se um fluxo está desconectado pela origem ou destino.

Uma linha de sequência deve ter sempre uma origem e um destino, o símbolo foi criado com esse propósito ligar outros dois símbolos, um sendo de origem e outro de destino. No entanto podem ocorrer casos em que um fluxo ficou levemente desconectado na hora da modelagem e trouxe assim uma quebra no fluxo do processo.

A Figura 23 ilustra como deveria ser o fluxo e quais as duas possibilidades poderiam ocorrer:



Fonte: Autora (2016).
Figura 23: fluxo quebrado.

Antes de apresentarmos como o tradutor gerou o código da figura 23, é valido informar que as letras das consultas são colocadas em minúsculas porque elas são constantes. Toda vez que o construtor automático recebe o programa em BPMN da modelagem, ele automaticamente transforma todas as letras maiúsculas do fluxo para letras minúsculas. Ou seja, aquele "A, C e E" da figura 23 serão transformados em: "a,c" e "e", quando passarem para código em Prolog.

Como o tradutor gera o fluxo quebrado:

Fluxo bem formado.

`fluxo(a, b, c, b,[rotulo1]).`
`fluxo(c, b, e, b,[rotulo2]).`
`fluxo(e, b, finalizarprocesso,b,[rotulo3]).`

Fluxo quebrado sem destino.

`fluxo(a, b, c,b,[rotulo1]).`
`fluxo(c, b,`
`atividadedestinoinexistente,raiadestinoinexistente,[rotulo2]).`
`fluxo(e, b, finalizarprocesso,b,[rotulo3]).`

Fluxo quebrado sem origem.

```
fluxo(a, b, c,b,[rotulo1]).
fluxo(atividadeorigeminexistente,raiaorigeminexistente,
e,b,[rotulo2]).
fluxo(e, b, finalizarprocesso,b,[rotulo3]).
```

A partir desse código definimos uma regra em Prolog para identificar o fluxo quebrado. Regra estabelecida:

```
fluxoQuebrado(A,B):-
fluxo(A,B,atividadedestinoinexistente,raiadestinoinexistente,_).

fluxoQuebrado(A,B):-
fluxo(atividadeorigeminexistente,raiaorigeminexistente,A,B,_).
```

Essa regra diz que quando um fluxo está quebrado ele pode vim de duas situações específicas. A primeira delas quando o fluxo apresenta uma origem, mas um destino inexistente, observe o segundo processo da Figura 23; a segunda situação ocorre quando o fluxo apresenta um destino, mas uma origem inexistente, observe o terceiro caso da Figura 23.

Para consultarmos se um fluxo está quebrado em determinada atividade, basta consultar se o fluxo está quebrado na atividade 1 da raia 1, e o interpretador responderá com "**true**" ou "**false**".

Observe a consulta abaixo:

fluxoQuebrado(atividade 1, raia 1).

As possíveis respostas são **true** ou **false**.

5.1.4 Consultas para descobrir se existe fluxo perdido sem origem nem destino.

Assim como existem várias inconsistências pode existir também um fluxo perdido. Um fluxo dito "perdido" ou "solto" é quando esse fluxo está presente em um diagrama, mas não está saindo de nenhum símbolo nem chegando em outro. Esse conector pode ter sido colocado na modelagem por engano, ou o analista de processos esqueceu de conectá-lo aos símbolos.

A consulta é a das mais simples que existe, a regra verifica se o conector tem origem e destinos inexistente dado um determinado rótulo, que é o nome do conector.

Para fluxos perdidos estabelecemos a seguinte regra:

```
fluxoPerdido(ROTULO):-
                          fluxo(atividadeorigeminexistente,
                          raiaorigeminexistente,
                          atividadedestinoinexistente,
                          raiadestinoinexistente, ROTULO).
```

Para consultar se uma linha de sequência (conector) está solta, ou seja, não conectada a nenhum símbolo apenas colocamos seu nome na consulta, Observe a seguir:

fluxoPerdido(linha1).

true.

A resposta à consulta pode ser “true” ou “false”, caso seja a verdadeiro então significa que o conector está desconectado, caso contrário o fluxo está conectado pelo menos de um lado.

fluxoPerdido(linha 2).

false.

5.1.5 Consultas para descobrir se existe Task e subProcessos perdidos sem entrada e saída

Assim como conectores podem estar desconectados, pode ocorrer o mesmo com outros símbolos. Nesta seção, apresentamos apenas dois dos símbolos. Os que não forem aqui apresentados, possivelmente poderão ser encontrados na seção trabalhos futuros.

Atividades e sub-processos estão na modelagem representando ações feitas por pessoas ou sistemas. Essas não são difíceis de identificar quando estão desconectadas do fluxo principal. Estabelecemos regras para identificar se uma tarefa está conectada ou não ao fluxo principal:

Regra:

```
tarefaEstaConectada(T):-tarefaOrigem(T).
tarefaEstaConectada(T):-tarefaDestino(T).
tarefaOrigem(T):- fluxo(T,_,_,_,_).
tarefaDestino(T):- fluxo(_,_,T,_,_).
```

Se a tarefa estiver no fluxo ela só poderá aparecer em dois lugares, ou no primeiro parâmetro, como a origem de um fluxo; ou no terceiro parâmetro como destino de um fluxo.

A regra foi elaborada seguindo a lógica do local, se a atividade estiver em origem está conectada, se ela estiver em destino também está conectada. As demais foram definidas como variável anônima "_", para que não ocorra o *Warning de Single Variáveis*.

Consultas podem ser feitas da seguinte maneira:

? - tarefaEstaConectada(atividade1).

true.

? - tarefaEstaConectada(atividade1).

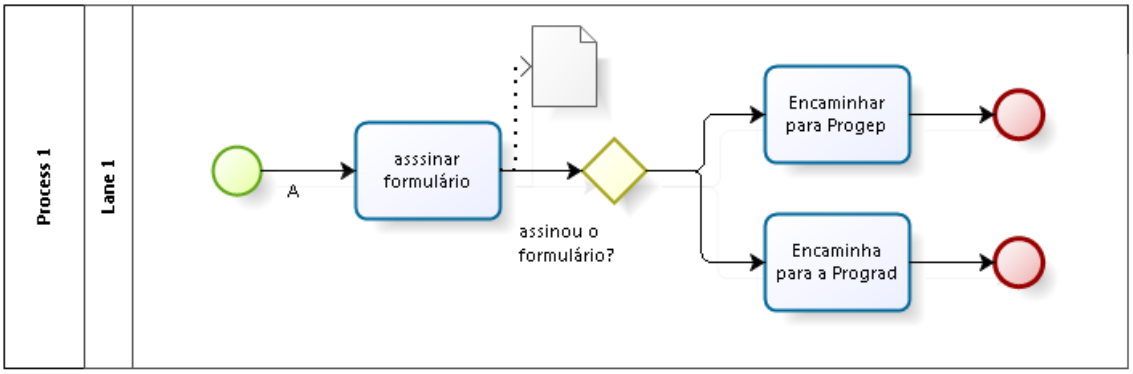
false.

Se a resposta for "true" significa que a atividade está conectada ou pelo lado direito do fluxo, ou pelo lado esquerdo, ou pelos dois lados no melhor caso.

Se a resposta for "false" significa que a atividade não está conectada a nenhum fluxo, ou seja, está solta ou desconectada do fluxo, sem nenhuma utilizada.

5.1.6 Consultas para descobrir se existe gateway sem rótulos.

Muitas vezes no momento da modelagem analistas podem esquecer de batizar as saídas ou as entradas de gateways e esses acabam ficando sem rótulos. Os rótulos no caso dos gateways não são opcionais são obrigatórios, porque a falta deles impossibilita a leitura da divergência. Observe o exemplo abaixo, saídas de gateways sem rótulos.



Fonte: Autora (2016).
Figura 24: Saída de gateways sem nomes de rótulos.

Deixar de colocar os rótulos no caso dos gateways é tão perigoso como repeti-los em fluxos divergentes.

Quando construímos o tradutor automático em Java tratamos os casos em que conectores tem rótulos vazios, para esses casos colocamos o nome reservado "rotulovazio" apenas para identificar os rótulos sem nomes. Com isso também evitamos que quando o código em Prolog venha para o interpretado evite erros de locais vazios.

Regra em Prolog para descobrir se existem rótulos vazios:

```
rotuloVazioEmGateway(A,B):- gateway(A,B),
                            fluxo(A,B,_,_,L),
                            member(rotulovazio,L).
```

Primeiro temos que conferir se existe um gateway em (A, B); se existir, precisamos chamar o fluxo a qual esse gateway pertence para obtermos acesso a lista de rótulos. Quando temos a lista de rótulos em mãos apenas conferimos se a palavra "rotulovazio" é membro dessa lista, se for é porque existe rótulo vazio nos fluxos.

5.1.7 Consultas para descobrir se existem ciclos no programa

Os ciclos, dependendo do caso, podem conferir um problema, por exemplo, quando decidimos percorrer o fluxo principal e existe um ciclo neste fluxo que é executado de forma automática, se a regra elaborada não tratar esse ciclo, a própria consulta entra em loop e o programa fica executando eternamente. Isso ocorre porque não existiu um caso de parada para esse ciclo.

Abaixo elaboramos como estabelecemos uma regra para identificar ciclos:

```
ciclo(A,B,L):- fluxo(A,B,C,D,E), member(fluxo(A,B,C,D,E),L).
ciclo(A,B,L):- fluxo(A,B,C,D,E),
            not(member(fluxo(A,B,C,D,E),L)),
            ciclo(C,D,[fluxo(A,B,C,D,E)|L]).
```

Nessa regra percorremos o fluxo principal e colocamos em uma lista os fatos que o compõem, chamamos essa regra de ciclos. Quando o interpretador tenta colocar um fluxo que já é membro da lista então ele cai no caso de parada e devolve a resposta que existe ciclo naquele ponto.

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Neste trabalho, construímos um programa Java que faz mapeamento automático de modelos de processos em BPMN para Prolog. Para a partir do código Prolog gerado analisarmos o BPD e identificarmos inconsistências através de consultas e regras em Prolog.

Apresentamos 4 tipos de problemas típicos em modelagem de processos em BPMN e um escopo de 7 inconsistências. Apresentamos como identificar essas inconsistências em Prolog com consultas e regras, e informamos a interpretação dessas consultas através de alguns exemplos genéricos. No apêndice A, modelamos um processo completo em BPMN e apresentamos seu código em Prolog gerado usando o construtor automático que elaboramos.

Como trabalhos futuros sugerimos:

1- Representar em Prolog os artefatos: formulários, grupos e anotações e tratar isso no construtor automático.
2- Diferenciar no mapeamento em Prolog cada tipo de símbolo: diferenciar por exemplo, o evento inicial de timer para o evento inicial de message e assim por diante com todos os tipos sejam eventos, gateways ou atividades, pois neste trabalho representamos apenas os símbolos no seu tipo genérico.
3- Representar no Prolog o evento intermediário e tratar isso no construtor automático.
4- Representar no Prolog os outros conetores: associação e fluxo de mensagens e acrescentar no construtor automático.
5- Modificar o construtor automático para tratar casos que símbolos foram colocados sem raia, ou que foram posicionados em cima da linha da raia ou da piscina.
6- Verificar o mau uso de símbolos, por exemplo o mau uso do gateway quando ele possui apenas uma origem e um destino, verificar se em cada situação estão usando os conectores corretos.
7- Verificar repetição de nomes em diferentes sequences flows e a repetição de nomes em diferentes tasks.
8- Tratar a composição de sub-processos, pois os mesmos estão representados apenas como atividades externas. Por exemplo, nos casos em que os sub-

processos são detalhados em outro arquivo a representação deste outro arquivo não está elaborada neste trabalho e não está tratada no construtor automático.

REFERÊNCIAS

ABE, Jair Minoro; SCALZITTE, Alexandre; SILVA FILHO, João Inácio da. Introdução à lógica para a ciência da computação. São Paulo: Arte e Ciência, 2002. 247p. ISBN 8574730459 (broch.).

ALMEIDA, Maurício Barcellos. **Uma Introdução ao XML, sua utilização na Internet e alguns conceitos complementares.** 2002. 13f. Artigo. Universidade Federal de Minas Gerais. Disponível em: <www.scielo.br/pdf/ci/v31n2/12903> Acesso em 19 de fev. de 2016.

ANDROCEC, Darko. Simulating BPMN models with Prolog. In: **CECIIS-2010**. 2010. 6 f. Disponível em: <http://ceciis.foi.hr/app/index.php/ceciis/2010/paper/view/288>. Acesso em: 24 jun. 2015.

AMBROSIO, Ana Paula. L; MORAIS Edson Andrade Martins. **Ontologias: conceitos, usos, tipos, metodologias, ferramentas e linguagens.** Technical Report. 22 p. UNIVERSIDADE FEDERAL DO GOIÁS. 2007. Disponível em: < http://www.portal.inf.ufg.br/sites/default/files/uploads/relatorios-tecnicos/RT-INF_001-07.pdf> Acesso em: 23 abr. 2015.

ARANTES, Rhaíssa Nogueira. **Introdução ao Business Process Modeling Notation (BPMN).** 08 de Jan. de 2014. Disponível em: <http://www.devmedia.com.br/introducao-ao-business-process-modeling-notation-bpmn/29892>.Acesso em: 20 de maio de 2105.

SOBRAL, V.S. **Gerenciamento de Processos de Negócio.** São Paulo: Erica, 2007.

BARBOSA, Alexandre de Andrade. CUNHA, Joseluce de Farias. **Apostila: introdução à programação em lógica.** 2006. 34 f. Disponível em: <http://www.dsc.ufcg.edu.br/~logica/PROLOG/apostila-Prolog.pdf>. Acesso em: 07 jun. 2015

BRASIL. Lei nº 8.112 de 11 de dezembro de 1990. **Lex:** Do Afastamento para participação em Programa de Pós-graduação Stricto Sensu no País, Brasília, DF,18 de abril de 1991. Seção 4. Disponível em: <http://www.planalto.gov.br/ccivil_03/leis/l8112cons.htm> Acesso em: 08 de jun.2015.

BPMI/OMG. **Business Process and Notation (BPMN) Information.** 2006. Disponível em: < http://www.bpmn.org/>. Acesso em: 25 jun. 2015.

BPMN: **Guia de Referência ESP**. [200-?]. Disponível em: <https://www.bizagi.com/docs/BPMN_Guia_de_Referencia_ESP.pdf>. Acesso em: 20 maio de 2015.

CAMARA, Marco Sérgio Leal. **Inteligência Artificial: Representação do Conhecimento.** 08 p. Departamento de Engenharia Informática da FCTUC. [entre 2000 e 2001] Disponível em:<http://www.academia.edu/3261902/Intelig%C3%AAncia_Artificial_Representa%C3%A7%C3%A3o_de_conhecimento> Acesso em: 23 abr. 2015.

COSTA, Hudson. **Programação Lógica**. 25 de maio de 2011. Universidade Estadual Vale Do Acaraú – Uva. Centro de Ciências Exatas e Tecnologia – CCET. 11 p. Disponível em: <https://hudsoncosta.files.wordpress.com/2011/05/programacaoemlogica.pdf>. Acesso em: 29 jun. 2015.

FERREIRA, Mardson da Silva. **Avaliação de soluções bpm para implantação de gestão de processos na Universidade Federal do Ceará Campus Quixadá.** 2013. 64 p. Monografia (Graduação em Sistemas de Informação) Universidade Federal do Ceará, Quixadá,2013. Disponível em: < http://www.repositoriobib.ufc.br/000012/00001269.pdf > Acesso em: 23 abr. 2015.

FRANCO, Cynthia Raphaella da Rocha. **Um catálogo de boas práticas, erros sintáticos e semânticos em modelos bpmn.** 2014. 69 p. Monografia (Graduação em Ciência da Computação) Universidade Federal de Pernambuco, Recife, 2014. Disponível em: <http://www.cin.ufpe.br/~tg/2013-2/crrf.pdf> Acesso em: 23 jan. 2016.

GERSTING, Judith L. **Fundamentos matemático para a ciência da computação: um tratamento moderno de matemática discreta**. 5ª ed. Rio de Janeiro: Livros Técnicos e Científicos, c2004. xiv, 597 p. ISBN 8521614225 (broch.).

GIANNINI, Fabio Clark. **50 Motivos para implantar um BPM**. 2013. Disponível em: <http://blog.selens.com.br/blog/index.php/2013/03/10/50-motivos-para-implantar-um-bpm/> Acesso em: 18 nov. 2014.

GUIMARÃES, Francisco José Zamith. **Utilização de ontologias no domínio B2C.** Rio de Janeiro, novembro de 2002. p. 49 – 72. Dissertação de Mestrado. PUC – Rio, 2002.

HAMMER, Michael, CHAMPY, James. **Reengineering the corporation**. New York: HarperBusiness, 1994.

HEREDIA, Leonardo Rodriguez. **Transformação de modelos de processos de negócio em bpmn para modelos de sistema utilizando casos de uso da uml.** Jan. de 2012. Porto Alegre. 126 f. Dissertação de mestrado. Fac. de Informática, PUCRS. Disponível em: <http://tede.pucrs.br/tde_busca/arquivo.php?codArquivo=4157> Acesso em: 26 de maio de 2015.

IEL EDUCAÇÃO. **Modelagem de Processos com BPMN**, 2016. Disponível em:<http://www.ielgo.com.br/iel/site/Cursos.do?idCurso=174&vo.codigo=184>. Acesso em: 17 de fev. de 2016.

INSTITUTO SERZEDELLO CORRÊA. **Curso de mapeamento de processos de trabalho com BPMN e BIZAGI**. [Brasília]: Tibunal de Contas da União, 2013. Disponível em: <http://portal.tcu.gov.br/comunidades/gestao-de-processos-de-trabalho/curso-de-mapeamento-de-processos-de-trabalho/curso-de-mapeamento-de-processos-de-trabalho.htm>. Acesso em: 21 maio 2015. 107 f. 4 aulas em pdf.

IPROCESS. Guia BPMN 2.0. 2014. 01 f. Disponível em: < http://www.iprocesseducation.com.br/guia_bpmn/iProcess%20-%20Guia%20de%20referencia%20BPMN2_V2.pdf >. Acesso em: 29 jun. 2015.

LAGO, Silvio. **Introdução à linguagem Prolog.** [200-?]. Disponível em: <http://www.ime.usp.br/~slago/slago-Prolog.pdf>. Acesso em: 03 de maio de 2015.

LIGEZA, A. KLUZA, K. NALEPA, G. J. AND POTEMPA, T. **AI Approach to Formal Analysis of BPMN Models**. **Towards a Logical Model for BPMN Diagrams.** In Proc. of Federated Conference on Computer Science and Information Systems (FedCSIS'12), pages 931–934, 2012.

LIMA, Edirlei Soares de. **INF 1771 – Inteligência Artificial**. 2012. Slides. Pontifícia Universidade Católica do Rio de Janeiro. Disponível em : <http://edirlei.3dgb.com.br/aulas/ia_2012_2/IA_Aula_09_Prolog_2012.pdf>. Acesso em: 18 de fev. de 2016.

MARQUES, Alexander Correia. SILVA, Ana Catarina Lima. **Análise Comparativa Entre Ferramentas de BPMS (Business Process Management Suite) para Organizações de Médio Porte.** 2011. 14f. Artigo (conclusão do curso de especialização *Lato Sensu* Gestão Estratégica de Processos de Negócio) - Pontifícia Universidade Católica de Minas Gerais, Instituto de Educação Continuada, Belo Horizonte. Disponível em: <http://pt.slideshare.net/alexandercorreiamarques/anlise-comparativa-entre-ferramentas-de-bpms-business-process-management-suite-para-organizaes-de-mdio-porte>. Acesso em: 18 de maio de 2015.

OBJECT MANAGEMENT GROUP. **Business Process Model and Notation (BPMN).** 2011. Disponível em: <http://www.omg.org/spec/BPMN/2.0/PDF/> Acesso em: 26 out.2014.

PAIN, Rafael. CARDOSO, Vinícius. CAULLIRAUX, Heitor. CLEMENTE, Rafael. **Gestão por processos. Pensar Agir e Aprender.** 1ª ed. Porto Alegre: Bookman, 2009. 328 p.

PALAZZO, Luiz A. M. **Introdução a Programação: Prolog.** Pelotas: editora da Universidade Católica de Pelotas, 1997. 198 f. Disponível em: <http://200.17.141.213/~gutanunes/hp/IA/Prolog_palazzo.pdf>. Acesso em 10 de jun. de 2015.

PIRES, Mônica Luíza Alves Venâncio. **Maratona CBOK.** [S.l],[200-?] Slides. Disponível em : <http://www.bpmglobaltrends.com.br/wp-content/uploads/2014/01/Maratona-CBOK-Cap-2-Gerenciamento-de-processos-de-Neg%C3%B3cio-M%C3%B4nica-Luzia-Alves-Venancio-Pires-CBPP.pdf> Acesso em: 26 de maio de 2015.

PIZZA, William Roque. **A metodologia Business Process Management (BPM) e sua importância para as organizações.** 2012. São Paulo. 28 f. Monografia. Faculdade de Tecnologia de São Paulo-FATEC. Disponível em: <http://www.fatecsp.br/dti/tcc/tcc00074>. Acesso em: 26 de maio de 2015.

OSEIAS, In: BÍBLIA. Português. **Bíblia sagrada:** nova versão internacional. São Paulo: Geográfica, 9ª edição, 2001.

RAMOS, Carlos. Instituto Superior de Engenharia do Porto**. Origem do Prolog.** [Porto-Portugal] [entre 2006 e 2007] Slides. Disponível em:< http://www.dei.isep.ipp.pt/~jtavares/ALGAV/downloads/ALGAV_TP_aula2.pdf> Acesso em: 23 de abr. 2015.

RAMOS, Thaís. **Conceitos básicos de Informática.** 2011. Disponível em: <http://novosresultados.com/conceitos-basicos-de-informatica/>. Acesso em: 25 jun. 2015.

PRATI, Ronaldo C. Universidade Federal do ABC. **Representação do Conhecimento Considerações Gerais.** [São Paulo], [200-?] Slides. Disponível em: <http://professor.ufabc.edu.br/~ronaldo.prati/InteligenciaArtificial/RC.pdf>. Acesso em: 23 abr.2015.

SEBESTA, Robert W. **Conceitos de Linguagens de Programação**. 5ª ed. Porto Alegre: Bookman, 2003. ix, 638p. ISBN 8536301716(broch.).

SMITH, Fabrizio; PROIETTI, Maurizio. **Ontology-based Representation and Reasoning on Process Models: A Logic Programming Approach.** Roma (Italy) p. 1-42, 07 Oct 2014. Disponível em: <http://arxiv.org/pdf/1410.1776v1.pdf>. Acesso em 10 de jun. de 2015.

TODESCO, J. L. RAUTENBERG, S. SETEIL, A. V, GAUITHEIR, F. A.O, **Uma metodologia para o desenvolvimento de ontologia.** Universidade Federal de Santa Catarina – UFSC. 26 pág. Revista ciências exatas e naturais Vol. 10 Nº 2 Jul/ Dez 2008.

TANENBAUM, Andrew S**. Sistemas Operacionais modernos.** 3ª ed. São Paulo: Pearson Prentice Hall, 2009. 625 p.

UNIVERSIDADE FEDERAL DO CEARÁ. **Manual de Administração de Pessoal.** Fortaleza, 2014. Disponível em: <http://www.prpl.ufc.br/images/arquivos/manuais_processos/prpl_pradm_progep_ufcinfra/manual_administracao_de_pessoal.pdf>. Acesso em: 08 de jun. 2015.

VALE, R. OLIVEIRA, S.B.de. (org). Análise e Modelagem de Processos de Negócio. São Paulo: Atlas, 2013. 207 p.

VAN DER AALST, Wil M.P; TER HOFSTEDE, Arthur H. M; WESKE, Mathias. **Business process management: A survey.** Em Proceedings of the International Conference on Business Process Management, volume 2678 de Lecture Notes in Computer Science, páginas 1019–1019, Eindhoven, Holanda, junho 2003. Springer. 1

APÊNDICE A – Caso de estudo: Processo de afastamento de Professores

Este apêndice contém o primeiro caso estudado para elaborarmos o primeiro mapeamento de BPMN para Prolog. Apresentamos primeiro a descrição do processo em linguagem natural, depois apresentamos em BPMN e por fim a representação em Prolog que foi gerada pelo aplicativo Java Desktop que elaboramos.

Descrição do Processo em Linguagem Natural

Nesta seção explicaremos o passo a passo de como ocorre o processo de Afastamento de servidores, requisitos e papéis, como não pretendemos nos aprofundar no assunto para mais informações indicamos o art. 96 da lei nº 8.112 de 11 de dezembro de 1990 e a seção 5.6.4 de UFC (2014).

O processo de afastamento de servidor docente constitui-se em um trâmite com o intuito de autorizar o afastamento do servidor docente de suas atividades na universidade. Um servidor se ausenta da Universidade para estudo ou missão (cursos de pós-graduação, congresso, conferência, seminário) oficial no país ou no exterior, obedecendo as regras da lei nº 8.112.

Existem três tipos de afastamento: 1º Afastamento dentro do país até 60 dias; 2º Afastamento dentro do país além de 60 dias e 3º Afastamento no exterior. Para o 1º afastamento acima citado existe um fluxo, ou seja, ocorre de uma maneira. Para o 2º e 3º afastamento ocorre de maneira similar ao primeiro, mas com algumas diferenças.

Descrição do Afastamento tipo 1:

1. O processo tem início com a solicitação do professor, este preenche um formulário, informando seus dados funcionais (siape, nome e cargo/função). Após no mesmo formulário o requerente solicita oficialmente a autorização do diretor, preenchendo os dados da solicitação (com ônus, com ônus limitado ou sem ônus, período de afastamento, objetivo do afastamento e local (instituição) de destino).

2. Logo em seguida a secretaria do campus confere o formulário, estando este incompleto volta para o professor que revisa, estando completo este é encaminhado pela secretaria à direção para análise.

3. Ao chegar na direção o diretor analisa aprovando ou negando a solicitação.

4. Em caso de processo negado é enviada uma mensagem ao docente informando a decisão do diretor. Caso seja aprovado o formulário é encaminhado para a secretaria.

5. Chegando novamente a secretaria são elaborados a portaria e o ofício.

6. Da secretaria volta para o diretor assinar.

7. Após assinado o formulário volta a secretaria de onde é aberto o processo de solicitação de registro no assentamento funcional do servidor interessado. Após aberto processo este é encaminhado a progep.

8. Chegando a progep é feito o registro e a solicitação é finalizada.

Descrição do Afastamento tipo 2 e 3:

1. O processo tem início com a solicitação do professor, este preenche um formulário, informando seus dados funcionais (siape, nome, sigla unidade, sigla unidade, subunidade e cargo/função). Após no mesmo formulário o requerente solicita oficialmente a autorização do reitor, preenchendo os dados da solicitação (com ônus, com ônus limitado ou sem ônus, período de afastamento, objetivo do afastamento e local (instituição) de destino). Quando não se trata de congressos seminários, congressos ou conferências o docente deve informar dados gerais da instituição para onde pretende ir. A chefia (diretor) informa a necessidade ou não de substituto. Além disso o requerente deve anexar alguns documentos que não serão especificados aqui, olhar Anexo 1.

2. Logo em seguida a secretaria do campus confere o formulário, estando este incompleto volta para o professor que revisa, estando completo este é encaminhado para análise do diretor.

3. Após análise do diretor o formulário volta para a secretaria que abre processo.

4. Aberto o processo o mesmo é encaminhado ao conselho de campus. Ao chegar no conselho este aprecia a solicitação negando ou aprovando. Caso seja negado o conselho finaliza o processo e uma mensagem é encaminhada ao docente informando a decisão. Em caso de aprovação poderá ocorrer dois fluxos diferentes, que são:

4. A – Aprovado com diligências.

Nesse caso o formulário volta ao requerente para sanar pendências.

4.1 - Após sanadas pendências o processo é encaminhado à secretaria que emite ofício para o diretor assinar.

4.2 - Depois de assinado o processo volta a secretaria de onde é encaminhado para a progep.

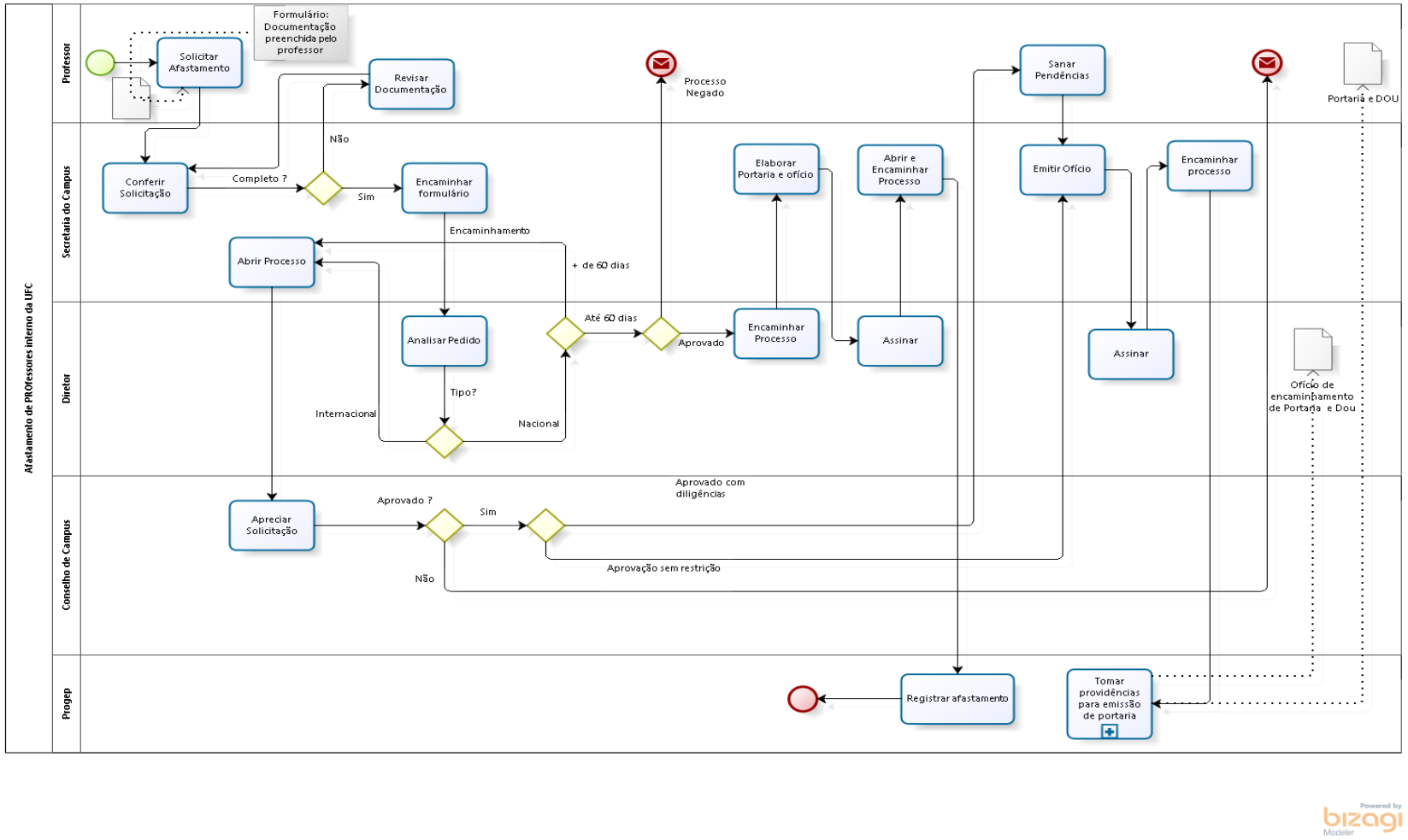
4.3 – Chegando a progep são tomadas providências para emissão de portaria.

4.4 – Após tomada de providências é finalizado o processo.

4. B – Aprovado sem restrições.

O único diferencial entre o 4.A e o 4.B é que neste último o processo não volta para o requerente sanar pendências o mesmo é enviado direto ao para a secretaria emitir ofício, e após esse passo os próximos são iguais ao do anterior.

Modelagem do Processo em BPMN



Fonte: Autora (2015).

Figura 25: Processo de Afastamento de Professores em BPMN.

Representação do Processo em Programação Lógica.

A Representação abaixo foi elaborada automaticamente pelo programa em JAVA que elaboramos. Enviamos como entrada o código em XML do Processo em BPMN, e a saída foi o conjunto de fatos em Prolog, abaixo:

Código em Prolog referente ao processo de Afastamento de Professores

```
fluxo(iniciarprocesso, professor,solicitarafastamento,professor,[rotulovazio]).
fluxo(solicitarafastamento, professor,
conferirsolicitação,secretariadocampus,[rotulovazio]).
fluxo(conferirsolicitação, secretariadocampus,
encaminharformulário,secretariadocampus,[completo, sim]).
fluxo(conferirsolicitação, secretariadocampus, revisardocumentação,professor,[completo,
não]).
fluxo(revisardocumentação, professor,
conferirsolicitação,secretariadocampus,[rotulovazio]).
fluxo(encaminharformulário, secretariadocampus,
analisarpedido,diretor,[encaminhamento]).
fluxo(analisarpedido, diretor, abrirprocesso,secretariadocampus,[tipo, internacional]).
fluxo(analisarpedido, diretor, abrirprocesso,secretariadocampus,[tipo, nacional,
+de60dias]).
fluxo(analisarpedido, diretor, encaminharprocesso,diretor,[tipo, nacional, até60dias,
aprovado]).
fluxo(analisarpedido, diretor, finalizarprocesso,professor,[tipo, nacional, até60dias,
rotulovazio]).
fluxo(encaminharprocesso, diretor,
elaborarportariaeofício,secretariadocampus,[rotulovazio]).
fluxo(abrireencaminharprocesso, secretariadocampus,
registrarafastamento,progep,[rotulovazio]).
fluxo(registrarafastamento, progep, finalizarprocesso,progep,[rotulovazio]).
fluxo(apreciarsolicitação, conselhodecampus, finalizarprocesso,professor,[aprovado,
não]).
fluxo(apreciarsolicitação, conselhodecampus, sanarpendências,professor,[aprovado, sim,
aprovadocomdiligências]).
fluxo(apreciarsolicitação, conselhodecampus, emitirofício,secretariadocampus,[aprovado,
sim, aprovaçãosemrestrição]).
fluxo(sanarpendências, professor, emitirofício,secretariadocampus,[rotulovazio]).
fluxo(emitirofício, secretariadocampus, assinar,diretor,[rotulovazio]).
fluxo(assinar, diretor, encaminharprocesso,secretariadocampus,[rotulovazio]).
fluxo(encaminharprocesso, secretariadocampus,
tomarprovidênciasparaemissãodeportaria,progep,[rotulovazio]).
fluxo(elaborarportariaeofício, secretariadocampus, assinar,diretor,[rotulovazio]).
fluxo(assinar, diretor, abrireencaminharprocesso,secretariadocampus,[rotulovazio]).
fluxo(abrirprocesso, secretariadocampus,
apreciarsolicitação,conselhodecampus,[rotulovazio]).

% 1ª Regra
evento_fim(finalizarprocesso,_,_).
evento_fim(A,B,L):- fluxo(A, B, C, D,_),
not(member(fluxo(A,B,C,D,_),L)),
evento_fim(C, D,[fluxo(A,B,C,D,_)|L]).

% 2ª Regra
evento_inicio(iniciarprocesso,_,_).
```

```
evento_inicio(A,B,L):- fluxo(C, D, A, B,_),
      not(member(fluxo(C,D,A,B,_),L)),
      evento_inicio(C, D,[fluxo(C,D,A,B,_)|L]).

% 3ª Regra Vefificar se existe gateway em determinado Ponto
gateway(O,Y):-fluxo(O,Y,A,_,_),fluxo(O,Y,C,_,_),A\=C.
gateway(O,Y):-fluxo(A,_,O,Y,_),fluxo(C,_,O,Y,_),A\=C.

rotuloVazio(A,B):- gateway(A,B),fluxo(A,B,_,_,L),member(rotulovazio,L).

% 4ª Regra
fluxoQuebrado(A,B):- fluxo(A,B,destinoinexistente,destinoinexistente,_).
fluxoQuebrado(A,B):-fluxo(origeminexistente,origeminexistente,A,B,_).

% 5ª Regra
fluxoPerdido(ROTULO):-
fluxo(origeminexistente,origeminexistente,destinoinexistente,destinoinexistente,ROTULO).

tarefaRaia(T,R):-tarefaRaiaOrigem(T,R).
tarefaRaia(T,R):-tarefaRaiaDestino(T,R).
tarefaRaiaOrigem(T,R):- fluxo(T,R,_,_,_).
tarefaRaiaDestino(T,R):- fluxo(_,_,T,R,_).

% 6ª Regra
tarefaEstaConectada(T):-tarefaOrigem(T).
tarefaEstaConectada(T):-tarefaDestino(T).

tarefaOrigem(T):- fluxo(T,_,_,_,_).
tarefaDestino(T):- fluxo(_,_,T,_,_).

% 7ª Regra
ciclo(A,B,L):- fluxo(A,B,C,D,E), member(fluxo(A,B,C,D,E),L).
ciclo(A,B,L):- fluxo(A,B,C,D,E),
not(member(fluxo(A,B,C,D,E),L)),
ciclo(C,D,[fluxo(A,B,C,D,E)|L]).
```