Uma abordagem para geração de dados para Teste Funcional Automatizado de aplicações baseadas em Gerenciamento de Processos de Negócio apoiada na notação BPMN

Trilha de Trabalhos Técnicos

Jéssica Lasch de Moura[1], **Andrea Schwertner Charão**[1]

[1] Laboratório de Sistemas de Computação (LSC)
Universidade Federal de Santa Maria

{jmoura,andrea}@inf.ufsm.br

Abstract. *This paper describes an approach designed to obtain data for the implementation of automated functional testing of applications based on Business Process Management using BPMN notation. The objective of this study was to extract information, through the analysis of BPMN files, for testing processes, in order that this information could assist in the creation and execution of tests. With this approach, it was possible to obtain data that have produced major elements for the test steps, helping to reduce the analysis time and create more complete testing.*

Resumo. *Este trabalho descreve uma abordagem criada para obter dados para a execução de testes funcionais automatizados de aplicações baseadas em Gestão de Processos de Negócio utilizando a notação BPMN. O objetivo deste trabalho era extrair informações, através da análise de arquivos BPMN, para o teste dos processos, com o propósito de que estas informações pudessem auxiliar na criação e execução de testes. Com esta abordagem, foi possível obter dados que permitiram gerar elementos importantes para as etapas de teste, ajudando a diminuir o tempo de análise e a criar testes mais completos.*

1. Introdução

A gestão de processos de negócio (*Business Process Management* – BPM) tem suscitado o interesse de empresas e da comunidade científica, tanto por seus benefícios quanto por seus desafios. Designa-se por BPM o conjunto de conceitos, métodos e técnicas para suportar a análise, modelagem, execução, monitoramento e otimização dos processos de negócio [Weske 2012]. *BPMN* ou *Business Process Management and Notation* é uma notação que fornece a capacidade de compreender processos de negócio em uma notação gráfica e a capacidade de exportar esses processos de uma forma padrão [BPMN 2016].

Algumas tarefas como verificação e testes ainda são consideradas um desafio na área de BPM [van der Aalst 2013]. De fato, o teste automatizado de aplicações BPM é pouco abordado, tanto pela comunidade da área de BPM [Weske 2012] como da área de testes de software [Graham and Fewster 2012]. Devido a isso, em um trabalho anterior [de Moura and Charão 2015] foram realizados testes de carga e teste funcional em uma aplicação BPM utilizando soluções já consagradas no teste de aplicações Web. Nesse

trabalho anterior, o teste funcional utilizando as ferramentas Selenium e Cucumber se mostrou mais promissor.

No trabalho anterior, também percebeu-se que a análise para obter os dados necessários para o teste funcional automatizado completo de uma aplicação pode ser complexa. Então, com a intenção de extrair dados importantes para o teste dos processos, decidiu-se por analisar os arquivos BPMN.

Optou-se por gerar uma tabela contendo os fluxos possíveis de cada processo, permitindo uma visão completa de como o processo é executado. Então foi criada uma abordagem para tratar os arquivos BPMN, gerar a tabela e, possívelmente, utilizar esses dados de forma a criar mais elementos importantes para o teste.

Assim, o objetivo deste trabalho foi extrair informações, através da análise de arquivos BPMN, para o teste dos processos, com o propósito de que estas informações pudessem auxiliar na criação e execução de testes.

2. Business Process Management - *BPM*

O termo BPM pode ser usado com ênfases diferentes, às vezes com foco em tecnologia (software) e outras vezes em gestão. Mesmo assim, a área tem convergido no entendimento do ciclo de vida de aplicações de BPM, que envolve as atividades de análise, modelagem, execução, monitoramento e otimização [AB 2009].

Os sistemas de BPM (BPMS) têm se afirmado como ferramentas essenciais para suporte a atividades desse ciclo de vida. Atualmente, pode-se dizer que um típico BPMS oferece recursos para definição e modelagem de processos em BPMN, controle da execução e monitoramento de atividades dos processos [Forrester Research 2013]. Há uma tendência dos BPMS em abreviar o desenvolvimento de software, por exemplo através de geradores de formulários Web associados a tarefas dos processos [Winter Green Research 2013].

2.1. Business Process Model and Notation - *BPMN*

O padrão Business Process Model and Notation, ou BPMN, foi criado com o objetivo de fornecer uma notação facilmente compreensível por todos os usuários, desde os analistas que criam os rascunhos iniciais dos processos até os desenvolvedores responsáveis por implementar os processos e, finalmente, para os usuários que irão gerenciar e monitorar esses processos [Model 2011]. A versão BPMN 2.0, a mais recente, define um padrão XML para arquivos contendo dados sobre o modelo e o funcionamento do processo bem como a sua representação visual [Kurz et al.], isto permite que o XML seja analisado para obter-se informações importantes sobre os processos. A maioria das ferramentas BPM disponibiliza a exportação do processo em formato XML seguindo o padrão BPMN. No padrão BPMN um processo é descrito como um diagrama de elementos de fluxo, que são: Tasks (Tarefas), Events (Eventos), Gateways e Sequence Flows [Kurz et al.]. Na Figura 1, podem ser vistos os principais elementos que compõem um diagrama BPMN e que serão importantes para este trabalho.

Uma *Activiti* é a menor parte de um diagrama BPM, e é utilizada quando o diagrama não pode ser dividido mais detalhadamente. Quando uma *Activiti* está inserida no contexto de um processo, ela é chamada de *Task*. Geralmente as Tasks são executadas por

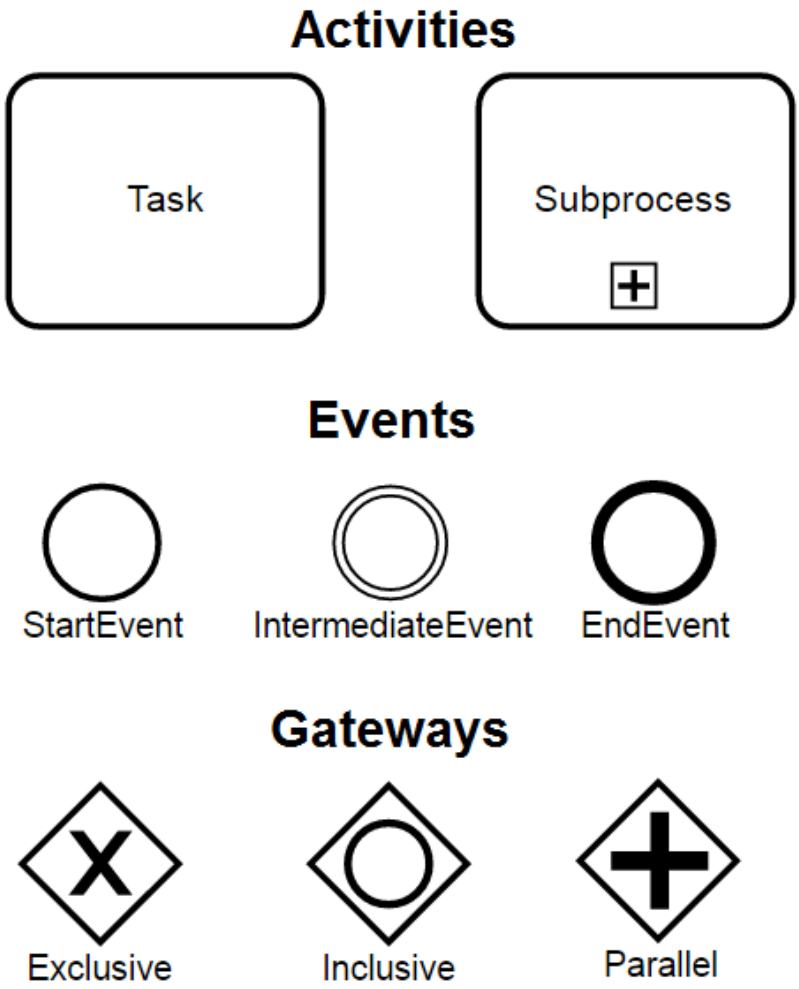

Figura 1. Principais elementos de um processo de acordo com o padrão BPMN.

um usuário final ou por uma aplicação. Existem diferentes tipos de Tasks para representar os diferentes comportamentos que cada tarefa pode representar. Por exemplo, uma *user-Task* representa uma tarefa em que um usuário deve executar uma ação. Um elemento do tipo *SubProcess* é uma abstração de um pequeno conjunto de *Tasks*.

Um *Event* é algo que ”acontece”durante o curso de um processo [Model 2011]. Existem três tipos principais de eventos: *Start Event* (indica o início do processo), *End Event* (indica um fim do processo) e *Intermediate Events* (indica um evento entre o início e o fim do processo).

Gateways são usados para controlar o fluxo do processo. Em gateways do tipo *Exclusive*, apenas um dos fluxos que partem do gateway poderão ser seguidos, já em gateways do tipo *Inclusive* um ou mais fluxos podem ser seguidos. Em gateways do tipo *Parallel* todos os fluxos são executados em paralelo.

Um elemento *Sequence Flow* é usado para exibir a ordem em que os demais elementos são executados em um processo. Cada *sequenceFlow* possui apenas uma fonte e um alvo. Analisando os elementos *sequenceFlow* é possível identificar todo o fluxo do

processo.

2.2. BPM e Teste

Os sistemas BPM geralmente oferecerem recursos para definição e modelagem de processos em BPMN, controle da execução e monitoramento de atividades dos processos [Forrester Research 2013]. Nota-se, no entanto, que a preocupação com testes não fica evidente nas ferramentas BPMS. De fato, examinando-se o material promocional e a documentação disponível sobre os principais BPMS, observa-se uma ênfase em etapas de modelagem e execução. Visivelmente, tais recursos são um diferencial no desenvolvimento de aplicações de BPM, em comparação ao desenvolvimento de software em geral. No entanto, aplicações de BPM também estão sujeitas a defeitos e, por isso, podem se beneficiar de avanços na área de testes de software.

Existem trabalhos que chamam a atenção para a importância e para a as dificuldades trazidas pela execução de testes, bem como trazem alternativas para a melhoria da seleção de casos de teste [Böhmer and Rinderle-Ma 2015]. Esses trabalhos fortalecem a ideia da importância de informações para facilitar a criação de testes para processos ou sistemas BPM, no entanto, os trabalhos geralmente abordam a criação de casos de teste baseados em modelos de processo e não em sistemas em si. Também não é descrita uma forma de automatizar ou executar os casos de teste criados. Existem também trabalhos que visam maneiras de automatizar e melhorar o monitoramento da conformidade dos modelos de processos [Ly et al. 2015, Van der Aalst et al. 2012] e que, novamente, são mais voltados para os processos em si e sua administração do que para sistema/software.

3. Descrição da abordagem

O objetivo deste trabalho é gerar dados para o Teste Funcional Automatizado de aplicações BPM com o apoio da notação BPMN. Um aspecto importante para o teste funcional são quais fluxos ou etapas serão testadas, pois estes podem definir ou limitar o teste, interferindo assim na qualidade e na cobertura dos testes. No entanto, a construção dessas informações pode ser uma tarefa complexa.

Com o objetivo de obter maiores informações que pudessem auxiliar no processo de teste, decidiu-se analisar os arquivos XML no formato BPMN exportados pelas ferramentas BPMS para extrair destes arquivos as informações. Para tratar essas informações, optou-se por analisar o processo de modo a transforma-lo em uma tabela contendo todos os possíveis fluxos que podem ser executados no processo para então serem realizadas as devidas análises.

Para analisar os arquivos foi elaborada um estratégia que percorre os arquivos XML e manipula as informações necessárias. A visão geral da abordagem pode ser vista na Figura 2. Basicamente, o arquivo BPMN é utilizado como entrada e, enquanto o arquivo é percorrido, os elementos são analisados através da nomenclatura padrão das *tags* e então é executada uma lógica para criar os diferentes cenários de teste. Ao fim da execução, é gerado um arquivo Excel contendo uma tabela com os todos os possíveis cenários/fluxos do processo.

3.1. Funcionamento da estratégia criada

Nesta abordagem, foi utilizada a linguagem Java, principalmente por esta linguagem já possuir uma opção sólida para a análise de documentos XML [Oracle 2015]. O elemento

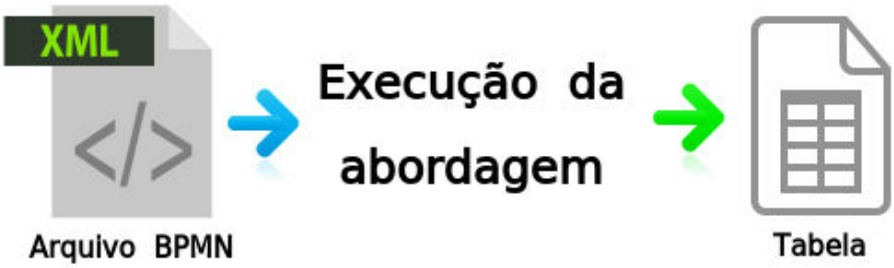

Figura 2. Visão geral da abordagem utilizada

BPMN mais importante para esta estratégia de análise é o *Sequence Flow*. Cada *Sequence Flow* possui um atributo *sourceRef* que indica de onde este *sequenceFlow* vem, ou sua ”fonte”, e um atributo *targetRef* que indica para onde ele vai, ou seja, seu alvo. Por conter esses atributos os elementos deste tipo permitem percorrer todo o diagrama. Ao iniciar a execução, é solicitado ao usuário o caminho para o arquivo BPMN e a ID do processo a ser avaliado. Um diagrama pode conter mais de um processo e, nesse caso, o usuário pode escolher que todos processos sejam avaliados.

Foi criada a classe *Node* que define um tipo de objeto que guarda informações básicas sobre as tarefas do processo e um *array* de objetos da mesma classe. Durante a execução, um *array* de objetos da classe Node é preenchido formando assim um *array* de adjacências. O método principal percorre o processo recursivamente através dos elementos do tipo *Sequence Flow*, enquanto um tarefa for encontrada, um objeto da classe Node é criado e o método é chamado recursivamente de modo a retornar o *array* de adjacências para este objeto.

A partir do *array* de adjacências são criados os fluxos possíveis pelos quais o processo pode passar. O método que cria os fluxos percorre recursivamente o *array* de adjacências criado anteriormente e ”constrói”um novo fluxo possível, tendo como condição de parada o encontro de uma das últimas tarefas a serem executadas. Uma tarefa é uma das últimas quando o elemento que a sucede no fluxo for elemento do tipo*End Event*. Ao encontrar uma tarefa final, a informação é armazenada e é iniciada a construção de um novo fluxo possível.

Os fluxos obtidos são base para a criação da tabela. Para criar a tabela, cada tarefa existente no processo representará uma coluna na tabela e cada fluxo representará uma nova linha. Para cada fluxo, se a tarefa estiver presente a coluna será marcada com ”X”caso contrário a coluna será marcada com um ”0”. Caso mais de um processo seja avaliado ao mesmo tempo, a tabela referente à cada processo estará separada individualmente no arquivo final.

Na Tabela 1, pode ser visto uma tabela resultante da execução dessa estratégia para um processo com cinco *tasks* e dois fluxos possíveis. Com a geração desta tabela podem ser realizadas diversas análises sobre o processo: número de fluxos possíveis, identificar gargalos, etc.

Tabela 1. Exemplo de tabela resultante

Quotation Handling	Approve Order	Order Handling	Shipping Handling	Review Order
X	X	0	0	0
X	X	X	X	X

3.2. Dificuldades/Limitações

A maioria das dificuldades encontradas são relacionadas à estrutura dos processos e como a implementação interpretaria essas características. Na criação dos fluxos possíveis, uma dificuldade ocorreu devido aos desvios causados por elementos do tipo *Gateway*. Para isso, para cada Node criado no *array* é guardado o tipo do elemento que o antecede. Assim, na criação dos fluxos, elementos que partem de um desvio de fluxo precisaram ser tratados para criar os fluxos corretamente, por exemplo: se dois elementos, X e Y, vem de um *gateway* do tipo exclusivo, os fluxos obtidos até estes elementos serão duplicados, ou seja, metade dos fluxos passaram apenas por X e metade dos fluxos passarão apenas por Y.

No caso de *gateways* do tipo Paralelo, onde os fluxos são executados ao mesmo tempo, foi necessária fazer uma escolha para representar estes fluxos na tabela. Assim, decidiu-se por expressar cada fluxo em paralelo separadamente. Estimou-se que esta opção facilitaria na visualização dos fluxos para os testes já que, para a execução dos testes funcionais automatizados, é analisado cada fluxo como um ”cenário”diferente.

Devido ao fato da estratégia implementada ser executada recursivamente e percorrer o processo baseado no fluxo dos elementos do tipo *SequenceFlow*, ocorreram problemas em processos onde uma parcela do processo também é executada recursivamente. Estes problemas foram causados devido ao fluxo nunca encontrar um ”fim”. Para solucionar esses problema foi criado um ”delimitador de recursão”que define e controla o número de vezes em que o mesmo *SequenceFlow* será executado.

Algumas ferramentas exportam o diagrama para o formato BPMN inserindo um “tipo” antes no nome de cada tag XML, por exemplo, a *tag* de nome *task* pode estar representada como “semantic:task” e isso pode impedir que o parser XML do Java identifique os elementos. Assim, foi necessário preparar métodos para tratar este tipo de situação removendo os elementos encontrados antes dos nomes de *tags*.

Apesar de utilizarem o mesmo padrão BPMN, algumas ferramentas podem utilizar nomes diferentes para representar tarefas no arquivo XML. Por isso, dependendo do BPMS que for utilizado, pode ser necessário substituir o nome das tarefas tratadas diretamente no código.

4. Teste Funcional de Software e criação de códigos para os testes

O teste funcional é um tipo de teste que permite verificar as saídas de um sistema produzidas a partir de entradas pré-definidas. Este tipo de teste permite testar as funcionalidades, requerimentos e regras de negócio presentes no software [Molinari 2003] verificando a existência de erros, o que auxilia na melhoria da qualidade do software.

Uma das principais medidas para o teste de software é a cobertura de teste. A cobertura de teste mede a abrangência do teste e pode ser expressa pela cobertura dos casos de testes ou pela cobertura do código executado. Existem diversos trabalhos que abordam

a importância da cobertura de testes de software [Zhu et al. 1997, Bieman et al. 1996], inclusive ligando o crescimento da qualidade e confiabilidade do software ao crescimento da cobertura dos testes [Malaiya et al. 2002].

No entanto, as atividades executadas para criar testes funcionais com uma boa cobertura podem muitas vezes se tornar exaustivas e trabalhosas, dificultando assim a execução dos testes de forma adequada. Com o objetivo de melhorar a qualidade da análise e o tempo de execução dos testes, foram criados os testes automatizados, que proporcionam a execução dos testes mais rapidamente [Fantinato et al. 2005]. Quando executada corretamente, a automação de teste é uma das melhores formas de reduzir o tempo de teste no ciclo de vida do software, diminuindo o custo e aumentando a produtividade do desenvolvimento de software como um todo, além de, consequentemente, aumentar a qualidade do produto final.

Para executar os testes funcionais automatizados em aplicações Web, pode-se utilizar ferramentas livres como Selenium [Selenium 2015], Watir [Watir 2015] ou Geb [Geb 2015]. No trabalho anterior [de Moura and Charão 2015], onde os testes funcionais se mostraram mais promissores, foi utilizada a ferramenta Selenium, aliada ao Cucumber-JVM [Cucumber Limited 2015a] para descrição dos testes. A escolha foi motivada pelo grande número de referências ao Selenium na Web, confirmadas por um trabalho que apresentou resultados satisfatórios com Selenium e Cucumber [Pannu , Chiavegatto et al. 2013].

4.1. Teste Funcional Automatizado utilizando *Selenium* e *Cucumber-JVM*

O processo para a execução de um teste funcional utilizando o Selenium aliado ao Cucumber-JVM é composto de cinco etapas: captura da interação do usuário com a aplicação e exportação do código gerado, criação dos cenários de teste, criação das definições dos passos de teste, implementação dos métodos para cada passo e, por fim, execução do teste.

Para efetuar a captura da interação do usuário com a aplicação é utilizado o Selenium IDE (*Integrated Development Environment*), que permite gravar as ações do usuário conforme elas são executadas. Assim, a funcionalidade que deseja ser testada deve ser executada para que os passos sejam gravados. Após a captura da interação é possível exportar o *script* utilizando diversas linguagens de programação disponíveis, neste caso foi escolhida a linguagem Java.

Os testes utilizando o Cucumber são compostos, basicamente, por dois arquivos: arquivos que especificam as funcionalidades *features* e por arquivos de definição de passos *steps*.

Os arquivos com as funcionalidades são escritos utilizando a linguagem Gherkin [Cucumber Limited 2015b] e são compostos por cenários, os cenários representam uma fração da aplicação que vai ser testada. Um cenário também pode ser descrito como a definição, em ordem de execução, das etapas que são executados nessa fração, bem como dos resultados esperados para validar a aplicação. Por exemplo, após um *gateway* exclusivo com dois fluxos disponíveis, já é possível obter dois cenários possíveis: um cenário executando cada fluxo.

Para que cada etapa do cenário seja executada, é necessária a criação de *steps*

que irão traduzir os passos definidos na linguagem Gherkin para ações que vão interagir com o sistema. Cada *step*, geralmente, irá chamar um método Java que irá efetivamente executar a interação com a aplicação. A implementação destes métodos pode ser feita apenas utilizando o código extraído através do Selenium após a captura das ações do usuário.

Assim, mesmo utilizando o Cucumber e Selenium para facilitar a criação dos testes, ainda é preciso analisar o processo e extrair os cenários de teste que devem ser criados manualmente no arquivo de *features*. Além dos cenários, também deve ser criado manualmente o arquivo de *steps*, contendo uma ”passo”correspondente a cada etapa de todos os cenários criados. Esta etapa pode ser trabalhosa, principalmente quando for necessário testar todos diversos cenários que um processo pode ter ou quando o processo for extenso.

4.2. Criação de códigos utilizados no teste funcional

A geração da tabela com informações relevantes para o teste do processo foi a principal motivação da criação da ferramenta. No entanto, além das análises que a própria tabela possibilita, também é possível utilizar a estratégia criada para gerar outros elementos conforme a necessidade de cada domínio. No caso deste trabalho, por o teste funcional automatizado utilizando o Selenium e o Cucumber se mostrou promissor em um trabalho anterior [de Moura and Charão 2015], decidiu-se por avaliar a possibilidade de aproveitar a abordagem criada para também gerar elementos importantes para o teste: o arquivo de *features* contendo os cenários cenários e o arquivo de *steps*.

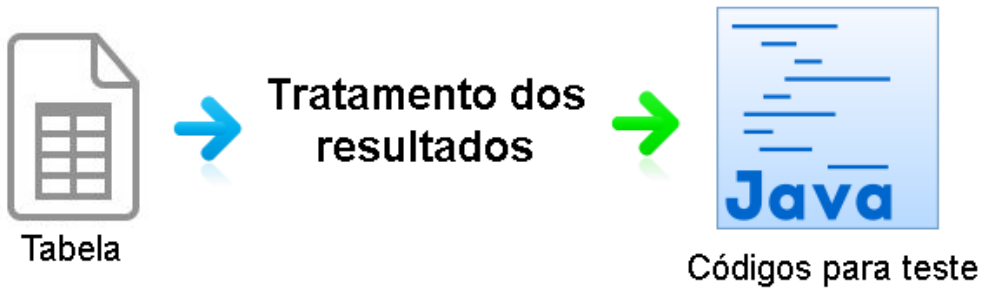

Figura 3. Uma das opções de uso da abordagem: geração de códigos para o teste funcional

O arquivo contendo os cenários de teste é essencial para o teste funcional com o Cucumber, no entanto a criação dos arquivos de *features* contendo os cenários e do arquivo de *steps* pode ser trabalhosa, como foi mencionado na seção anterior. Uma visão geral deste exemplo pode ser vista na Figura 3, onde a tabela resultante da execução da abordagem é utilizada para gerar os arquivos com códigos. Para criar estes elementos, foram adicionadas algumas funcionalidades na ferramenta para que os dados já obtidos fossem tratados. Assim, cada fluxo que foi obtido pela abordagem, e consta na tabela de saída, foi considerado um cenário diferente. Como trata-se de um teste funcional, apenas tarefas que podem ser executadas por um usuário foram ser avaliadas (*userTask*,*manualTask*...). Assim, para cada fluxo é criado um novo cenário no arquivo de *features* utilizando a notação do Cucumber e contemplando apenas as tarefas que podem ser executadas por

Tabela 2. Resumo dos processos avaliados

Número de tarefas	Número de Gateways	Tipo do Gateway	Fluxos possíveis
3	0	Sem divisão de fluxos	1
3	1	Exclusivo	2
4	0	Divisão de fluxos sem Gateways	2
5	0	Divisão de fluxos sem Gateways	3
5	1	Exclusivo	3
6	2	Exclusivo	3
7	1	Exclusivo	2
8	3	Inclusivo, Exclusivo e Paralelo	3
8	3	Paralelo, Exclusivo e Inclusivo	5
10	1	Exclusivo	2

um usuário. O método correspondente a cada passo do cenário é criado ao mesmo tempo e inserido no arquivo de *steps*.

5. Avaliação

Para avaliar a abordagem foram buscados processos já existentes, de diferentes domínios, contendo ao menos três tarefas e uma divisão de fluxo com ou sem *gateway*, também era interessante que os processos selecionados tivessem algumas diferenças de notação para verificar como a ferramenta trataria essas diferenças. Assim, os processos utilizados para a validação foram obtidos no site da *Object Management Group* (OMG) e estão disponíveis no site da organização.

A Tabela 2 apresenta uma visão geral dos processos avaliados, as informações referentes ao número de tarefas e de fluxos foram extraídas das tabelas obtidas após executar a abordagem para cada um dos processos. A divisão de fluxos é importante para os testes pois ela delimita quantos fluxos existem no processo, ou seja, quantos fluxos podem ser testados.

Pode-se dizer que, quanto maior o número de fluxos possíveis, mais complexa é análise para a criação dos testes e, assim, a criação dos elementos obtidos através da abordagem utilizada mostra-se útil.

Outro ponto importante é que, em alguns casos, podem existir divisões de fluxo sem *gateways*. As divisões sem *gateways*, dependendo da ferramenta BPMS utilizada para gerar o processo, só podem ser visualizadas através da análise do documento. Isso pode atrapalhar a análise ”manual”, mas a abordagem adotada permite que essas divisões sejam detectadas e tratadas automaticamente.

Com a análise do arquivo BPMN através da ferramenta criada foi possível obter a tabela relacionando as tarefas e os fluxos possíveis dentro de um processo. Por exemplo, o processo na Figura 4 possui *gateways* de diferentes tipos, gerando assim várias divisões de fluxo e aumentando os fluxos que podem ser seguidos.

Ao executar a ferramenta utilizando o arquivo BPMN referente ao processo da Figura 4 foi obtida a Tabela 3. Nesta tabela, cada coluna representa uma tarefa do diagrama e cada linha representa um fluxo possível, facilitando a visualização do processo. A geração da tabela pode auxiliar a reduzir o tempo de análise necessário para executar o teste funcional dos processos pois possibilita que os fluxos possíveis sejam vistos

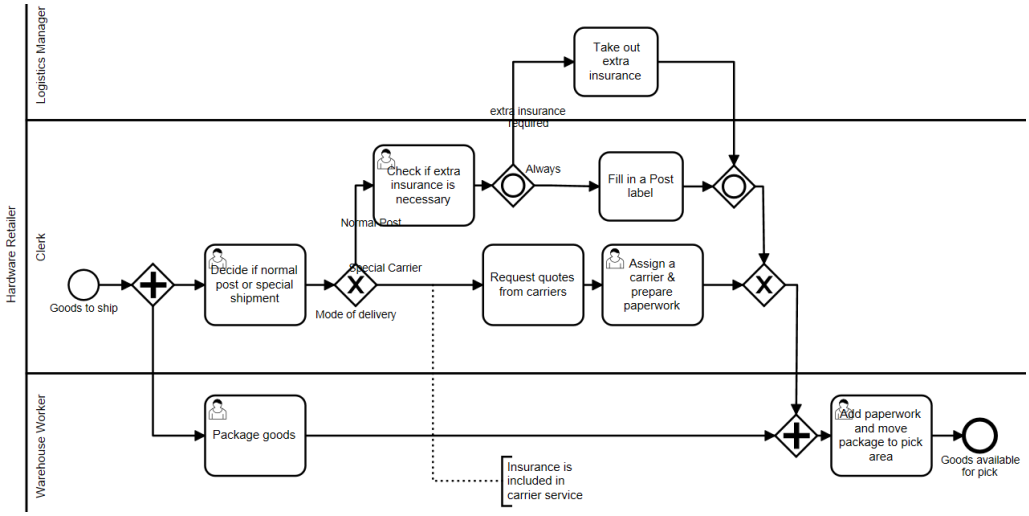

Figura 4. Exemplo de Processo. Adaptado de: Object Management Group

Tabela 3. Tabela obtida através do arquivo BPMN

Decide if normal...	Request quotes...	Assign a carrier...	Check if extra...	Take out extra insurance	Fill in...	Package goods	Add paperwork and...
X	X	X	0	0	0	0	X
X	0	0	X	X	0	0	X
X	0	0	X	0	X	0	X
0	0	0	0	0	0	X	X
X	0	0	X	X	X	0	X

rapidamente.

A tabela também permite que sejam feitas análises importantes para o teste funcional do processo como, por exemplo: identificar gargalos, identificar quais tarefas são executadas mais frequentemente, identificar os caminhos mais importantes para os testes, entre outros. Por exemplo, na Tabela 3 pode-se dizer que o teste da primeira e da última tarefa do processo é importante, pois a grande maioria dos fluxos possíveis passarão por essas tarefas. Da mesma forma que permite análises importantes para os testes, a tabela gerada também auxilia na criação de testes mais completos pois permite visualizar todos os fluxos possíveis de serem testados dentro de um processo.

Nesse trabalho abordamos a criação, a partir da tabela gerada pelo abordagem, de códigos para teste funcional de processos utilizando as ferramentas de automação Selenium e Cucumber-JVM. Para executar o teste completo do processo, testando todas as possibilidades e representando as informações obtidas na tabela, seria necessário planejar os cenários de teste e criá-los manualmente. Para criar os cenários também é necessário identificar quais tarefas são executadas por um usuário em um processo que possui várias *tasks*. Por fim, também seria necessário criar os *steps* para cada etapa do cenário manualmente.

Executando o parser criado utilizando o arquivo BPMN referente ao diagrama da Figura 4 como exemplo, obtemos os cenários de teste representados no trecho de código abaixo.

```
Feature: Testing BPM Processes

Scenario: Hardware Retailer 0
```

```
Given I am on task Decide if normal post or special shipment
When
Then
When I am on task Assign a carrier & prepare paperwork
Then
When I am on task Add paperwork and move package to pick area
Then

Scenario: Hardware Retailer 1
Given I am on task Decide if normal post or special shipment
When
Then
When I am on task Check if extra insurance is necessary
Then
When I am on task Add paperwork and move package to pick area
Then

Scenario: Hardware Retailer 2
When I am on task Package goods
Then
When I am on task Add paperwork and move package to pick area
Then
```

Para o teste funcional do processo em questão, três cenários são possíveis, representando as duas principais divisões de fluxo encontradas no processo. A primeira divisão de fluxo formada pelo *gateway* do tipo paralelo logo no início do processo, está contida no cenário de forma que os cenários *Hardware Retailer 0* e *Hardware Retailer 1* fazem parte de um fluxo e o cenário *Hardware Retailer 2* representa o segundo fluxo possível. A segunda divisão de fluxo no processo da Figura 4 é formada pelo *gateway* do tipo exclusivo, e essa divisão é representada pelos cenários *Hardware Retailer 0* e *Hardware Retailer 1*. Há ainda uma terceira divisão de fluxo no processo, esta divisão não foi levada em conta pelo parser pois as tarefas subsequentes a essa divisão não são tarefas executadas pelo usuário.

Para cada etapa nos cenários criados será criado um método correspondente dentro do arquivo de *steps* chamado *stepDefinition*. Uma parte do arquivo de *steps* resultante da análise do processo em questão pode ser visto no trecho de código abaixo. Como pode ser visto neste codigo, para a etapa de cenário chamada *I am on the task Decide if normal post or special shipment* foi criado um método com um nome genérico e contendo a anotação *@Given("^I am on the task Decide if normal post or special shipment$")*. Os métodos podem ser preenchidos posteriormente com os dados gravados pelo Selenium na etapa de captura da interação com a aplicação.

```
import cucumber.api.java.en.*;
import cucumber.runtime.PendingException;
public class StepsDefinition{
@Before
public void beforeScenario(){}
```

```
@After
public void afterScenario(){}
//Methods for process Hardware Retailer
@Given("^I am on the task Decide if normal post or special shipment$")
public void method0() throws Exception {}
@when("^name$")
public void method1() throws Exception {}
@Then("^name$")
public void method2() throws Exception {}
@When("^I am on the task Assign a carrier & prepare paperwork$")
public void method3() throws Exception {}
```

Neste exemplo, apenas um processo foi analisado, mas arquivos BPMN podem conter vários processos. Caso for necessário gerar os dados para todos os processos dentro de um arquivo, o arquivo contendo a tabela resultante será composto por várias tabelas, uma para cada processos analisado. Como é gerado o código de teste referente a todos os fluxos que contêm *tasks* executadas por usuários é possível executar o teste para todos os fluxos ou, por outro lado, também é possível utilizar a tabela gerada para verificar quais os fluxos mais importantes e utilizar os códigos gerados referentes apenas a esses fluxos.

Nesse trabalho foi possível obter uma tabela importante para o teste dos processos, permitindo que sejam obtidas informações úteis e importantes para a execução de testes completos e com boa cobertura. As informações geradas também podem auxiliar na análise e a reduzir o tempo necessário para efetuar os testes. Como um exemplo de utilização dos dados obtidos foram gerados dois arquivos essenciais para o teste funcional automatizado de aplicações BPM: o arquivo com os cenários de teste e o arquivo com os métodos para executar cada etapa dos cenários.

6. Conclusão

O objetivo desse trabalho foi gerar dados para a execução do teste funcional automatizado de aplicações BPM. Foi possível obter uma tabela com todos fluxos possíveis dentro do processo. A criação da tabela possibilita a criação de testes mais completos e com boa cobertura. A tabela gerada também permite que sejam feitas análises importantes sobre o processo como, por exemplo, identificar gargalos.

Além da obtenção da própria tabela, a abordagem permite que as informações sejam usadas como entrada na criação de outros elementos. As informações podem ser reutilizadas das mais diversas formas, dependendo de qual estratégia/ferramenta de teste se deseja utilizar. Nesse trabalho decidiu-se por abordar o teste funcional com as ferramentas Selenium e Cucumber. Em um trabalho anterior a execução do teste funcional automatizado em aplicações BPM com as ferramentas mencionados se mostrou promissora, mas trabalhosa. A abordagem criada auxilia nesse ponto pois permitiu que se utilizassem as informações obtidas para gerar códigos necessários para a execução dos testes, auxiliando na automação destes.

A criação automatizada da tabela e dos dois arquivos auxilia a criação dos testes para os sistemas BPM, diminuindo o tempo de análise necessário para a criação dos elementos para o teste dos processos bem como tornando o teste mais completo, pois verifica

todos os fluxos que um processo pode seguir. Assim, a qualidade e a cobertura dos testes também pode ser melhorada pois todos os fluxos e cenários possíveis são analisados. Por consequência, a qualidade dos sistemas também pode ser beneficiada com um processo de teste mais completo.

Como trabalhos futuros têm se a melhoria de alguns aspectos técnicos da abordagem como melhorar, por exemplo, o tratamento a processos com recursão e reconhecimento dos diferentes nomes de *tag*. Também podem ser avaliadas outras formas de utilizar as informações extraídas de forma a gerar outros elementos.

Referências

[AB 2009] AB (2009). *Guide to the business process management common body of knowledge (BPM CBOK®): ABPMP BPM CBOK®-[version 2.0-second release]*. ABPMP.

[Bieman et al. 1996] Bieman, J. M., Dreilinger, D., and Lin, L. (1996). Using fault injection to increase software test coverage. In *Software Reliability Engineering, 1996. Proceedings., Seventh International Symposium on*, pages 166–174. IEEE.

[Böhmer and Rinderle-Ma 2015] Böhmer, K. and Rinderle-Ma, S. (2015). A genetic algorithm for automatic business process test case selection. In *On the Move to Meaningful Internet Systems: OTM 2015 Conferences*, pages 166–184. Springer.

[BPMN 2016] BPMN (2016). Business Process Management and Notation.

[Chiavegatto et al. 2013] Chiavegatto, R. et al. (2013). Especificação e automação colaborativas de testes utilizando a técnica BDD. In *XII Simpósio Brasileiro de Qualidade de Software*, pages 334–341.

[Cucumber Limited 2015a] Cucumber Limited (2015a). Cucumber-JVM Reference.

[Cucumber Limited 2015b] Cucumber Limited (2015b). Gherkin Reference.

[de Moura and Charão 2015] de Moura, J. L. and Charão, A. (2015). Automação de testes em aplicações de bpms: um relato de experiência. In *XIV Simpósio Brasileiro de Qualidade de Software*, pages 212–219.

[Fantinato et al. 2005] Fantinato, M., da Cunha, A. C. R., Dias, S. V., Cardoso, S. A. M., and Cunha, C. A. Q. (2005). Autotest–um framework reutilizável para a automação de teste funcional de software. *Cad. CPqD Tecnologia*, 1(1):119–131.

[Forrester Research 2013] Forrester Research (2013). The forrester wave: BPM suites, Q1 2013.

[Geb 2015] Geb (2015). Groovy Browser Automation.

[Graham and Fewster 2012] Graham, D. and Fewster, M. (2012). *Experiences of Test Automation: Case Studies of Software Test Automation*. Addison-Wesley.

[Kurz et al.] Kurz, M., Menge, F., and Misiak, Z. Diagram interchangeability in bpmn 2.

[Ly et al. 2015] Ly, L. T., Maggi, F. M., Montali, M., Rinderle-Ma, S., and van der Aalst, W. M. (2015). Compliance monitoring in business processes: Functionalities, application, and tool-support. *Information systems*, 54:209–234.

[Malaiya et al. 2002] Malaiya, Y. K., Li, M. N., Bieman, J. M., and Karcich, R. (2002). Software reliability growth with test coverage. *Reliability, IEEE Transactions on*, 51(4):420–426.

[Model 2011] Model, B. P. (2011). Notation (bpmn) version 2.0. *OMG Specification, Object Management Group*.

[Molinari 2003] Molinari, L. (2003). *Testes de software: produzindo sistemas melhores e mais confiáveis: qualidade de software: soluções, técnicas e métodos*. Érica.

[Oracle 2015] Oracle (2015). Java Dom XML Reference.

[Pannu] Pannu, Y. S. Test automation using cucumber and selenium webdriver.

[Selenium 2015] Selenium (2015). Selenium browser automation.

[Van der Aalst et al. 2012] Van der Aalst, W., Adriansyah, A., and van Dongen, B. (2012). Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192.

[van der Aalst 2013] van der Aalst, W. M. P. (2013). Business process management: A comprehensive survey. *ISRN Software Engineering*, 2013(507984).

[Watir 2015] Watir (2015). Web Application Testing in Ruby.

[Weske 2012] Weske, M. (2012). *Business Process Management: Concepts, Languages, Architectures*. Springer, 2nd edition.

[Winter Green Research 2013] Winter Green Research (2013). Business process management (BPM) cloud, mobile, and patterns: Market shares, strategies, and forecasts, worldwide, 2013 to 2019.

[Zhu et al. 1997] Zhu, H., Hall, P. A., and May, J. H. (1997). Software unit test coverage and adequacy. *Acm computing surveys (csur)*, 29(4):366–427.