Utilizando OSWorkflow em um Web Service RESTful, a fim de automatizar processos de negócios

Hernane Batista Rabelo, Marcos Alberto Lopes da Silva

Instituto de Informática – Centro Universitário do Triângulo (UNITRI)
Caixa Postal 309 – 38.411-106 – Uberlândia – MG – Brasil

hernane_rabelo@hotmail.com, maloes21@gmail.com

Resumo. *Com o crescimento das empresas, seus processos tendem a se tornarem complexos. Com isso, o avanço tecnológico vem auxiliando as empresas a monitorar seus processos, através de ferramentas de orquestração. No entanto, como uma ferramenta de orquestração está centralizada no processo, necessita-se de uma integração facilitada, e neste momento os Web Services auxiliam nesta integração. Neste trabalho, serão abordados conceitos de workflow e Web Services. Também sera abordado um framework open source de orquestração chamado OSWorkflow, que acoplado em um Web Service RESTful, desenvolvido com Spring Boot, auxilia na integração entre as aplicações, com isso automatizando os processos de negócios. Finalizando este trabalho, será demonstrado uma aplicação de orquestração de processos, utilizando as tecnologias citadas.*

1. Introdução

Com tantos processos em uma empresa, é evidente que um gestor necessite compreender todo o processo. Existem algumas metodologias que auxiliam nesse entendimento, facilitando assim a melhoria dos processos.

No entanto, alguns processos se tornam demorados e complexos, e assim, ocorrem desvios e perda de informações, e neste momento ferramentas de orquestração de processos, ou fluxo de trabalho, auxiliam as empresas a terem seus processos mais eficazes, aumentando seu desempenho processual.

Pretende-se com este artigo abordar uma ferramenta de *workflow open source* chamada *OSWorkflow*, que auxilia na construção de fluxos, otimizando processos de negócio, com isso, deixando os processos mais eficazes. Contudo, esta ferramenta necessita ter seus fluxos desenvolvidos em XML, e não é *plug & play* (conecte e rode), criando assim alguns obstáculo ao seu uso [LAZO, 2007].

Este artigo também abordará alguns conceitos de Web Services, pois como um *workflow* requer comunicar com outras aplicações, nada melhor que um Web Service para fazer essa integração. Com isso, será abordado o Web Service RESTful, que segue o conceito arquitetural REST.

O artigo está estruturado da seguinte forma, primeiramente serão abordados os conceitos de *workflow* e Web Service, demonstrando assim a vantagem de se utilizar

uma ferramenta de orquestração de dados integrada em um Web Service. Em seguida será abordada a ferramenta do estudo de caso, OSWorkflow, e uma outra ferramenta que auxilia no desenvolvimento de Web Service, o Spring Boot. Em sequência, será demonstrada a aplicação desenvolvida para este artigo, com uso das ferramentas exemplificadas no tópico 3. Finalizando o artigo, será realizada a conclusão de todo o trabalho, evidenciando assim as vantagens em utilizar um *workflow* com um Web Service RESTful.

2. Definindo *workflow* e RESTful

Neste tópico serão abordados alguns conceitos sobre w*orkflow* e Web Services RESTful, evidenciando assim as vantagens de utilizar uma ferramenta de w*orkflow* para otimizar processos de negócios, assim como as vantagens de se utilizar um Web Service RESTful para integração de aplicação.

2.1 Entendendo um workflow

Workflow ou, no português, fluxo de trabalho, define uma sequência de passos para se atingir determinado resultado. Cada passo no *workflow* possibilita uma regra específica, fazendo com que cada um seja independente [LAZO, 2017].

Uma forma de simplificar a construção de um w*orkflow,* consiste em utilizar a BPMN (*Business Process Modeling Notation* – Notação de Modelagem de Processos de Negócio), que consiste basicamente de desenhos, que auxiliam visualmente a entender os processos. Com uso da BPMN é possível visualizar e modificar todo processo, com isso melhorando e automatizando o mesmo [NOGUEIRA, 2014].

Na figura 1, apresenta-se um w*orkflow* utilizando BPMN, onde cada elemento geométrico tem uma representação. Os retângulos são atividades a serem realizadas, o círculo com a bordas verde representa o inicio do processo, o círculo com a borda vermelha representa o final do processo, os losangos são conhecidos como *Gateway*, que são condições. O *Gateway* utilizado é o exclusivo, fazendo o fluxo seguir apenas o passo que a condição for verdadeira. As setas são utilizadas para representar a sequência a ser seguida. Existem outras formas geométricas, porém não é o foco deste trabalho detalhar todas as formas [NOGUEIRA, 2014].

Existem várias ferramentas de w*orkflow* que auxiliam na busca em automatizar processos, possibilitando uma rastreabilidade mais eficaz, aumentando a confiabilidade da informação e deixando o processo mais ágil. Mais a frente será abordado um *framework* de w*orkflow* chamado OSWorkflow [LAZO, 2007].

Um motivo de se automatizar um processo com uso de ferramentas de w*orkflow*, é devido a grande chance de se perder informação durante todo o processo. Para exemplificar esta dificuldade de perder informações durante um processo, imagine um processo que necessite colher assinaturas de várias pessoas em um documento, no qual a chance de se perder esse documento é muito alta, ou existe a possibilidade do documento ficar parado em alguma mesa, devido a pessoa responsável não estar no

local naquele momento, ou estar sobrecarregado de serviço, portanto neste caso já tem uma chance de perder informação durante o processo [LAZO, 2007].

A figura 1 demonstra um diagrama referente ao pedido de férias, partindo de um funcionário assinando o pedido, depois enviando para o gestor aprovar. Caso seja recusado, é realizado o cancelamento da solicitação e notifica o funcionário, depois finaliza o processo. No entanto, se o gestor aprovar a solicitação, é enviado para a aprovação do recursos humanos, se aprovado, o funcionário é notificado e finaliza o fluxo, se não autorizado, é realizado o cancelamento da solicitação das férias e notifica o funcionário, logo depois finaliza o fluxo.

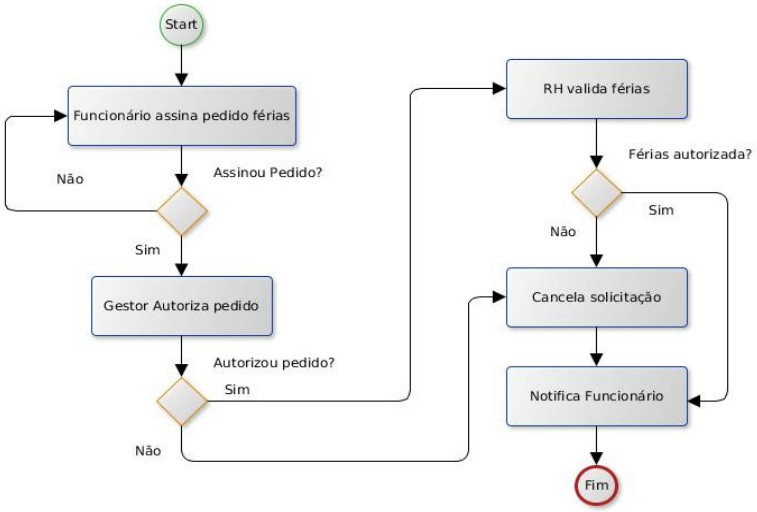

Figura 1 – Diagrama de uma solicitação de pedido de férias

Esse processo citado na figura 1 demandaria algum tempo se fosse realizado sem o uso de uma ferramenta de orquestração, o documento poderia ser perdido entre um passo e outro. Se o fluxo fosse maior, a complexidade aumentaria, fazendo com que possa ocorrer mais erros durante todo processo, e esse é um dos motivos para se usar ferramentas de w*orkflow* [LAZO, 2007].

2.2 Conceituando Web Service RESTful

Com o crescimento tecnológico, surgem muitas aplicações desenvolvidas em diversas linguagens de programação. Nesta hora, o uso de Web Service auxilia na integração entre as aplicações

Para entender o que é um Web Service RESTful primeiro será explicado um pouco sobre o estilo arquitetural REST (*Representation State Transfer* – Transferência de Estado Representacional*).*

O padrão arquitetural REST foi desenvolvido pelo Ph.D (*Doctor of Philosophy* – Doutor em Filosofia) Roy Fielding no ano de 2000, ele também foi um dos principais autores do protocolo HTTP utilizado na Web. Apesar do conceito REST ser antigo, visto que as tecnologia tem evoluído muito rápido, o conceito vem sendo utilizado na

construção de API (*Application Programming Interface* – Interface de Programação de Aplicativos) [FIELDING, 2000].

Existe uma confusão entre o conceito de REST e RESTful, para resumir a diferença entre eles de forma rápida, RESTful é uma implementação que utiliza o conceito arquitetural REST [TILKOV, 2008].

Para construção de um Web Service RESTful deve-se seguir alguns conceitos do REST, um dos conceitos é a tomada de decisão a partir da URI (*Uniform Resource Identifier* - Identificador Uniforme de Recurso), que juntamente com os métodos do HTTP, fornecem algum recurso para quem estiver solicitando a requisição. Os principais métodos, também conhecidos como verbo, do HTTP para a criação de um CRUD (*Create* - Criar, *Read* - Ler, *Update* - Atualizar, *Delete* - Deletar) estão listadas abaixo [WEBER, 2010].

- POST: utilizado como forma de envio de uma informação, seguindo a linha do CRUD o POST seria a letra C, representando assim o *create* (criar).

- GET: utilizado como forma de solicitar uma informação, no CRUD este método representaria a letra R, representando assim o *read* (leitura).

- PUT: utilizado como formata de atualizar uma informação, este método representaria letra U do CRUD, representando assim o *update* (atualizar).

- DELETE: utilizado como forma de remover uma informação, este método representaria a letra D do CRUD, representando assim o *delete* (deletar).

Uma vantagem em desenvolver um Web Service RESTful, é que pode utilizar o JSON (JavaScript Object Notation – Notação de Objetos JavaScript), que é uma forma mais leve que o XML (Extensible Markup Language – Linguagem Extensível de Marcação Genérica), fazendo com que os dados transmitidos fiquem menores. O JSON é um objeto já representado no javaScript, fazendo com que o objeto seja trabalhado na Web sem necessidade de realizar uma transformação antes [DIAS, 2016].

As principais características de um Web Services RESTful são [FIELDING, 2000]:

- *Client-Server* (Cliente Servidor): é a forma de separar o cliente do servidor, ou seja, o cliente fica responsável pela parte da experiência do usuário, interface, e o servidor responsável pela parte de persistência das informações no banco de dados, gerenciamento da informação, validações de regras de negócio, entre outros.

- *Stateless* (Sem estado): requisições do cliente são independentes, e a requisição realizada pelo cliente tem que conter todas as informações necessárias para que o servidor consiga processá-la.

- *Cacheable* (Armazenado em Cache): as informações que o cliente solicita são armazenadas no cache temporariamente, para que quando outro cliente solicitar a mesma informação, o servidor retorna a informação que está no cache.

- *Uniform Interface* (Interface Uniforme): são regras ou contratos para a troca de informação entre cliente e sevidor.

- *Layered System* (Sistema em camadas): o cliente não comunica diretamente com o servidor, as requisições do cliente voam para o intermediador e o intermediador é responsável em redirecionar a requisição para o servidor.

Para Roy Fielding uma características que um Web Service RESTful não pode faltar seria o HATEOAS (*Hypermedia as the Engine of Application State* – Hipermídia como Motor de Estado do Aplicativo), que é a forma como uma informação é fornecida, de modo que uma informação mostre o seu estado atual e mostre todas as opções para os estados futuros, com isso consiga "caminhar" em todas as opções disponíveis, formando assim uma "teia" de informação [DIAS, 2016].

Neste tópico foi abordado o que é um w*orkflow* assim como suas vantagens, também foram abordados os principais conceitos sobre Webservice RESTful, no tópico seguinte serão abordadas as ferramentas que auxiliam na construção de um w*orkflow* e de um Webservice RESTful.

3. Ferramenta de *workflow* e desenvolvimento de Web Service

Neste tópico será demonstrado um *framework* que auxilia no desenvolvimento de um Web Services RESTful e de um w*orkflow*. Serão demonstrados o *framework* OSWorkflow e o Spring Boot.

Todo conteúdo do sub-tópico referente a OSWorkflow foi baseado do livro OSWorkflow de Lazo (2014). Já o tópico referente ao *framework* Spring Boot foi baseado em sua documentação disponível em http://docs.spring.io/spring-boot/docs/2.0.0.BUILD-SNAPSHOT/reference/htmlsingle/.

3.1 OSWorkflow

OSWorkflow é um motor de *workflow*, *open source* (código aberto) baseado em Java. Ele possui implementações de todas as funções que um w*orkflow* precisa, também utiliza alguns projetos Java de código aberto, como o BeanShell, que é utilizado para processar scripts no fluxo de trabalho.

OSWorkflow possibilita trabalhar em container JEE (Java Enterprise Edition), incluindo container Servlet, ele foi projetado para ser o mais flexível possível, de modo a atender muitos requisitos. É recomendado desenvolver os fluxos utilizando arquivos XML, pois facilita o desenvolvimento.

O arquivo XML tem um DTD (*Document Type Definition* – Definição de Tipo de Documento) que permite a construção de fluxo com a sintaxe correta, na Figura 2 retirada do livro OSWorkflow demonstra a estrutura utilizando DTD para um modelo de w*orkflow*.

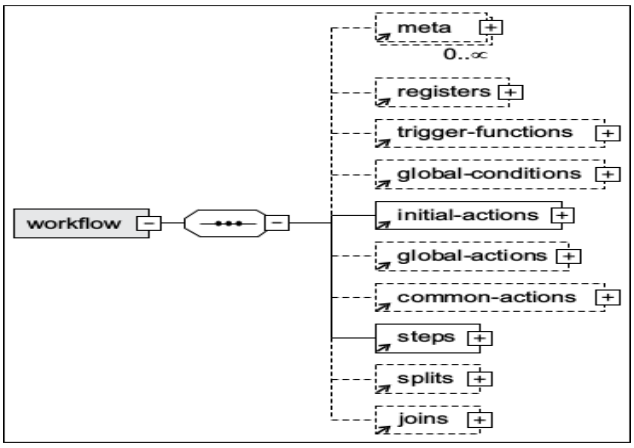

Figura 2 – Modelo DTD para construção de fluxo via XML do OSWorkflow

Um *initial-actions* (ações iniciais) é responsável por iniciar o fluxo, ele possui um grupo de elementos *action* (ação). Cada *action* possui *results* (resultados) que são responsáveis por enviar para outros *steps* (passos), com o auxilio de *conditions* (condições), por padrão um *results* possui um *uncoditional-result* que é executado caso nenhum *result* seja executado.

Cada *step* pode ter uma lógica específica, utilizando *pre-functions* ou *pos-functions*, com isso possibilita ao desenvolvedor, colocar lógica antes ou depois de um passo ser executado. Uma *action* pode ser executada automaticamente com o uso da propriedade *auto* igual a *true* (verdadeiro), por padrão a *action* não é executada automaticamente. O exemplo demonstrado na figura 3 representa um *step* responsável em enviar um e-mail para um funcionário.

```
<step id="4" name="Notifica Funcionário">
    <actions>
        <action id="4" name="Notifica Funcionário" finish="true" auto="true">
            <pre-funtions>
                <function type="class">
                    <arg name="class.name">
                        com.opensymphony.workflow.util.SendEmail </arg>
                    <arg name="to">destino@email.com</arg>
                    <arg name="from">hernane@email.com</arg>
                    <arg name="subject">TCC</arg>
                    <arg name="message">Sua mensagem é ${result}</arg>
                </function>
            </pre-functions>
            <results>
                <unconditional-result old-status="Finished"
                      old-status="Waiting" step="5" />
            </results>
        </action>
    </actions>
</step>
```

Figura 3 – Step de envio de e-mail a um funcionário

Observe, que no fragmento de código de um *workflow* exemplificado na figura 3, possui um "${result}", este valor representa uma chave de um mapa de valores (*map*). O OSWorkflow possui dois tipos de *maps*, o TransietVars e o PropertySet, sendo que o primeiro tem os valores armazenados em memória, com isso os valores salvos nele devem ser utilizados no step corrente, enquanto que o segundo possui os valores persistidos em banco de dados, com isso possibilitando utilizar seus valores em todo o fluxo.

OSWorkflow possui a funcionalidade de trabalhar com o fluxo todo em memória ou persistindo em banco de dados, cabe ao desenvolvedor escolher qual a opção melhor irá atender a regra de negócio.

3.2 *Framework* Spring Boot

O Spring Boot é um *framework* da família do Spring, ele utiliza as funcionalidades que o Spring já possuía, mas sua diferença, é que suas configurações podem ser realizadas nas classes do Java com uso de anotações, e não somente em arquivos XML.

O foco do Spring Boot é a produtividade, fazendo com que muitas configurações já venham pré definidas. Para utilizar os recursos que o Spring Boot oferece com auxilio do Maven, basta configurar o arquivo pom.xml conforme exemplificado no trecho de código da figura 4, e com isso o Maven irá baixar todas as dependências necessárias para realizar o famoso "Hello World".

```
//...
<parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.3.1.RELEASE</version>
</parent>
<dependencies>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
                <scope>compile</scope>
        </dependency>
        //...
</dependencies>
```

Figura 4 – Configuração do arquivo pom.xml do Maven para usar Spring Boot

Essa configuração do arquivo pom.xml indica que serão utilizadas as dependências de um projeto Spring Boot, e com isso será utilizado recursos de uma aplicação Web. A dependência spring-boot-starter-web possibilita baixar as dependências necessárias para a construção de um Web Service.

Com as dependências inseridas no projeto, a maioria das configurações poderá ser realizada com uso de anotações, a primeira a ser realizada é anotar a classe principal, para isso basta usar a anotação @SpringBootApplication e criar um método *main* conforme demonstra um trecho de código na figura 5.

```java
@SpringBootApplication
public class Boot
{
   public static void main(String[] args)
   {
      SpringApplication.run(Boot.class, args);
   }
}
```

Figura 5 – Classe principal do Spring Boot

Com o Spring Boot não precisa gerar um arquivo WAR (Web Application Archive) e depois colocar em um servidor Web, a dependência spring-boot-starter-web possui as dependências de um servidor (tomcat), com isso bastar rodar a classe principal e o servidor irá iniciar. Outra possibilidade, é gerar um JAR (Java Archive), e utilizar o java para rodar a aplicação.

Uma facilidade em usufruir do Spring Boot, é a forma como se fornece recursos, bastando criar uma classe e utilizar algumas anotações e a classe estará finalizada para fornecimento de recursos. Na figura 6 é exemplificado um trecho de código Java que demonstra uma forma de fornecimento de recursos para o cliente.

```java
@RestController
public class ClienteController {
  @Autowired
  ClienteRepository clienteRepository;
  @RequestMapping( value="/cliente", method=RequestMethod.GET )
  public ResponseEntity<List<Cliente>> buscarTodasClientes( ){
    List<Cliente> clientes = clienteRepository.findAll();
    if( clientes == null){
        return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }
    return new ResponseEntity<List<Cliente>>(clientes, HttpStatus.OK);
  }
  @RequestMapping( value="/cliente/{id}", method=RequestMethod.GET)
  public ResponseEntity<Cliente>
           buscarClientePorId(@PathVariable("id") Integer id ){
    Cliente cliente = clienteRepository.findOne(id);
    if( cliente == null){
        return new ResponseEntity<Cliente>(HttpStatus.NOT_FOUND);
    }
    return new ResponseEntity<Cliente>(cliente, HttpStatus.OK);
  }
}
```

Figura 6 – Exemplo de uma classe controladora de recursos

A figura 6 possui algumas anotações, a anotação @RestController permite criar um controlador para manipular requisições vindas do cliente. A anotação @RequestMapping permite fazer o mapeamento da URI. Uma outra anotação bastante utilizada é a @PathVariable que permite fazer um *bind* (vínculo) entre um valor passado na URI com uma variável definida, no exemplo da figura 6 é utilizado esse *bind* nos métodos de busca.

A seguir no tópico 4, será demonstrado um Webservice RESTful, desenvolvido com uso do Spring Boot, que utiliza o *framework* OSWorkflow para gerenciamento de um w*orkflow*.

4. Aplicando na prática um *workflow* com RESTful

Neste tópico será demonstrado na prática o uso do *framework* OSWorkflow em um Web Service RESTful, demonstrando assim a facilidade em fornecer recursos a outras aplicações por meio do protocolo HTTP. Parte do código referente ao estudo de caso deste artigo pode ser acessado através do *link* https://github.com/hernanerabelo/tcc.git.

4.1 Configurando beans para uso do OSWorkflow com Spring Boot

OSWorkflow possibilita ao desenvolvedor a escolha de persistir as informações do fluxo em banco de dados ou em memória. Na figura 7, é demonstrado um trecho de código da configurações de um *bean* (componente) no spring boot, para que as informações do fluxo sejam persistidas em banco de dados Mysql.

```
@Bean
public SpringJDBCWorkflowStore workflowStore() throws StoreException {
  SpringJDBCWorkflowStore springJDBCWorkflowStore = new SpringJDBCWorkflowStore();

  springJDBCWorkflowStore.setDatasource(dataSource);
  Map props = new HashMap();

  props.put("entry.sequence", "SELECT nextVal('seq_os_wfentry')");
  props.put("entry.table", "OS_WFENTRY");
  props.put("entry.id", "ID");
  props.put("entry.name", "NAME");
  props.put("entry.state", "STATE");
  props.put("step.sequence", "SELECT nextVal('seq_os_currentsteps')");
  props.put("history.table", "os_historystep");
  props.put("current.table", "OS_CURRENTSTEP");
  props.put("historyPrev.table", "OS_HISTORYSTEP_PREV");
  props.put("currentPrev.table", "OS_CURRENTSTEP_PREV");
  props.put("step.id", "ID");
  props.put("step.entryId", "ENTRY_ID");
  props.put("step.stepId", "STEP_ID");
  props.put("step.actionId", "ACTION_ID");
  props.put("step.owner", "OWNER");
  props.put("step.caller", "CALLER");
  props.put("step.startDate", "START_DATE");
  props.put("step.finishDate", "FINISH_DATE");
  props.put("step.dueDate", "DUE_DATE");
  props.put("step.status", "STATUS");
  props.put("step.previousId", "PREVIOUS_ID");

  props.put("step.sequence.increment", "INSERT INTO OS_STEPIDS (ID) values (null)" );
  props.put("step.sequence.retrieve", "SELECT max(ID) FROM OS_STEPIDS" );
  props.put("entry.sequence.increment", "INSERT INTO OS_ENTRYIDS (ID) values (null)");
  props.put("entry.sequence.retrieve", "SELECT max(ID) FROM OS_ENTRYIDS" );
  springJDBCWorkflowStore.setProps(props);
  springJDBCWorkflowStore.initSpring();
  return springJDBCWorkflowStore;
}
```

Figura 7 – Trecho de código referente a criação de um bean para persistência em banco de dados

No trecho de código exemplificado na figura 7, é demonstrado a configuração referente a um *bean* da classe SpringJDBCWorkflowStore, esta classe estende a classe MySQLWorkflowStore que vem do *framework* OSWorkflow, esta configuração é responsável em gerenciar a instância do *workflow*. A configuração de outro *bean* da classe JDBCWorkflowFactory faz-se necessário para gerenciamento do descritor, que é

o fluxo em si, ou seja, ele tem todas as informações do fluxo, como os *steps*, *joins*, *splits* entre outros.

Com os dois *beans* anteriores configurados, agora faz-se necessário criar um novo *bean* de configuração do OSWorkflow, este novo *bean* é referente a classe SpringConfiguration. Este *bean* possui dois atributos, o store e o factory, o primeiro deve fazer referencia ao *bean* referente a classe SpringJDBCWorkflowStore, e o segundo fazer referencia ao *bean* JDBCWorkflowFacotry.

Agora que os principais *beans* de configuração estão desenvolvidos, basta criar o *bean* que refere-se a classe Workflow do *framework* OSWorkflow. Este *bean* tem uma diferença entre os outros *beans* configurados anteriormente, o seu escopo é do tipo *prototype* (protótipo), já os outros por padrão são singleton (único). A diferença entre esses 2 tipos de escopos, é que o primeiro cria uma nova instância e o segundo utiliza a mesma instância.

As configurações realizadas anteriormente, são as principais para uso do OSWorkflow em uma aplicação spring. No próximo sub tópico será demonstrado alguns métodos do *workflow*.

4.2 Utilizando principais métodos do OSWorkflow na aplicação

Com o OSWorkflow configurado, basta utilizar alguns métodos do *workflow* para iniciar um novo fluxo, o método *initialize* espera receber três parâmetros, o primeiro o nome do fluxo, o segundo a *action* inicial e o terceiro e último um map, neste map os valores inseridos podem ser utilizados durante o fluxo. O método *canInitialize* recebe os mesmo parâmetros do *initialize*, porém ele retorna se é possível ou não inicializar aquele fluxo em questão.

Sempre que um fluxo é inicializado com uso do método *initialize*, ele retorna a instância do fluxo criado, com essa instância pode-se chegar ao detalhes daquele fluxo em questão. O uso do método *getAvailableActions* recebe a instância do fluxo inicializado e retorna quais *actions* o fluxo possui para ser executado no passo que parou, caso retorne um array vazio, significa que não possui mais actions a serem executadas, ou seja, o fluxo já finalizou.

Para restartar um fluxo que esteja parado, basta usar o método *doAction*, que recebe três parâmetros, a instância do fluxo, a ação a ser executada, e o map do mesmo estilo do *initialize*. Assim que restartado o fluxo, ele irá parar de executar quando encontrar uma *action* sem a opção automática.

Esses são os principais métodos do OSWorkflow para que possa criar e finalizar um fluxo. Na figura 8 será demonstrado um trecho de código do projeto deste artigo.

```
Long idExecution = workflow.initialize( osWorkflowdefs.getWfName(), 0, inputs);
workflowInstance.setIdExecution( idExecution );
workflowInstanceRepository.saveAndFlush( workflowInstance );
int[] actions = workflow.getAvailableActions( idExecution );
```

Figura 8 – Trecho de código que utiliza o método initialize para inicializar um novo fluxo

Este trecho de código exemplificado anteriormente, na figura 8, é responsável em inicializar um novo fluxo, depois persiste a instância criada e recupera as *actions* do passo que o fluxo parou daquela instância. No próximo sub tópico, será demonstrado como foi utilizado o RESTful no projeto, demonstrando assim alguns exemplos de como criar uma nova instância de um fluxo através de URI, assim como restartar um fluxo parado.

4.3 Utilizando RESTful na aplicação

Alguns conceitos do RESTful, como tomada de decisão a partir da URI e dos métodos do HTTP, estão exemplificados no trecho de código da figura 9.

```
@RequestMapping(value = "/workflow/instance", method = RequestMethod.POST)
public ResponseEntity createWorkflow(@RequestBody ServiceOrder serviceOrder) {
  WorkflowInstance workflowInstance = null;
  try {
    workflowInstance = workflowService.createWorkflow(serviceOrder);
  } catch (WorkflowException e) {
    e.printStackTrace();
  } catch (ServiceOrderException e) {
    e.setServiceOrder(serviceOrder);
    return new ResponseEntity<ServiceOrderException>(e, HttpStatus.BAD_REQUEST);
  }
  Resource<WorkflowInstance> workflowInstanceResource =
                             getWorkflowInstanceResource(workflowInstance);
  if (workflowInstanceResource == null) {
    return ResponseEntity.notFound().build();
  }
  return new ResponseEntity<Resource<WorkflowInstance>>(
              workflowInstanceResource, HttpStatus.CREATED);
}
```

Figura 9 – Trecho de código referente a fornecimento de recurso para criação de um novo fluxo

Neste trecho de código exemplificado na figura 9, é criado um novo fluxo sempre que for realizado uma requisição do tipo POST para a URI http://localhost:8080/workflow/instance, sendo que a URL (Uniform Resource Locator – Localizador Padrão de Recursos) é o endereço do servidor, neste caso está sendo localhost pois a chamada é local. Este método recebe um objeto no corpo do requisição POST, este objeto faz um *bind* com uma classe criada com nome ServiceOrder.

Uma classe de serviço foi desenvolvida para realizar o gerenciamento de criação de fluxo, e uma resposta utilizando ResponseEntity facilita o retorno ao cliente que solicitou o recurso. A classe Resource que é inserida através da dependência spring-boot-starter-hateoas, auxilia na construção de respostas no conceito HATEOAS, na figura 10 é exemplificado a resposta da criação de um novo fluxo utilizando esse conceito.

```
{
  "id": 24,
  "status": "Aguardando Assinatura do Funcionário",
  "createdDate": 1479086504540,
  "lastUpdatedDate": 1479086505341,
  "idExecution": 59,
  "_links": {
    "self": {
       "href": "http://localhost:9000/workflow/instance/24"
    },
    "serviceOrder": {
      "href": "http://localhost:9000/serviceOrder/23"
    },
    "historySteps": {
     "href": "http://localhost:9000/historyStep/workflow/instace/24"
    }
  }
}
```


Figura 10 – Formato de um JSON seguindo o conceito HATEOAS

Neste objeto JSON exemplificado na figura 10, são retornados no objeto criado, *links* que referenciam ao objeto criado, além de *links* para a ordem enviada para criação do fluxo, também possui um *link* para o histórico dos passos que o fluxo passou. Caso o cliente resolva escolher o link do historySteps ele receberá o seguinte objeto JSON exemplificado na figura 11.


```
[
  {
    "id": 102,
    "entry_id": 59,
    "step_id": 1,
    "action_id": 1,
    "owner": "System",
    "start_date": 1479086505000,
    "status": "Documento Assinado Pelo Funcionário","caller": "System",
    "links": [
      { "rel": "self",
        "href": "http://localhost:9000/historyStep/102"
      },
      { "rel": "workflowInstance",
        "href": "http://localhost:9000/workflow/instance/24"
      }
    ]
  }
]
```


Figura 11 - JSON no conceito HATEOAS referenciando um historyStep

Neste exemplo de JSON exemplificado na figura 11, além do objeto HistoryStep, ele possui links, um link para o objeto em questão e outro para a instância do fluxo, caso o cliente escolha o link do workflow/instance, o mesmo retornará um objeto JSON exemplificado anteriormente na figura 10. Deste modo através de um link, o cliente consegue chegar a outros resultados, formando assim uma “teia” de links.

4.4 Finalizando a aplicação

Com a aplicação finalizada, basta rodar o método principal para iniciar a aplicação, ou gerar um JAR e fornecer ao servidor para que a aplicação seja iniciada com o uso do Java. Com a aplicação rodando, basta fornecer os dados referente aquela requisição ao

cliente que irá consumir os recursos. Com isso, o cliente estará apto a criar novos fluxos e ter um controle sobre ele.

5. Conclusão

Como foi abordado neste artigo, pode-se concluir que um Web Service RESTful desenvolvido com o auxílio do Spring Boot, aprimora o fornecimento de recursos, tento menor gasto no tempo de desenvolvimento da aplicação. Já o OSWorkflow, auxilia a aplicação a gerenciar um *workflow*, bastando apenas que o desenvolvedor crie um fluxo em XML.

Como o OSWorkflow necessita que seus fluxos sejam desenvolvidos em XML, é preciso que o desenvolvedor tenha um conhecimento sobre a DTD do OSWorkflow. Também é necessário o conhecimento sobre a linguagem de programação Java, uma vez que o OSWorkflow possibilita criar classes Java e inserir no contexto do fluxo.

Com isso, como a aplicação do *workflow* está centralizada no gerenciamento do processo, o mesmo pode comunicar-se com outras aplicações, e vice-versa, bastando apenas passar as informações necessárias para que a comunicação seja realizada, como por exemplo a URI do recurso a ser fornecido.

Com esse trabalho, fica claro a necessidade de uma ferramenta de orquestração de processos integrada a um Web Service, concluindo assim, que empresas que utilizem tais ferramentas, tem seus processos automatizados, fazendo com que tenham um ganho de tempo durante o processo. Também fica evidente, que com uso de ferramentas de orquestração de processos, todos os passos são facilmente alterados, e com isso dando uma maior flexibilidade ao processo.

5.1 Trabalhos futuros

Para trabalhos futuros, sugere-se a implementação da aplicação desenvolvida neste trabalho em um ambiente corporativo, a fim de avaliar a eficiência da aplicação de orquestração desenvolvida neste projeto.

Outro trabalho futuro que seria bastante relevante, diz respeito a criar novos métodos para que o cliente tenha um controle maior sobre o fluxo. Também seria útil a criação de uma classe de serviço referente as requisições HTTP, a fim de que o servidor de orquestração envie requisição para outros servidores.

Também seria necessário a implementação referente a controle de acesso, para que seja bloqueado requisições de clientes sem acesso ao servidor e com isso aumentando a segurança da aplicação.

6. Referências

DIAS, Emílio. **Desmistificando REST com Java**. AlgarWorks Softwares, Treinamento e Serviços Ltda. 2016. p. 10-27.

FIELDING, Roy Thomas. **Architectural Styles and the Design of Network-based Software Architectures**. Doctoral dissertation, University of California, Irvine, 2000.

LAZO, Diego Adrian Naya. **OSWorkflow. A guide for Java developers and architects to integrating open-source Business Process Management**. Packt Publishing. 2007.

NOGUEIRA, Rhaíssa. **Introdução ao Business Process Modeling Notation**. Disponível em: <http://www.devmedia.com.br/introducao-ao-business-process-modeling-notation-bpmn/29892>. Acesso em: 1 de Outubro de 2016.

TILKOV, Stefan. **Uma rápida Introdução ao REST.** 2008. Disponível em: <https://www.infoq.com/br/articles/rest-introduction>. Acesso em: 29 Setembro 2016.

WEBB, Phillip et al. **Spring Boot Reference Guide**. 2013. Disponível em: <http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>. Acesso em: 11 de Outubro de 2016.

WEBER, Jim. et al. **REST in Pratice: Hypermedia and Systems Architecture**. O'Reilly Media. 2010.

W3SHOLLS. **HTTP Methods: GET vs. POST**. Disponível em: <http://www.w3schools.com/TAgs/ref_httpmethods.asp>. Acesso em: 11 Outubro 2016.