UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

EDIÇÃO GRÁFICA DE FLUXOS DE TRABALHO BASEADA EM BPMN COM EXECUTOR BASEADO EM SOA PARA MÁQUINAS DE *WORKFLOW*

FÁBIO ISENSEE

BLUMENAU
2011

2011/1-18

FÁBIO ISENSEE

EDIÇÃO GRÁFICA DE FLUXOS DE TRABALHO BASEADA EM BPMN COM EXECUTOR BASEADO EM SOA PARA MÁQUINAS DE *WORKFLOW*

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Marcel Hugo, Mestre - Orientador

BLUMENAU
2011

2011/1-18

EDIÇÃO GRÁFICA DE FLUXOS DE TRABALHO BASEADA EM BPMN COM EXECUTOR BASEADO EM SOA PARA MÁQUINAS DE *WORKFLOW*

Por

FÁBIO ISENSEE

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Marcel Hugo, Mestre – Orientador, FURB
Membro:	Prof. Everaldo Artur Grahl, Mestre – FURB
Membro:	Prof. Jacques Robert Heckmann, Mestre – FURB

Blumenau, 04 de julho de 2011

Dedico este trabalho aos amigos que me auxiliaram na realização deste e a todos os interessados na automatização de fluxos de trabalho utilizando máquinas de *workflow*.

AGRADECIMENTOS

À minha família, que mesmo longe, sempre acreditou em mim e torceu pelo meu sucesso.

Aos meus amigos, por terem continuado meus amigos mesmo eu tendo sumido durante o desenvolvimento deste trabalho.

Ao meu orientador, Marcel Hugo, por ter me auxiliado e acreditado na conclusão deste trabalho.

Ao meu amigo e colega de trabalho, Carlos Augusto Grahl, por ter contribuído com várias idéias e materiais encontrados por ele.

Aos meus amigos e também colegas de trabalho, Cristian Rodrigo Santin e Glauco da Silva Rocha, por terem auxiliado na escolha deste tema.

O único lugar onde sucesso vem antes do trabalho é no dicionário.

Albert Einstein

RESUMO

Este trabalho apresenta a especificação, a implementação e um estudo de caso referente a um conjunto de componentes que possibilitam a automatização de fluxos de trabalho utilizando máquinas de *workflow*. O fluxo a ser automatizado é modelado em um editor com elementos baseados em BPMN, o qual é implementado como um *applet* Java para que possa ser acessado por um navegador *web*. Depois de modelado, o fluxo é executado por um *framework* desenvolvido em Java, o qual é responsável por disparar a execução, tratar os desvios e manter o sincronismo das atividades definidas. O *framework* disponibiliza uma API para a criação de extensões, possibilitando a integração em nível de código fonte com outros sistemas e ferramentas. Cada extensão é responsável pelo processamento da atividade para a qual é destinada. Também é implementada uma extensão baseada em SOA para integração com máquinas de *workflow* e outros sistemas através de *web services*, utilizando o protocolo SOAP. A persistência dos dados é feita utilizando o padrão de projetos DAO, nativamente estendido para armazenamento em banco de dados Microsoft SQL *Server*.

Palavras-chave: *Workflow*. BPMN. SOA. SOAP. *Web service*.

ABSTRACT

This work shows the specification, the implementation and a case study relating to a set of components that allow the workflows automation using workflow engines. The flow that will be automated is modeled by a BPMN element based editor, which is implemented as a Java applet so it is accessible by web browsers. After modeled, the flow is performed by a framework developed in Java, which is responsible for start running, catch de switches and keep the defined activity synchronism. The framework provides an API that allows the creation of extensions, allowing a source level integration to other systems and tools. Each activity is responsible for processing the activities it was target for. It also implements a SOA based extension to integrate with workflow engines and other systems over web services based on SOAP protocol. The data persistence is performed based on DAO design pattern, originally extended to storage in a Microsoft SQL Server database.

Key-words: Workflow. BPMN. SOA. SOAP. Web service.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API – *Application Programming Interface*

AWT – *Abstract Windowing Toolkit*

BAM – *Business Activity Monitoring*

BPD – *Business Process Diagram*

BPEL – *Business Process Execution Language*

BPMI – *Business Process Management Initiative*

BPMN – *Business Process Modeling Notation*

DAO – *Data Access Object*

EE – *Enterprise Edition*

HTTP – *HyperText Transfer Protocol*

IDE – *Integrated Development Environment*

JBI – Java *Business Integration*

JDBC – Java *DataBase Connectivity*

JDK – Java *Development Kit*

JRE – Java *Runtime Environment*

JVM – Java *Virtual Machine*

JSP – Java *Server Pages*

NMR – *Normalized Message Router*

RF – Requisito Funcional

RNF – Requisito Não Funcional

SGBD – Sistema Gerenciador de Banco de Dados

SOA – *Service Oriented Architecture*

SOAP – *Simple Object Access Protocol*

SQL – *Structured Query Language*

TI – Tecnologia da Informação

UML – *Unified Modeling Language*

W3C – *World Wide Web Consortium*

WSDL – *Web Service Definition Language*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Ao acompanhar a evolução da informática e dos sistemas de automação, é perceptível a necessidade crescente que as empresas têm de melhorar continuamente seu funcionamento e seus resultados, otimizando e automatizando seus processos de negócio, diminuindo o desperdício de tempo e riscos trazidos com o manuseio de documentos em papel.

Uma das formas mais modernas de automatizar os processos de negócio é com o uso de máquinas de *workflow*. Segundo Monteiro (2007), o conceito de *workflow* geralmente é discutido no contexto de software como um modelo que irá ajudar a gerenciar informações com políticas ou regras de controle do fluxo de informações e/ou tarefas. Ao acompanhar a evolução das ferramentas disponíveis no mercado, é comum encontrar soluções onde uma máquina de *workflow* é desenvolvida para automatizar processos de negócio utilizando recursos já disponíveis, como sistemas legados ou sistemas de terceiros. A máquina de *workflow* é responsável por controlar o fluxo de execução dos processos automatizados, disponibilizar a interface para os usuários iniciarem a execução dos processos e consultarem suas tarefas e prover recursos para administração e monitoramento dos processos. O processamento das atividades em si é delegado aos sistemas que são utilizados pela máquina de *workflow*, os quais podem ou não interagir com o usuário final para tratá-la.

Com o objetivo de disponibilizar uma notação de fácil compreensão para todos os usuários do negócio, desde os analistas de negócio, passando pelos desenvolvedores, até as pessoas que vão gerenciar e monitorar os processos, a *Business Process Management Initiative* (BPMI), em um esforço de mais de dois anos, especificou a *Business Process Modeling Notation* (BPMN) (WHITE, 2004, p. 1). Atualmente, com uma simples pesquisa no Google, pode-se constatar que a BPMN é um padrão muito utilizado no mercado e vem ganhando cada vez mais espaço entre profissionais de consultoria especializada, contando com uma considerável quantidade de ferramentas que possibilitam a criação de *Business Process Diagram* (BPD).

Diante do exposto, a proposta deste trabalho foi juntar os conceitos de modelagem de processos de negócio utilizando BPMN e automatização com máquinas de *workflow* em uma ferramenta que possibilite desde a criação do BPD até a publicação e execução do processo. A ligação com a camada de execução é feita dentro dos próprios elementos da BPMN contidos no modelo gráfico elaborado pelo analista de negócio, sem a necessidade de uma linguagem específica para realização (implementação propriamente dita) de todo o modelo.

O executor foi desenvolvido como um *framework*, o qual disponibiliza uma *Application Programming Interface* (API) que permite sua extensão para integração em nível de código Java com outros sistemas e ferramentas. Cada extensão é responsável pelo processamento da atividade para a qual foi desenvolvida. Para que o executor seja capaz de se comunicar com diferentes sistemas, legados ou não, independente de linguagem e plataforma de desenvolvimento, foi construída uma extensão baseada em *Service Oriented Architecture* (SOA) utilizando protocolo *Simple Object Access Protocol* (SOAP), o qual é regulamentado pela W3C (WORLD WIDE WEB CONSORTIUM, 2007). Segundo Sampaio (2006, p. 15), "SOA visa criar componentes de granularidade grossa, chamados serviços, que requerem baixo acoplamento com seus clientes."

Com a finalidade de facilitar a extensão da ferramenta com o desenvolvimento de rotinas de monitoramento e administração dos processos em execução, o editor foi desenvolvido como um aplicativo *web* para possibilitar a reutilização das rotinas de renderização[1] dos elementos da BPMN.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho foi desenvolver uma ferramenta que facilite o processo de automatização de fluxos de trabalho através de um editor BPMN via *web* e de um executor de baixo acoplamento.

Os objetivos específicos do trabalho foram:

a) criar um editor de processos de negócio com elementos baseados em BPMN para facilitar a adaptação de pessoas já familiarizadas com esta notação;
b) disponibilizar o editor para acesso *web* para permitir a reutilização dos elementos gráficos ao desenvolver rotinas de administração e monitoramento remoto;
c) criar um executor dos processos que se deseja automatizar para integração com máquinas de *workflow*;
d) utilizar SOA para permitir um baixo acoplamento a alguma arquitetura proprietária e possibilitar a integração com sistemas, legados ou não, para obtenção de informações e para a execução de tarefas humanas.

[1] Renderização é o processo de geração de uma imagem a partir de um modelo, o qual é composto por objetos definidos por uma linguagem ou estrutura de dados.

1.2 ESTRUTURA DO TRABALHO

Além do texto já apresentado, este trabalho é composto por mais três capítulos. O segundo capítulo apresenta a fundamentação teórica que embasa a ferramenta desenvolvida. O terceiro contém os requisitos e a especificação que definiram a forma como a ferramenta foi construída, além dos resultados obtidos com um estudo de caso. Por fim, o quarto capítulo apresenta as conclusões do trabalho considerando os resultados obtidos.

2 FUNDAMENTAÇÃO TEÓRICA

A seguir, na seção 2.1, explica-se o que é BPMN, qual seu objetivo e as categorias nas quais ela é dividida. A seção 2.2 comenta o que é *framework* e como ele é utilizado no desenvolvimento de ferramentas e sistemas. A seção 2.3 descreve o que é SOA e qual sua relação com SOAP e *web services*. Na seção 2.4, comenta-se sobre a aplicação de máquinas de *workflow*. Na seção 2.5 está descrito o conceito de *applet* e sua utilização com Java 2D[2]. Por fim, na seção 2.6, é comentado sobre as ferramentas *Enterprise Architect*, BPEL *Designer* e BPEL *Service Engine*, além do trabalho de Reinert (2006).

2.1 *BUSINESS PROCESS MODELING NOTATION* (BPMN)

A BPMN é uma notação muito rica e intuitiva, que diminui as distâncias entre o mapeamento dos processos nas áreas de negócio e a adaptação técnica destes processos na área de TI (BITENCOURT, 2007, p. 1).

> Existem muitos softwares para BPM com notações diferentes e proprietárias. Ao contrário disso, a BPMN é **simples** e ao mesmo tempo muito **poderosa** e, principalmente, um **padrão de mercado** sem a necessidade de ficar preso a um determinado fornecedor. (BITENCOURT, 2007, p. 1, grifo nosso).

Segundo o Object Management Group (2010, p. 27-28), com o objetivo de criar um modelo simples e inteligível de modelagem de processos, e ao mesmo tempo tratar toda complexidade destes mesmos processos de negócio, a BPMN está dividida em cinco categorias básicas de elementos:

a) *flow objects*: contém os principais elementos para a definição do comportamento de um processo de negócio, denominados *events*, *activities* e *gateways*;
b) *data*: é representada por quatro elementos: *data objects*, *data inputs*, *data outputs* e *data stores*;
c) *connecting objects*: representa os quatro objetos que permitem conectar os elementos do fluxo: *sequence flows*, *message flows*, *associations* e *data associations*;

[2] Termo utilizado na área de computação gráfica para designar um espaço plano, o qual é representado em duas dimensões.

d) *swimlanes*: contém os dois elementos de agrupamento de elementos primários: *pools* e *lanes*;

e) *artifacts*: representa os elementos *group* e *text annotation*, os quais permitem prover informações adicionais sobre o processo.

A Figura 1 ilustra um exemplo de processo modelado com BPMN, onde alguns elementos foram rotulados conforme sua categoria:

a) A: *flow objects*;

b) B: *data*;

c) C: *connecting objects*;

d) D: *swimlane*;

e) E: *artifact*.

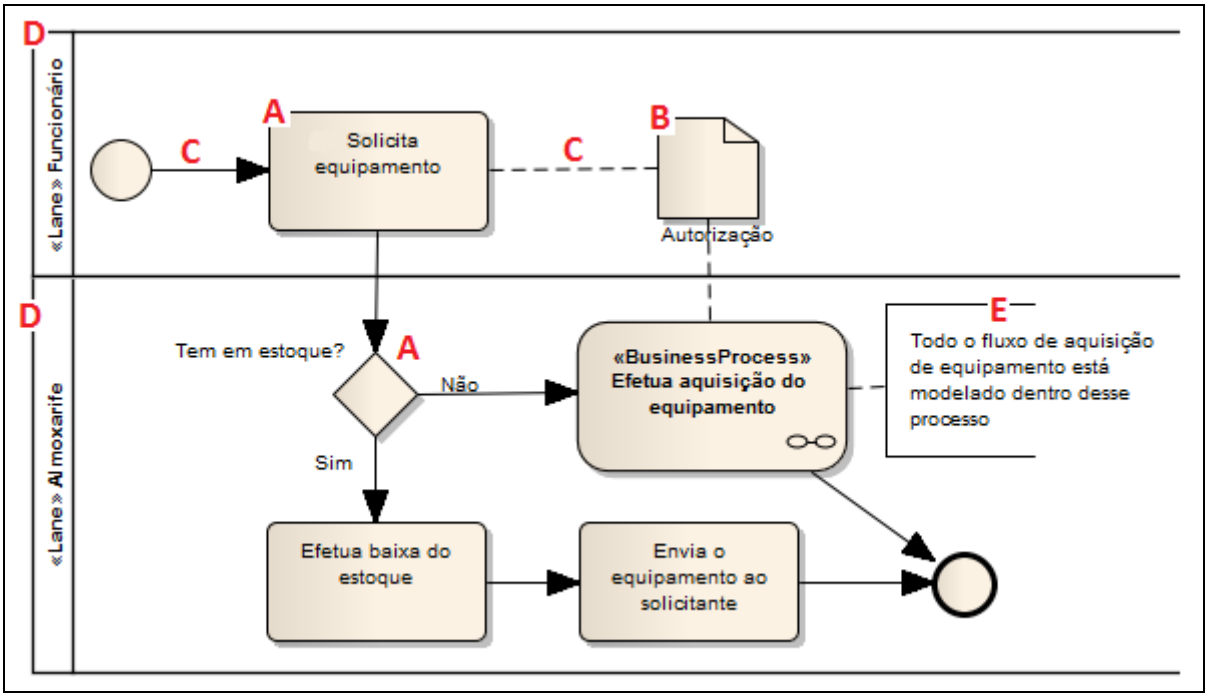


Figura 1 – Exemplo de processo modelado com BPMN

A primeira versão da especificação da BPMN foi lançada ao público em maio de 2004 após mais de dois anos de esforço do BPMI (WHITE, 2004, p. 1). A versão disponível atualmente da BPMN é a 2.0, cuja última revisão foi concluída em janeiro de 2011 (OBJECT MANAGEMENT GROUP, 2011).

2.2 *FRAMEWORK*

Segundo Fayad e Schmidt (1997), *framework* orientado a objeto é uma tecnologia promissora para modelagem e implementação de software visando reduzir seu custo e melhorar sua qualidade. Um *framework* é uma aplicação reutilizável e semi-completa que pode ser especializada para produzir aplicações personalizadas. Diferentemente das técnicas anteriores de orientação a objetos, as quais são baseadas em classes e bibliotecas, os *frameworks* são destinados a unidades de negócio particulares (como processamento de dados ou comunicações de celulares) e domínios de aplicações (como interfaces de usuários ou sistemas aviônicos de tempo real).

Ainda segundo Fayad e Schmidt (1997), os benefícios primários dos *frameworks* originam-se na modularidade, reusabilidade, extensibilidade e inversão de controle que eles provêem aos desenvolvedores. A modularidade é aperfeiçoada ao encapsular detalhes das implementações por trás de interfaces estáveis. Esta modularidade aperfeiçoa a qualidade do software ao concentrar o impacto de mudanças de modelagem e implementação, diminuindo o esforço de entendimento e manutenção. As interfaces estáveis melhoram a reusabilidade através da definição de componentes genéricos que podem ser reutilizados para criar novas aplicações. A disponibilização de métodos específicos para estender as funcionalidades das interfaces estáveis aperfeiçoam a extensibilidade, a qual é essencial para garantir o desenvolvimento de novas aplicações e recursos em tempo adequado.

2.3 SOA, SOAP E *WEB SERVICES*

Service Oriented Architecture (SOA) é um novo paradigma de desenvolvimento de aplicações cujo objetivo é criar módulos funcionais de granularidade grossa, chamados de serviços, com baixo acoplamento e permitindo a reutilização de código (SAMPAIO, 2006, p. 14-15). Segundo Sampaio (2006, p. 15), o serviço deve executar unidades completas de trabalho, recebendo requisições e respondendo de forma a ocultar os detalhes do processamento.

Web service é uma “encarnação” de SOA que favorece muito a criação de componentes fracamente acoplados de granulação grossa (SAMPAIO, 2006, p. 19). Ainda

segundo Sampaio (2006, p. 38), normalmente os *web services* utilizam o protocolo SOAP para troca de mensagens.

O *Simple Object Access Protocol* (SOAP) foi criado, inicialmente, para possibilitar a invocação remota de métodos pela *internet*, utilizando-se de duas tecnologias já existentes, *eXtensible Markup Language* (XML) e *Transfer Protocol* (HTTP). O XML é a forma mais conhecida, flexível e padronizada de troca de dados, enquanto o HTTP é um protocolo simples, aberto e bem difundido (SAMPAIO, 2006, p. 25). Atualmente o SOAP é um protocolo completo de troca de mensagens XML regulamentado como padrão pela W3C (WORLD WIDE WEB CONSORTIUM, 2007).

2.4 MÁQUINAS DE *WORKFLOW*

Tendo serviços bem definidos, conforme o conceito de SOA, é possível organizá-los em vários fluxos através de um programa que os invoque no momento apropriado, certificando-se que suas precondições e pós-condições estejam atendidas. É possível ainda organizá-los de forma diferente ou colocar alguns destes serviços em um fluxo diferente. Isso é chamado de orquestração (SAMPAIO, 2006, p. 17).

> A orquestração é composta por um fluxo de etapas, com verificações de pré e pós-condições, e **um coordenador, responsável por dar andamento ao fluxo**. O cliente se comunica com o coordenador e efetua a macro-solicitação, e o coordenador inicia o fluxo, invocando e verificando todas as etapas necessárias. Cada etapa invoca um serviço, que é oferecido por um *Provider*. Desta maneira, podemos mudar a ordem das etapas, acrescentar outras etapas, mudar os critérios de verificação ou mesmo criar outros fluxos sem alterar o código dos serviços. (SAMPAIO, 2006, p. 18, grifo nosso).

O papel de coordenação de orquestração pode ser realizado por uma máquina de *workflow* que, segundo Han e Kim (2001, p. 1), são sistemas designados a ajudar organizações a coordenar, monitorar e executar suas várias atividades em um ambiente de trabalho distribuído.

Segundo Wulong (1998), “fluxo de trabalho” é um termo utilizado para definir tarefas, passos procedurais, organizações ou pessoas envolvidas, informações de entrada e saída obrigatórias e ferramentas necessárias para cada passo em um processo de negócio. A automatização de fluxos de trabalho possibilita que empresas criem modelos e depois os utilizem como forma de gerenciar e forçar um tratamento consistente do trabalho. Ainda segundo Wulong (1998), uma máquina de *workflow* é o componente do programa de

automatização de fluxos de trabalho que conhece todos os procedimentos, os passos em um procedimento e as regras para cada passo. É a máquina de *workflow* que determina quando o processo está pronto para ir para o próximo passo.

2.5 *APPLET* E JAVA 2D

Applet é um programa Java que pode ser embarcado em páginas HTML e rodar em navegadores *web* que suportam Java, como Mozilla e *Internet Explorer*. É utilizado para tornar as páginas *web* mais dinâmicas e divertidas (ROSE INDIA TECHNOLOGIES PVT. LTD, 2007). Ainda segundo esta definição, devido aos *applets* serem projetados para executar remotamente em um navegador, eles têm restrições, como não poder acessar recursos do sistema do computador local.

As vantagens dos *applets* são (ROSE INDIA TECHNOLOGIES PVT. LTD, 2007):

a) são multi-plataforma, podendo ser executados em Windows, Linux e Mac OS, entre outros;
b) funcionam em qualquer versão da extensão Java;
c) são executados em um contexto isolado, então o usuário não precisa confiar no seu código, fazendo com que ele funcione sem confirmação de segurança;
d) são suportados pela maioria dos navegadores *web*;
e) são mantidos em *cache* pela maioria dos navegadores *web*;
f) pode ter acesso completo à máquina se o usuário permitir.

As desvantagens dos *applets* são (ROSE INDIA TECHNOLOGIES PVT. LTD, 2007):

a) é necessário uma extensão do Java para executá-los;
b) necessita da Java *Virtual Machine* (JVM), então toma um tempo significativo na primeira utilização;
c) é difícil construir boas telas em *applet* se comparadas à tecnologia HTML.

O Java 2D é uma API que provê aos programas Java as habilidades de gráficos bidimensionais, texto e desenho de imagens através da extensão do *Abstract Windowing Toolkit* (AWT). É um *framework* de renderização repleto de recursos para o desenvolvimento de telas ricas, programas de desenho sofisticados e editores de imagens (ORACLE, 2011). Ainda segundo Oracle (2011), o Java 2D provê as seguintes habilidades:

a) um modelo de renderização uniforme para dispositivos de visualização e

impressoras;

b) uma ampla gama de primitivas geométricas como curvas, retângulos e elipses, bem como mecanismos para desenhar qualquer forma geométrica;
c) mecanismos para detectar cliques sobre formas, textos e imagens;
d) controle sobre como objetos sobrepostos são renderizados;
e) suporte a cores aperfeiçoado, que possibilita o gerenciamento das cores;
f) suporte à impressão de documentos complexos;
g) controle de qualidade da renderização através de sugestões de renderização.

2.6 TRABALHOS CORRELATOS

Existem algumas ferramentas de mercado que possibilitam a modelagem de processos com BPMN, dentre as quais pode-se citar a ferramenta *Enterprise Architect* (SPARX SYSTEMS, 2010). Outras disponibilizam recursos para automatização destes processos, como as ferramentas BPEL *Designer* e BPEL *Service Engine* da Sun Microsystems (2009). Na Universidade Regional de Blumenau (FURB) é possível encontrar trabalhos voltados para a área de *workflow*, como o sistema desenvolvido por Reinert (2006).

2.6.1 *Enterprise Architect*

A ferramenta *Enterprise Architect* foi desenvolvida pela Sparx Systems. Atualmente, na sua oitava versão, disponibiliza recursos que permitem fazer especificações de análises, projetos, implementações, testes e modelos de manutenção utilizando UML, BPMN e outros padrões abertos (SPARX SYSTEMS, 2010). As versões posteriores a 7.1 do *Enterprise Architect* suportam a versão 1.1 da BPMN e do BPEL, além de possibilitar utilizar a BPMN 1.0 por compatibilidade (SPARX SYSTEMS, 2004).

Segundo Sparx Systems (2004), o *Enterprise Architect* possibilita exportar processos modelados em BPMN 1.1 para BPEL, com as seguintes restrições impostas ao modelo:

a) usar os elementos da BPMN 1.1 da caixa de ferramentas para modelagem de BPEL;
b) todos os processos e sub-processos de BPEL devem iniciar com um *start event* e

terminar com um *end event*;

c) um *start event* ou um *end event* não podem ser anexados ao limite de um sub-processo;

d) loops de *sequence flows* não são suportados. Apenas loop de *activity*. Todos os *sequence flows* devem fluir para baixo e não para cima;

e) o BPEL não suporta mapeamento de *intermediate events* com gatilhos múltiplos;

f) o BPEL não suporta mapeamento de múltiplas instâncias paralelas de loops;

g) o BPEL não suporta mapeamento de sub-processos independentes.

A Figura 2 demonstra a tela principal do Enterprise Architect no momento da criação do modelo de um processo de negócio em BPMN. Nesta tela destaca-se, além da área de edição do modelo localizada no centro, a caixa de ferramentas com os componentes da BPMN, à esquerda, e o navegador de componentes do projeto à direita.

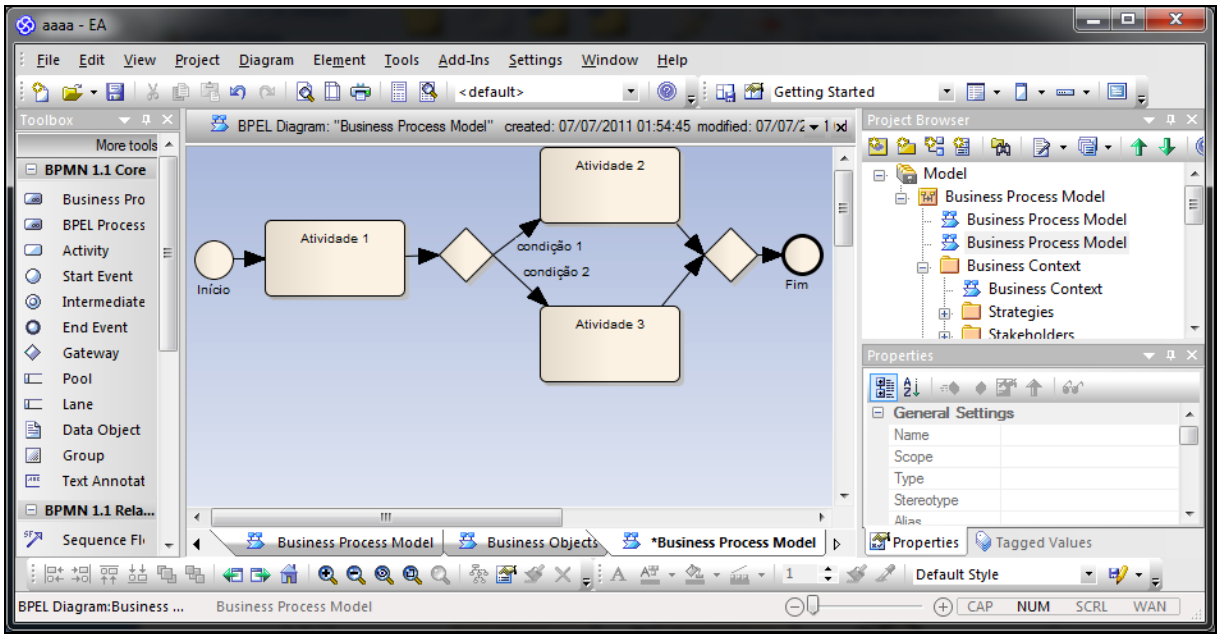


Figura 2 – Tela principal do *Enterprise Architect*

2.6.2 Sun BPEL *Designer* e BPEL *Service Engine*

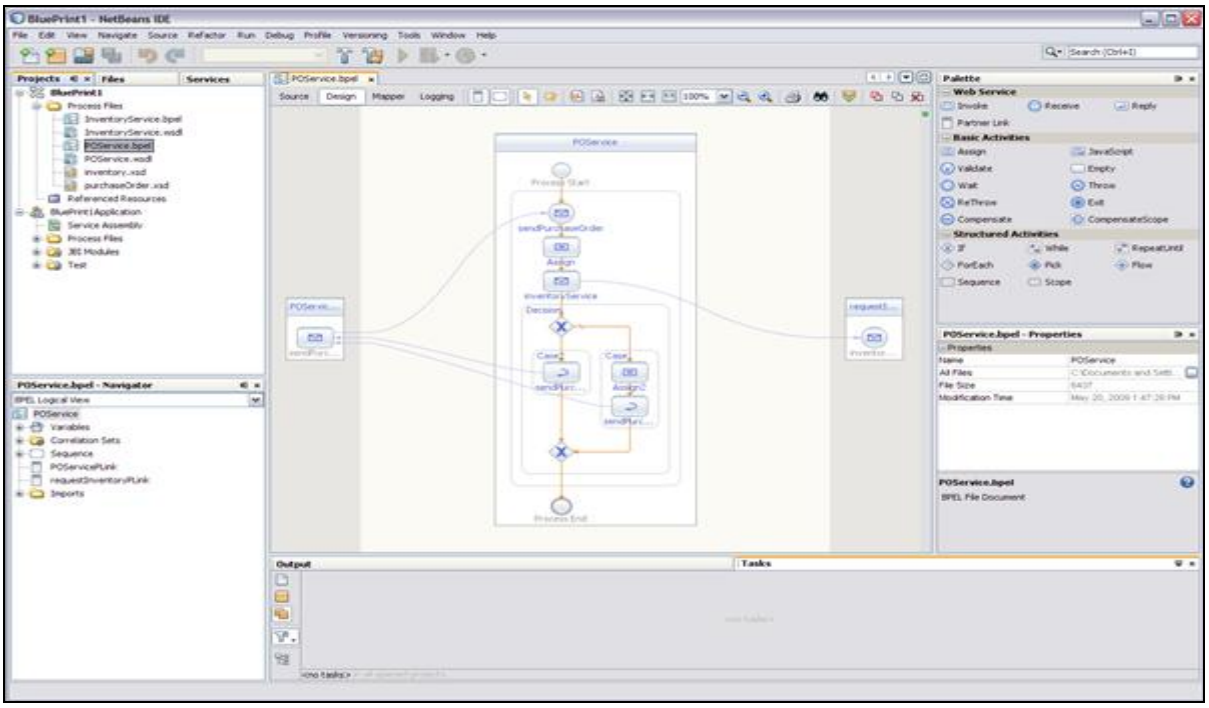
O BPEL *Designer* é um editor gráfico que possibilita criar e editar processos BPEL facilmente e depois fazer *deploy* deles para o BPEL *Service Engine*. Além disso, contém recursos que permitem executar estes processos em modo de depuração ou teste (SUN MICROSYSTENS, 2009, p. 7).

O BPEL *Service Engine* é um componente, implementado conforme a especificação

Java *Business Integration* (JBI), que provê serviços para a execução de processos de negócio desenvolvidos em BPEL (SUN MICROSYSTENS, 2009, p. 7). Processos de negócio tipicamente envolvem a troca de mensagens entre o processo e outros *web services*, conhecidos como *partner services*. No BPEL *Service Engine*, esta troca de mensagens é feita encapsulando esta mensagem conforme a *Web Service Definition Language* (WSDL), versão 1.1, a qual é transportada através de um JBI *Normalized Message Router* (NMR) que, por sua vez, interage com os *web services* externos (SUN MICROSYSTENS, 2009, p. 10).

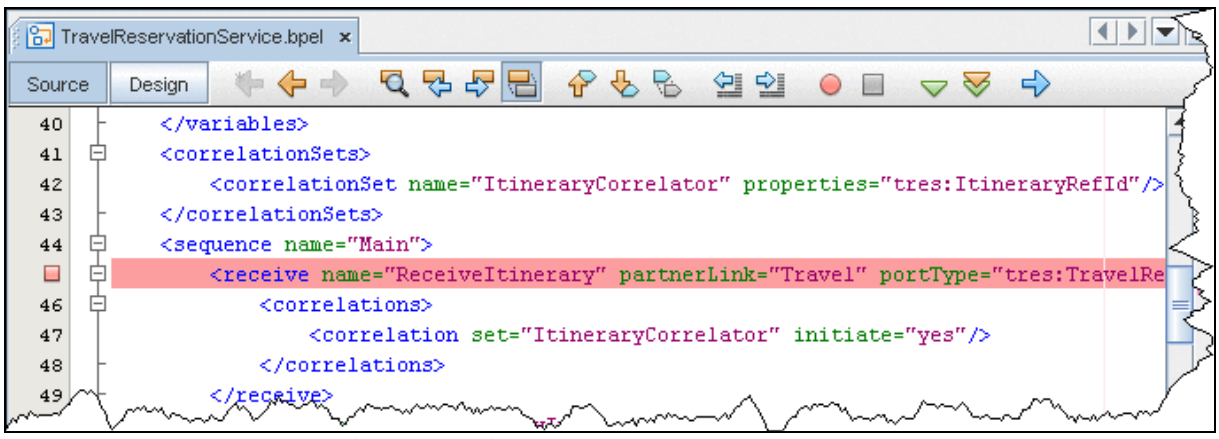
A Fonte: SUN MICROSYSTENS (2009, p. 26).

Figura 3 mostra a representação gráfica de um arquivo BPEL aberto na tela principal do BPEL *Designer*. Na Figura 4 pode-se ver a codificação de um arquivo BPEL no momento de sua depuração.



Fonte: SUN MICROSYSTENS (2009, p. 26).

Figura 3 – Tela principal do BPEL *Designer*



Fonte: SUN MICROSYSTENS (2009, p. 136).

Figura 4 – Tela de depuração do BPEL *Designer*

2.6.3 Sistema desenvolvido por Reinert (2006)

Reinert (2006) utiliza-se dos conceitos de *workflow* para a automatização do processo de desenvolvimento de software nas empresas, partindo da necessidade de um processo bem definido que contemple todas as etapas do seu ciclo de vida. Neste trabalho foi desenvolvida uma ferramenta dividida em dois módulos:

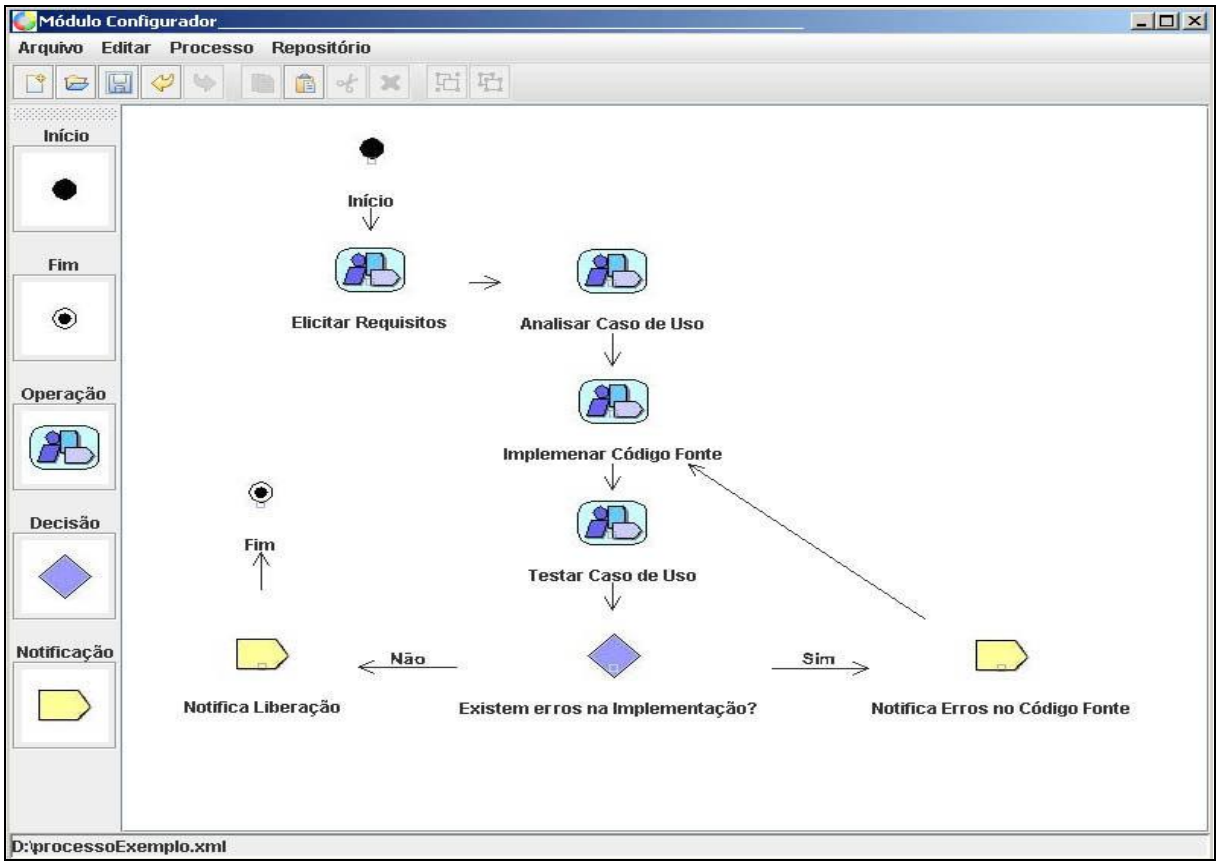
a) módulo de configuração: aplicativo *desktop* onde o administrador cadastra os componentes do processo, como as disciplinas, os papéis, as equipes, etc. Possui recursos que permitem a modelagem gráfica do processo de desenvolvimento de software na forma de um fluxograma;
b) módulo de execução: aplicativo responsável pela execução do processo em si, mantendo o relacionamento entre as atividades e os participantes, conforme definido no módulo de configuração. Disponibilizado para acesso *web*, permite iniciar novos processos, listar as atividades pendentes e alterar os atributos dos artefatos atrelados a cada atividade.

Segundo Reinert (2006, p. 72), a ferramenta desenvolvida em seu trabalho possui algumas limitações, como a impossibilidade de criar processos em vários subníveis e a impossibilidade de utilizar paralelismo na modelagem das atividades do processo. Ainda segundo o autor, a ferramenta tem potencial para modelagem de processos genéricos, e não apenas processos de desenvolvimento de software.

A ferramenta de Reinert foi implementada utilizando a linguagem Java com base na arquitetura Java EE, além das seguintes tecnologias (Reinert, 2006, p. 41 - 47):

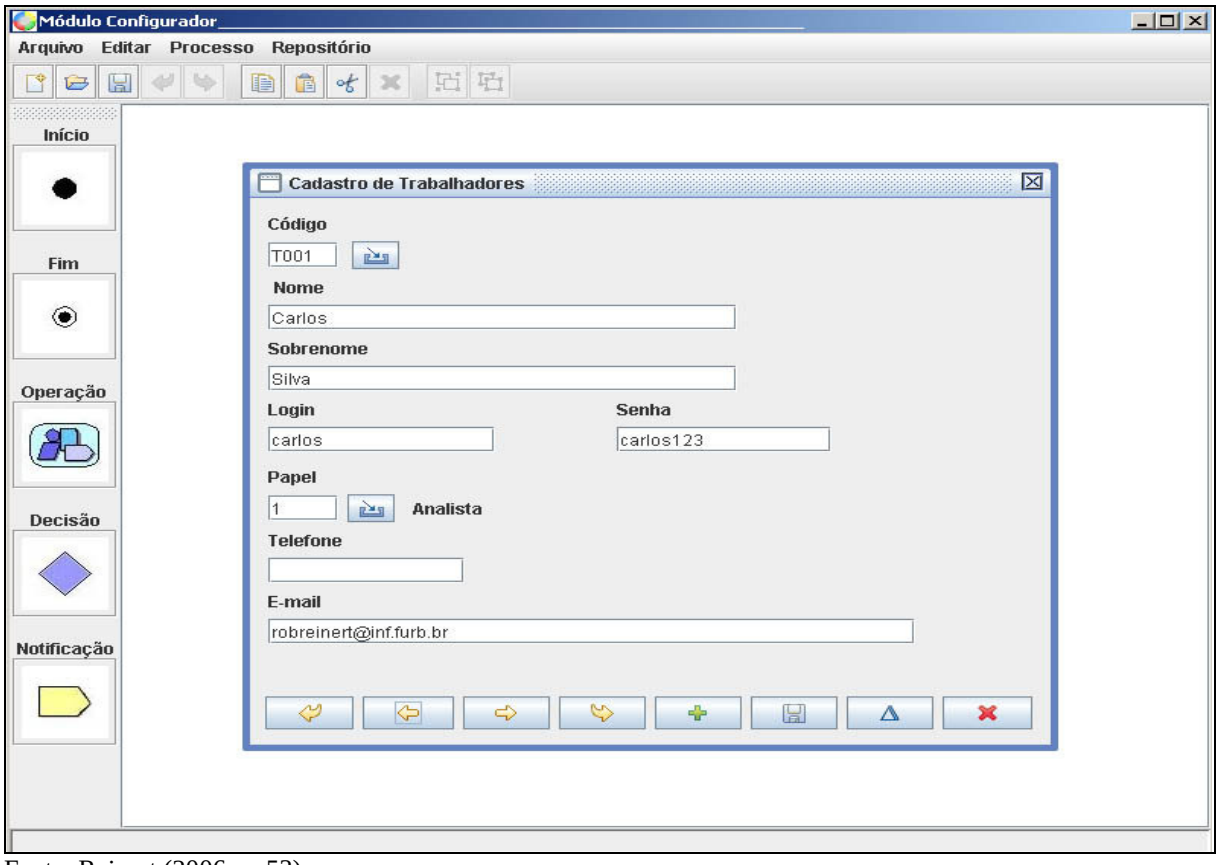
a) páginas Java *Server Pages* (JSP) para as telas do módulo de execução;
b) *framework* JGraph para a área de modelagem de processo do módulo de configuração;
c) *framework* Mentawai como servidor de execução *web* da camada servidora;
d) *framework Hibernate* com banco de dados Apache Derby para a camada de persistência.

A Figura 5 ilustra a tela de modelagem de processo do módulo de configuração. Na Figura 6 é exemplificada uma das telas de configuração da ferramenta. A Figura 7 e a Figura 8 demonstram, respectivamente, as telas de iniciar processo e de acompanhamento de atividades pendentes do módulo de execução.



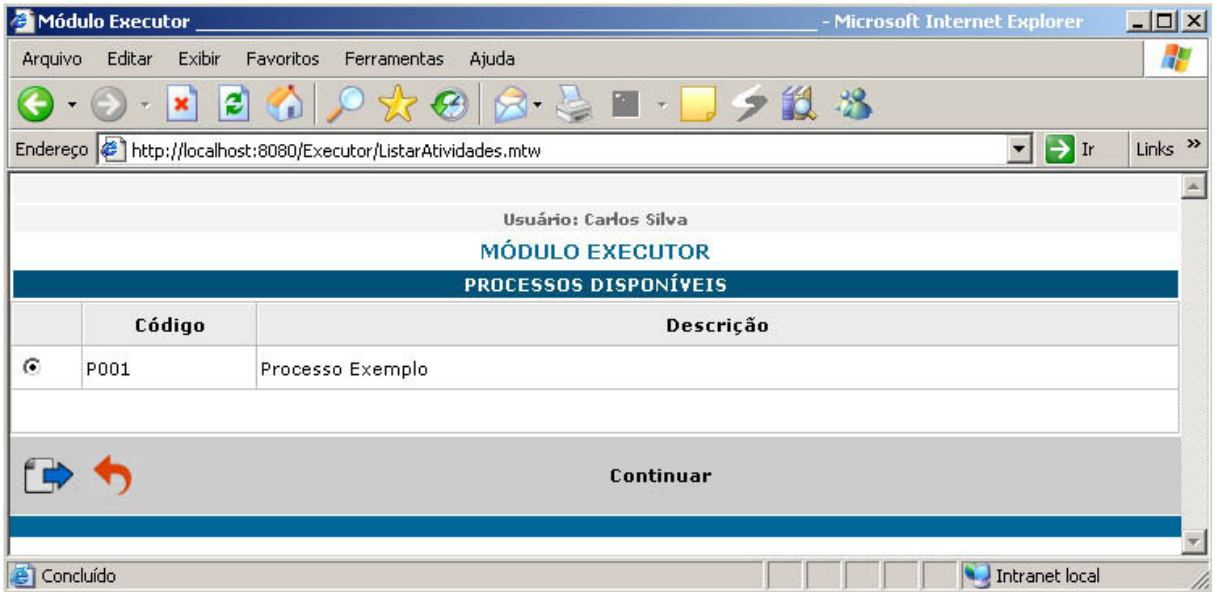
Fonte: Reinert (2006, p. 50).

Figura 5 – Tela de modelagem de processo da ferramenta de Reinert



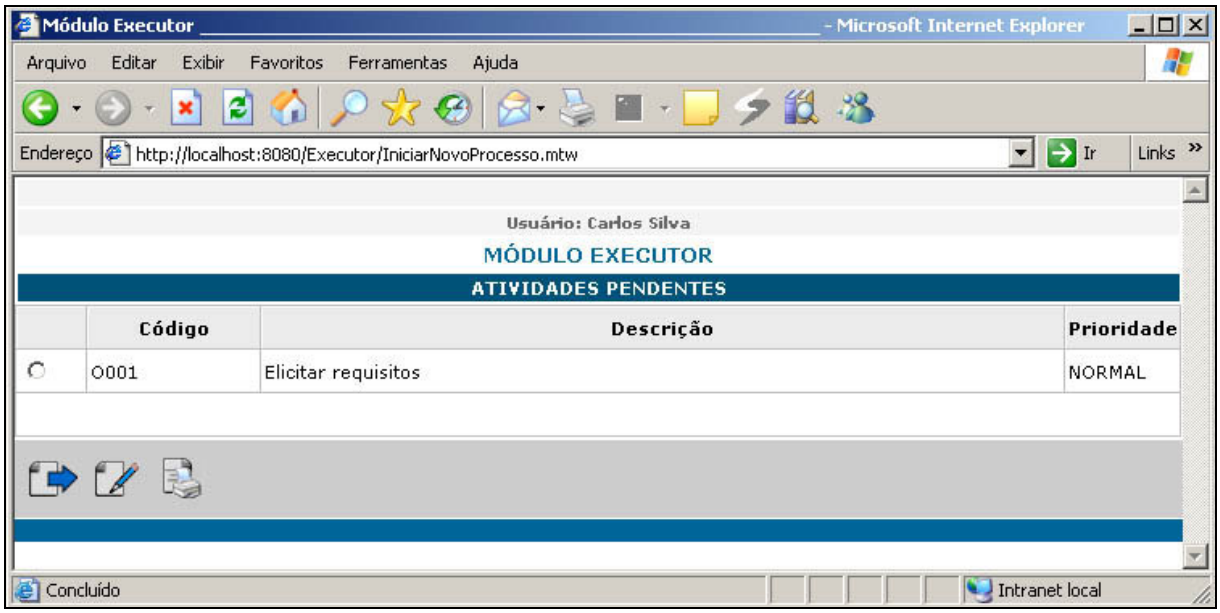
Fonte: Reinert (2006, p. 53).

Figura 6 – Tela de cadastro de trabalhadores da ferramenta de Reinert



Fonte: Reinert (2006, p. 64).

Figura 7 – Tela de iniciar processos da ferramenta de Reinert



Fonte: Reinert (2006, p. 64).

Figura 8 – Tela de atividades pendentes da ferramenta de Reinert

3 DESENVOLVIMENTO DA FERRAMENTA

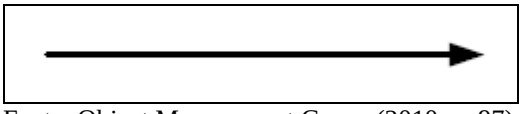
Este capítulo descreve o processo de desenvolvimento da ferramenta. Inicia explicando quais os elementos da BPMN foram contemplados neste trabalho. Em seguida lista quais os requisitos levantados, demonstra a especificação e os detalhes da implementação. Por fim, expõe os resultados obtidos após o estudo de caso.

3.1 IMPLEMENTAÇÃO DA BPMN

Com relação à BPMN, o foco deste trabalho foi na execução de atividades dentro de uma organização com a possibilidade de tomada de decisão e paralelismo. Para isto, foram contemplados os subtipos básicos dos principais elementos de definição de processos de negócio, que são os *events*, *activities* e *gateways* (OBJECT MANAGEMENT GROUP, 2010, p. 27), além do *sequence flow*, que é utilizado para conectar esses elementos, e da *lane* para organizá-los visualmente. As seções a seguir explicam cada um desses componentes conforme a especificação da BPMN, indicando como cada um deles foi implementado. Foi tomado como base a revisão de junho de 2010 da versão 2.0 da BPMN.

3.1.1 *Sequence flows*

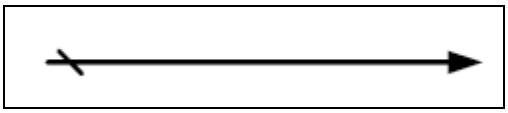
O *sequence flow* é um elemento utilizado para mostrar a ordem dos elementos em um processo. É representado por uma seta unidirecional conectada a um elemento origem e a um elemento alvo, ambos pertencentes à categoria *flow objects* (*events*, *activities* e *gateways*) (OBJECT MANAGEMENT GROUP, 2010, p. 97). A Figura 9 ilustra a representação gráfica do *sequence flow*.



Fonte: Object Management Group (2010, p. 97).
Figura 9 – Representação gráfica do *sequence flow*

Ainda segundo o Object Management Group (2010, p. 97), o *sequence flow* pode

conter uma expressão condicional que indica que o fluxo só pode passar por ele se esta expressão for avaliada como verdadeira. É possível definir um *sequence flow* como padrão, o qual é seguido quando as expressões de todos os outros *sequence flows* de saída do mesmo elemento de origem forem inválidas (OBJECT MANAGEMENT GROUP, 2010, p. 98). O sequence flow padrão é sinalizado com uma barra no início do conector, conforme ilustrado na Figura 10.



Fonte: Object Management Group (2010, p. 98).
Figura 10 – Representação gráfica do *sequence flow* padrão

Apesar da especificação da BPMN não limitar o uso de expressões condicionais nos *sequence flows* pelos tipos de elemento conectados, neste trabalho as expressões condicionais só podem ser utilizadas em *sequence flows* cujo elemento de origem é um *gateway* do tipo *exclusive* ou *inclusive*, conforme detalhes descritos na seção 3.1.4.

3.1.2 *Events*

Conforme o Object Management Group (2010, p. 240), os *events* representam algo que acontece durante a execução de um processo e que afetam o seu fluxo. São divididos nos tipos *start events*, *end events* e *intermediate events*.

O *start event* indica onde a execução de um processo em particular irá ser iniciada, não aceita *sequence flows* de entrada e é representado por um círculo desenhado com uma única linha fina (OBJECT MANAGEMENT GROUP, 2010, p. 245). Quando o *start event* é dividido em dois ou mais pontos do processo de negócio, ele é representado com um sinal de paralelismo, desenhado como um sinal de adição (+), no seu centro (OBJECT MANAGEMENT GROUP, 2010, p. 248).

O *end event* indica onde a execução de um processo termina, não aceita *sequence flows* de saída e é representado com uma única linha espessa (OBJECT MANAGEMENT GROUP, 2010, p. 253).

Devido à complexidade e à necessidade de interação com outros recursos que fogem ao escopo deste trabalho, como serviços de mensagens e de agendamento, os *intermediate events* e os demais subtipos de *start event* e *end event* não explicados nesta seção não foram contemplados por este trabalho. Os tipos de *events* implementados são ilustrados na Figura

11.

Figura 11 – Representação gráfica dos tipos de *events* implementados

Quando um processo é iniciado, o executor procura pelo *start event* definido na modelagem do fluxo e executa-o. Se houver mais de um *start event* no fluxo, estes são iniciados em linhas de execução paralelas. Cada linha é executada, seguindo o fluxo definido pelo *sequence flow* correspondente, até que encontre um *end event*.

Apesar da especificação da BPMN não definir que os *events* são obrigatórios, foi convencionado neste trabalho que é necessário ao menos um *start event* e um *end event* no fluxo para iniciar e terminar sua execução.

3.1.3 *Activities*

As *activities* representam pontos do processo de negócio onde é executado algum trabalho. São divididas nos tipos *task*, *sub-process* e *call*. Estes são os elementos executáveis de um processo modelado com BPMN (OBJECT MANAGEMENT GROUP, 2010, p. 155).

Uma *activity* do tipo *task* representa uma tarefa atômica dentro do fluxo de um processo. Geralmente, um usuário final ou uma aplicação é utilizado para realizar o trabalho quando uma *task* é executada. Graficamente, é representada por um retângulo de cantos arredondados desenhado com linha fina (OBJECT MANAGEMENT GROUP, 2010, p. 160). A representação gráfica da *task* é ilustrada na Figura 12.

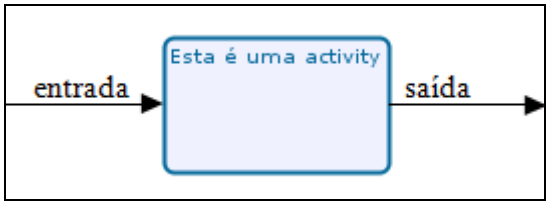


Figura 12 – Representação gráfica da *activity* do tipo *task*

Por questão de escopo, as *activities* dos tipos *sub-process* e *call* não foram contempladas por este trabalho.

Segundo o Object Management Group (2010, p. 157), uma *activity* pode conter zero ou mais *sequence flows* de entrada e de saída, considerando os seguintes critérios:

a) quando não possui entrada, assume-se que a *activity* é executada ao iniciar o processo;

b) quando possui mais de uma entrada, a *activity* é executada sempre que atingida por algum *sequence flow* de entrada, sem qualquer controle de sincronismo;
c) quando não possui saída, assume-se que a *activity* é o fim da linha de execução que a atingiu;
d) quando possui mais de uma saída, significa que é criada uma linha de execução para cada *sequence flow*.

Devido às *activities* não possuírem controle de sincronismo e também como forma de padronizar os processos modelados, neste trabalho foi assumido como convenção que cada *activity* deve possuir um, e somente um, *sequence flow* de entrada e um, e somente um, *sequence flow* de saída. A execução de uma tarefa correspondente a uma *activity* do tipo *task* é iniciada assim que ela é atingida pelo seu *sequence flow* de entrada. Quando a tarefa é concluída, o fluxo da linha de execução correspondente é continuado através do *sequence flow* de saída da *activity* em questão.

3.1.4 *Gateways*

Gateways são utilizados para controlar como os *sequence flows* interagem sob as condições em que eles convergem ou divergem em um processo. Possuem mecanismos que permitem ou bloqueiam a passagem do fluxo através deles. Cada gateway pode conter múltiplos *sequence flows* de entrada e, simultaneamente, múltiplos *sequence flows* de saída. Isso possibilita que, conforme as linhas de execução chegam ao *gateway*, elas sejam unidas (convergência) ou separadas (divergência) de acordo com os mecanismos do próprio gateway (OBJECT MANAGEMENT GROUP, 2010, p. 295-296).

Os *gateways*, como em outras notações de fluxo de execução, são representados por um diamante desenhado com linhas finas. Além do diamante tradicionalmente utilizado para representar decisões exclusivas, a BPMN estende esse comportamento para permitir representar qualquer tipo de controle de fluxo de um processo. Para tal, os *gateways* são divididos nos tipos *exclusive, inclusive, parallel, complex, event-based* e *parallel-based*, sendo que cada um possui uma marcação interna que indica o seu tipo (OBJECT MANAGEMENT GROUP, 2010, p. 295-296).

Os tipos *event-based* e *parallel-based* não são contemplados por este trabalho pelo mesmo motivo dos *intermediate events*, conforme descrito na seção 3.1.2. O tipo *complex* não foi contemplado porque não foi encontrada uma lógica para automatizá-lo, devido à

subjetividade de sua interpretação. A Figura 13 ilustra os tipos implementados neste trabalho.

Figura 13 – Representação gráfica dos tipos de *gateway* implementados

Os *gateways* tipo *exclusive*, quando divergentes, são utilizados para criar caminhos alternativos no fluxo de um processo, sendo que apenas um deles pode ser seguido. Os caminhos são representados por *sequence flows* de saída do *gateway*, cada qual associado a uma expressão condicional. Durante a execução, o primeiro caminho que tiver a expressão avaliada como verdadeira é seguido e os demais são descartados. O *gateway* pode conter um *sequence flow* de saída padrão, o qual não está associado a uma expressão. Quando nenhuma das expressões do *gateway* for verdadeira, se existir um *sequence flow* padrão, este é seguido. Caso contrário, é gerado um erro de execução (OBJECT MANAGEMENT GROUP, 2010, p. 298-299).

Ainda segundo o Object Management Group (2010, p. 299), os *gateways* tipo *exclusive*, quando convergentes, são utilizados para juntar caminhos alternativos sem sincronização. Quando um *sequence flow* de entrada atinge o *gateway*, ele é direcionado para o *sequence flow* de saída. A Figura 14 mostra um exemplo de utilização de um *gateway* tipo *exclusive* de divergência e outro para convergir os caminhos.

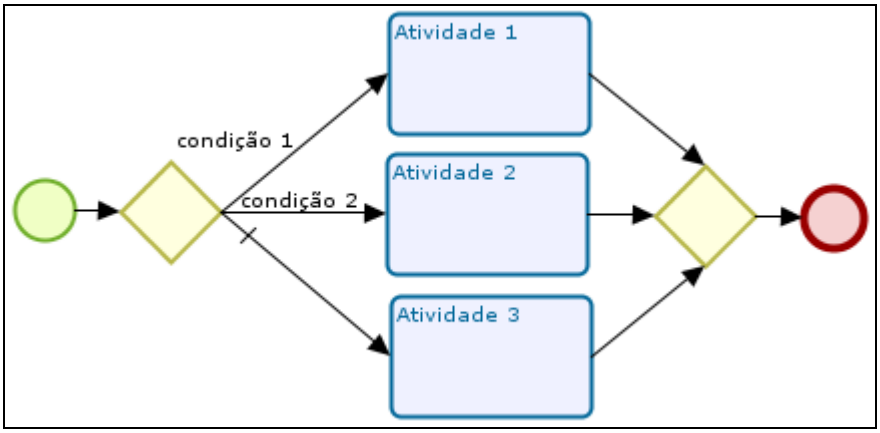


Figura 14 – Exemplo de utilização de *gateway* tipo *exclusive*

Os *gateways* tipo *inclusive*, quando divergentes, diferem do tipo *exclusive* apenas pelo fato de suportar paralelismo. Todas as expressões dos *sequence flows* de saída são avaliadas, e todas que forem verdadeiras tem o caminho correspondente seguido. Quando convergentes, os *gateways* tipo *inclusive* direcionam o fluxo de entrada para o fluxo de saída de forma sincronizada, esperando por todos os caminhos tomados a partir do *gateway* divergente

correspondente (OBJECT MANAGEMENT GROUP, 2010, p. 300). A Figura 15 mostra um exemplo de utilização de um *gateway* tipo *inclusive* de divergência e outro para convergir os caminhos.

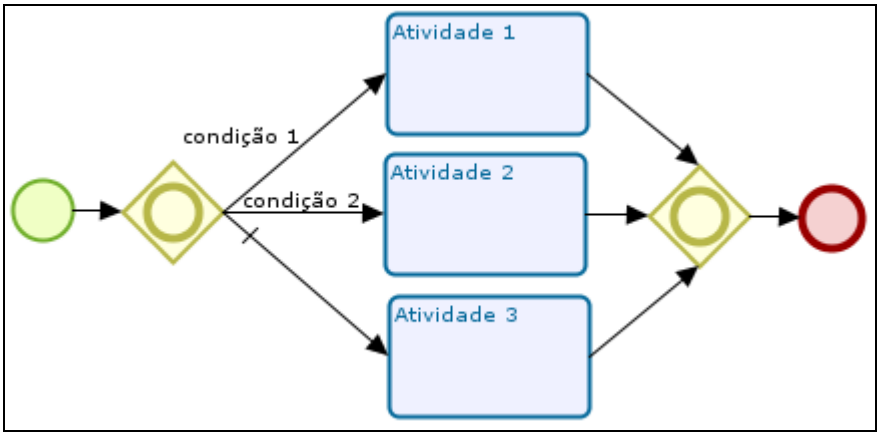


Figura 15 – Exemplo de utilização de *gateway* tipo *inclusive*

Os *gateways* tipo *parallel* são utilizados para criar fluxos paralelos e combiná-los de forma sincronizada. No *gateway* divergente, todos os caminhos de saída são executados paralelamente sem validar nenhuma expressão. No *gateway* convergente, os caminhos são combinados e direcionados para o *sequence flow* de saída após todos os *sequence flows* de entrada atingi-lo (OBJECT MANAGEMENT GROUP, 2010, p. 301-302). A Figura 16 mostra um exemplo de utilização de um *gateway* tipo *parallel* de divergência e outro para convergir os caminhos.

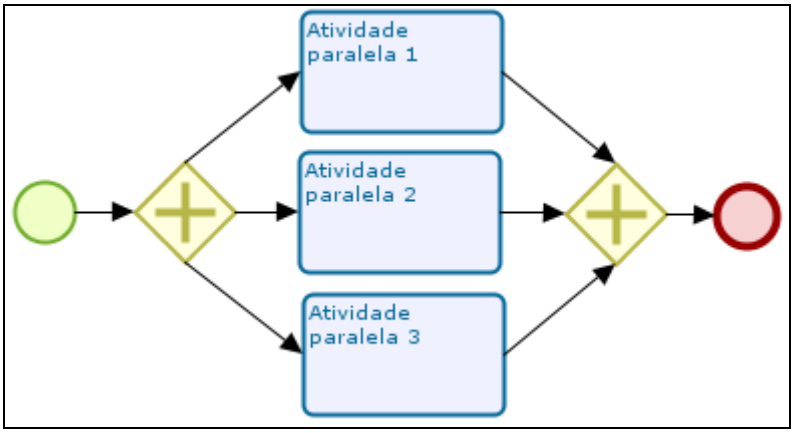


Figura 16 – Exemplo de utilização de *gateway* tipo *parallel*

Neste trabalho, estes três tipos de *gateway* são implementados seguindo os critérios definidos na especificação, descritos anteriormente.

3.1.5 *Lanes*

As *lanes* são sub-partições dentro de um processo. São utilizadas para agrupar e organizar *activities*. Por exemplo, separar as atividades de acordo com o setor da corporação e com as regras envolvidas. Uma *lane* pode conter outras *lanes*. Cada *lane* é representada por um retângulo desenhado com linha fina. Sua legenda pode ser colocada em qualquer ponto do retângulo, desde que não seja separado do seu conteúdo (OBJECT MANAGEMENT GROUP, 2010, p. 313-314).

Neste trabalho, a legenda da *lane* é desenhada verticalmente na sua extremidade esquerda, conforme ilustrado na Figura 17.

Figura 17 – Representação gráfica das *lanes*

3.2 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Vale lembrar que a ferramenta é dividida em dois módulos: o editor e o executor de processos de negócio. O editor tem a finalidade de possibilitar a modelagem do fluxo do processo e gerar a definição deste fluxo em um formato que possa ser processado pelo executor. O executor tem a finalidade de executar o processo modelado seguindo o seu fluxo conforme as regras descritas na seção 3.1.

O editor de processos de negócio deve:

a) possibilitar a criação de novos fluxos (Requisito Funcional – RF);
b) possibilitar a alteração de fluxos existentes (RF);
c) possibilitar a execução de fluxos em modo de testes (RF);

d) suportar controle de versão, para que fluxos que já estejam sendo utilizados permaneçam disponíveis para o executor (RF);
e) permitir ser estendido (RF);
f) utilizar elementos baseados em BPMN (Requisito Não Funcional – RNF);
g) suportar servidor Java EE GlassFish versão 3.1 (RNF);
h) permitir acesso *web* via navegador Google Chrome versão 12 (RNF).

O executor de processos de negócio deve:

a) interagir com interfaces externas (outros sistemas e máquinas de *workflow*) utilizando *web services* (RF);
b) utilizar protocolo SOAP na comunicação via *web services* (RNF);
c) permitir ser estendido para interagir nativamente com interfaces externas (RF);
d) permitir interromper a execução de processos (RF);
e) executar um fluxo do início ao fim na mesma versão (RF);
f) suportar servidor Java EE GlassFish versão 3.1 (RNF);
g) suportar banco de dados SQL *Server* 2008 (RNF);
h) permitir ser estendido para utilizar outras formas de armazenamento, além de banco de dados SQL *Server* (RF).

3.3 ESPECIFICAÇÃO

A especificação foi feita através da modelagem de diagramas de casos de uso, de atividades, de sequência, de classes, de componentes e entidade-relacionamento. Os diagramas de casos de uso, atividades, sequência, classes e componentes foram criados com base nas definições da UML a partir da IDE do Eclipse com a extensão de modelagem desenvolvida pela Senior Sistemas S/A. O diagrama entidade-relacionamento foi criado com a ferramenta SQL *Server Management Studio*, a qual acompanha o SQL *Server* 2008 e gera a base de dados diretamente a partir do diagrama.

3.3.1 Diagramas de casos de uso

Esta seção contém os diagramas dos casos de uso da ferramenta. Cada caso de uso é detalhado no Apêndice A, juntamente com a descrição dos atores.

A Figura 18 demonstra os casos de uso de modelagem de fluxos de processos. Estes representam a forma como o ator interage com o editor baseado em BPMN. Possibilitam a criação de novos fluxos, a alteração e exclusão dos já existentes, além da execução da versão de testes para garantir a qualidade do fluxo modelado.

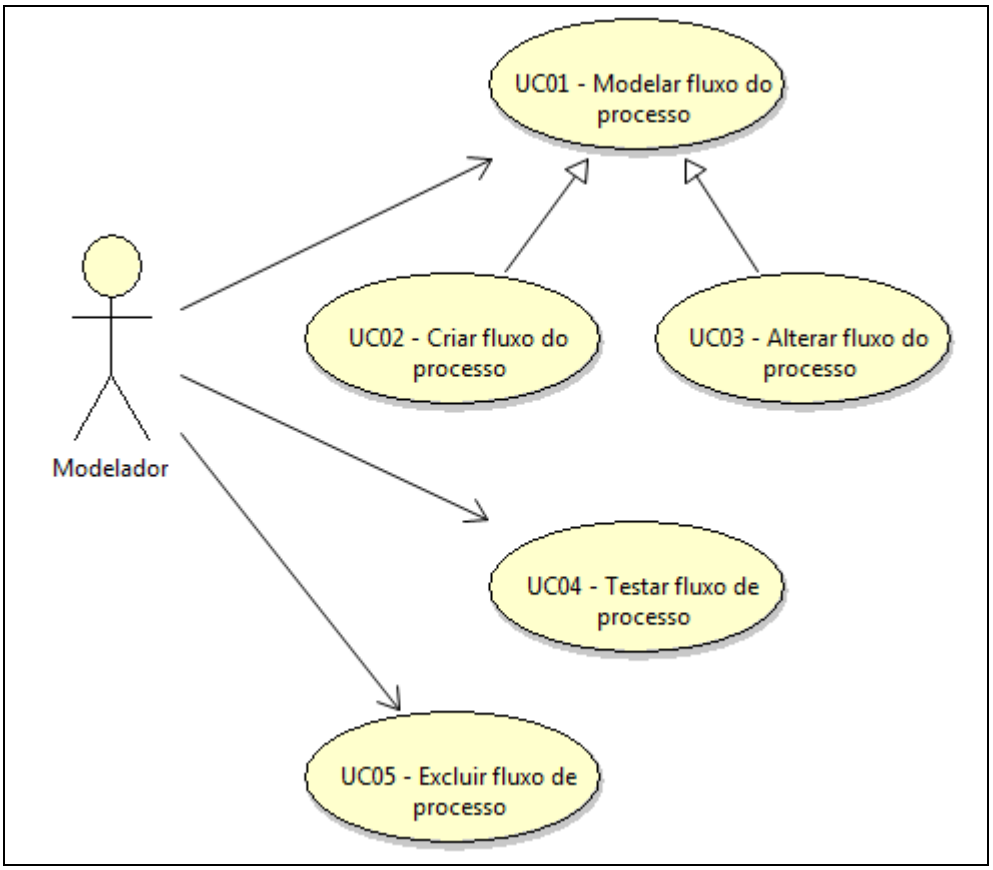


Figura 18 – Diagrama de casos de uso de modelagem

Na Figura 19 constam os casos de uso que possibilitam a administração do ambiente de execução. Consistem em tornar os fluxos modelados disponíveis ou indisponíveis para uso e acompanhar sua execução, podendo cancelá-los quando o administrador julgar necessário.

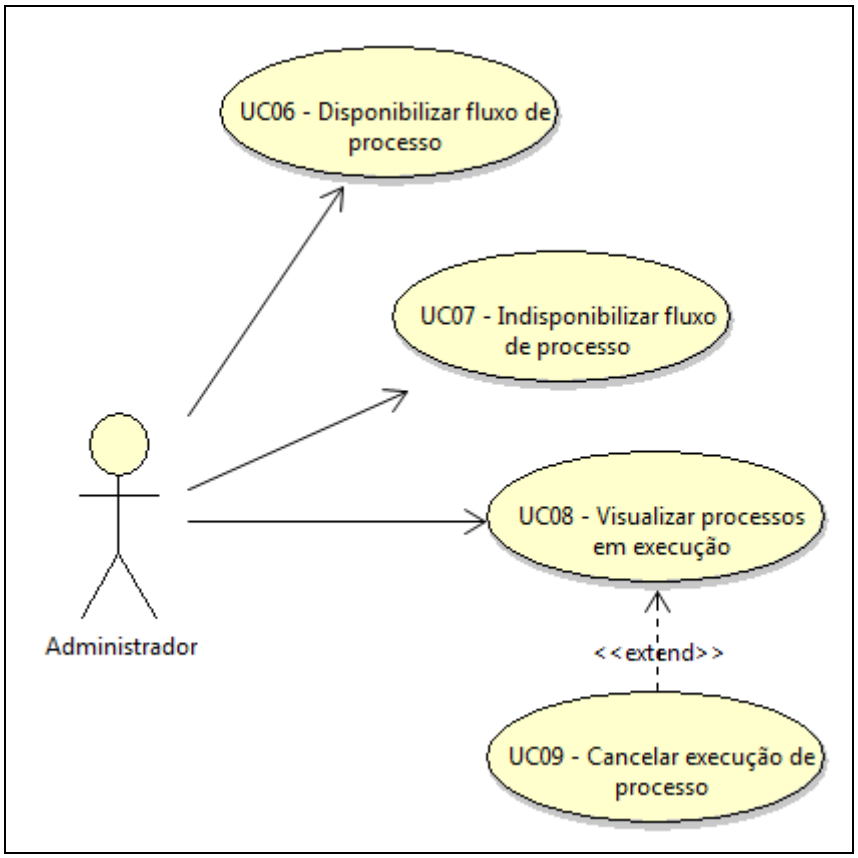


Figura 19 – Diagrama de casos de uso de administração

Para especificar os casos de uso de execução de processos, os quais representam o uso propriamente dito dos fluxos modelados com o editor, optou-se por utilizar os atores "sistema externo" e "usuário". A interação direta com o executor de processos é feita por um ou mais sistemas externos, normalmente uma máquina de *workflow*. No entanto, o início de um novo processo e o tratamento das tarefas originadas por ele normalmente são atividades realizadas pelos usuários desses sistemas externos. Sendo assim, para facilitar o entendimento da ferramenta o ator "usuário" foi considerado nesta especificação, conforme demonstrado na Figura 20.

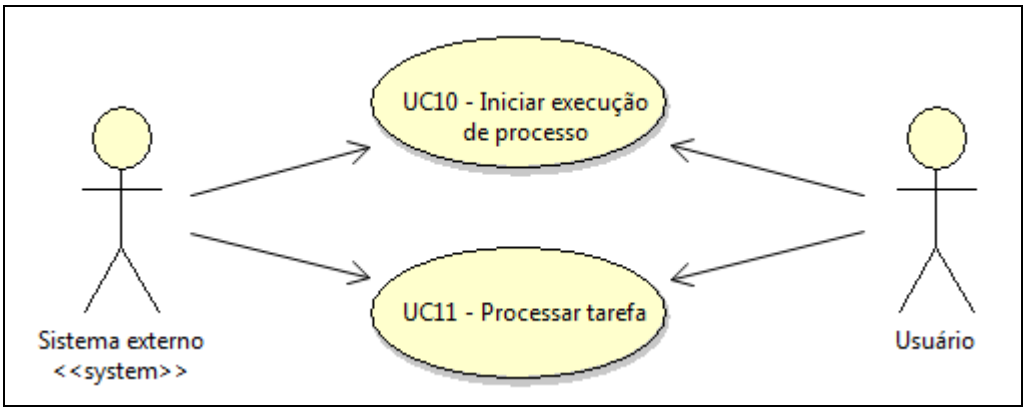


Figura 20 – Diagrama de casos de uso de execução

A Figura 21 representa o diagrama do caso de uso que possibilita desenvolver extensões para a ferramenta, agregando a ela novas funcionalidades.

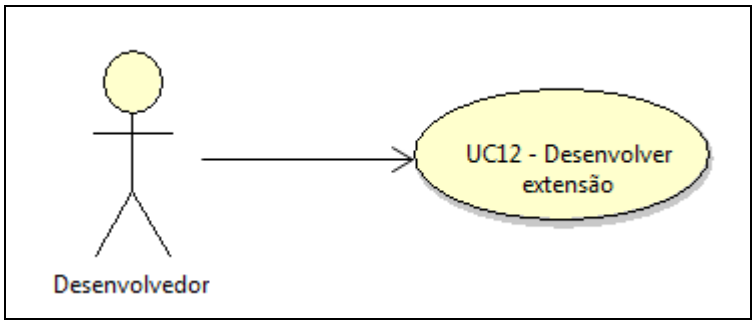


Figura 21 – Diagrama de caso de uso de desenvolvimento

3.3.2 Diagramas de atividades

Nesta seção são apresentados os diagramas de atividade que definem como cada fluxo de processo é executado. Inicialmente, na Figura 22, é demonstrado como o executor inicia o processamento da definição do fluxo gerada pelo editor a partir do seu modelo. Em seguida estão os diagramas específicos do processamento de cada elemento da BPMN contemplado. Para estes diagramas, a atividade inicial representa o momento em que um *sequence flow* de entrada do elemento em questão é atingido. A Figura 23 representa o processamento das *activities*. A Figura 24, a Figura 25 e a Figura 26 correspondem aos *gateways* de tipo *exclusive*, *inclusive* e *parallel*, respectivamente. Por fim, na Figura 27 é apresentado o *start event* e na Figura 28 o *end event*. O *lane* não possui diagrama porque ele é meramente representativo. O processamento dos *sequence flows* está embutido no diagrama de cada elemento.

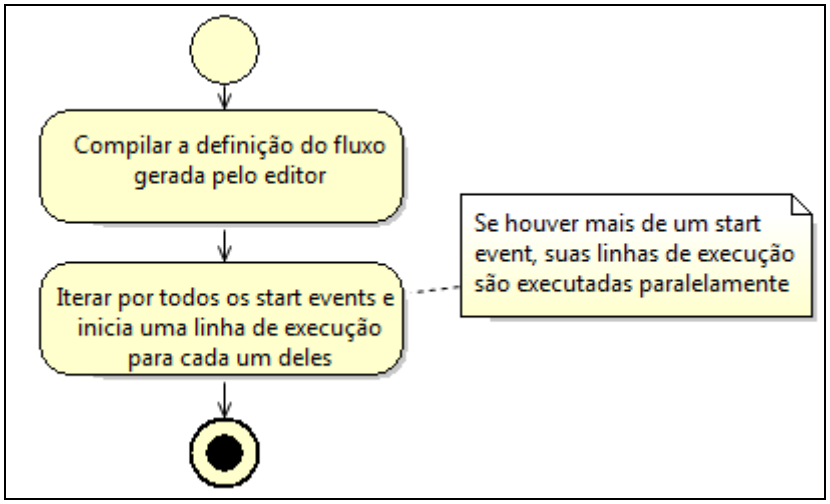


Figura 22 – Diagrama de atividade do início de execução do fluxo

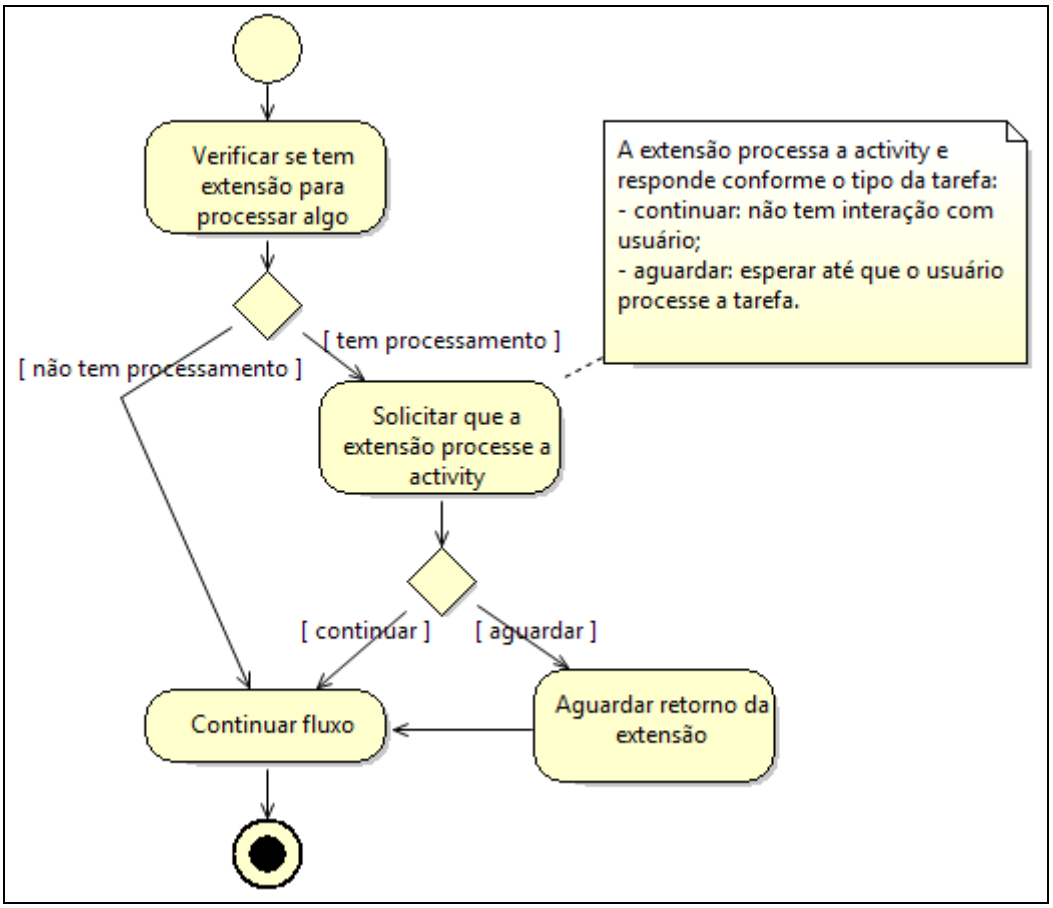


Figura 23 – Diagrama de atividade do elemento *activity*

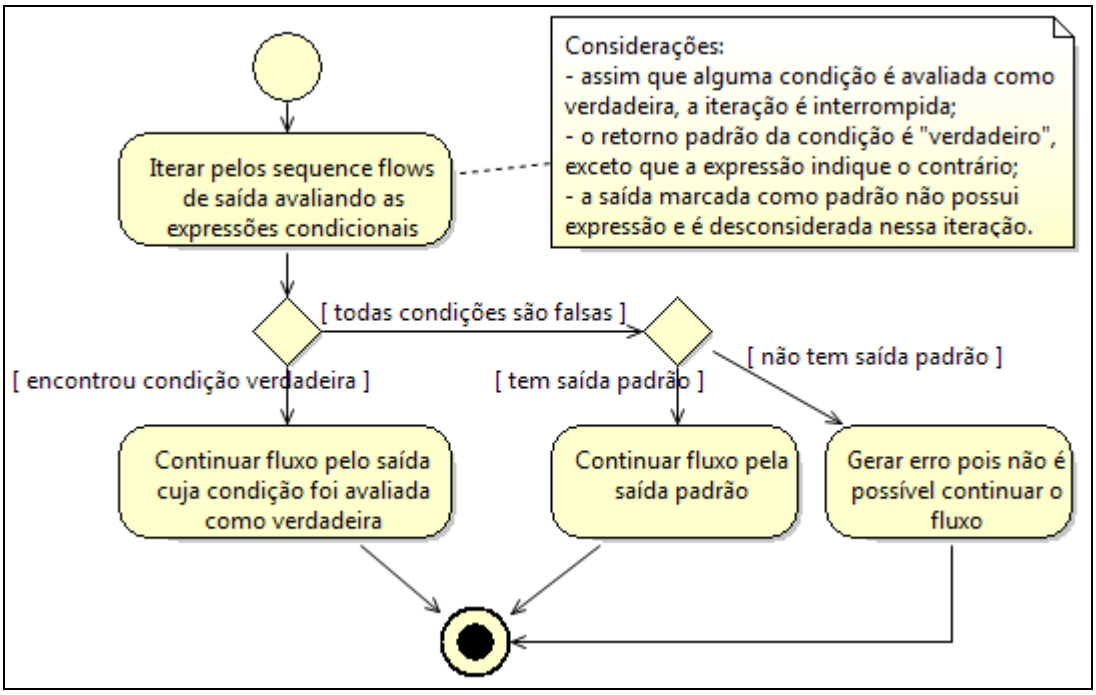


Figura 24 – Diagrama de atividade do elemento *gateway* tipo *exclusive*

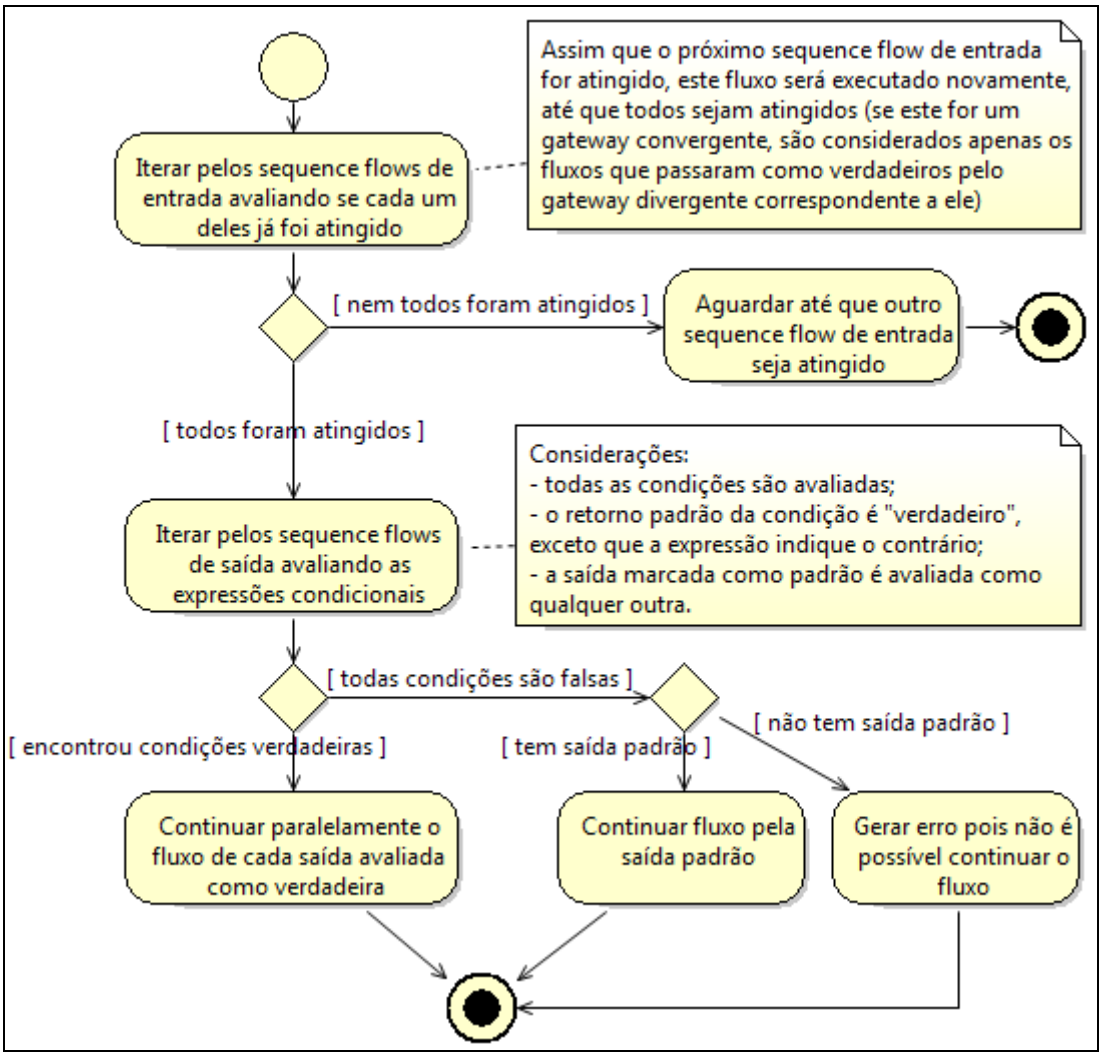


Figura 25 – Diagrama de atividade do elemento *gateway* tipo *inclusive*

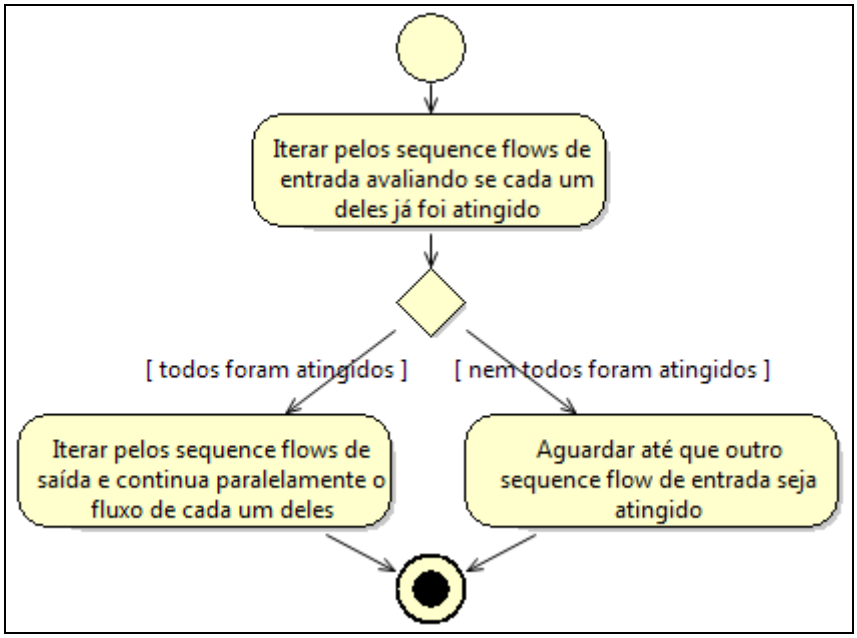


Figura 26 – Diagrama de atividade do elemento *gateway* tipo *parallel*

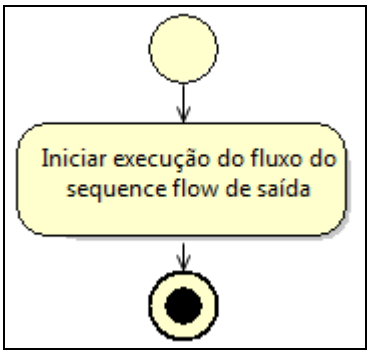


Figura 27 – Diagrama de atividade do elemento *start event*

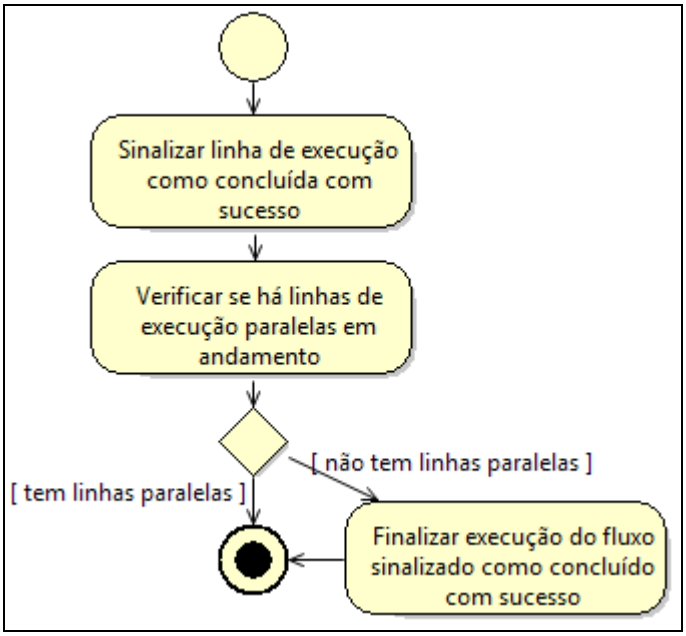


Figura 28 – Diagrama de atividade do elemento *end event*

3.3.3 Diagrama de sequência

A Figura 29 demonstra a sequência de interações entre os participantes da execução do fluxo de um processo no seu cenário mais usual. Um usuário, através de um sistema externo, solicita que um processo seja iniciado. Após iniciado, o processo pode conter *activities* que representam tarefas que são processadas por usuários e outras que são processadas por sistemas externos sem interação humana. É importante ressaltar que:

a) o ator “Usuário” representa um ou mais usuários do processo. Normalmente um processo de negócio envolve a participação de diferentes pessoas, cada uma executando seu papel;
b) o ator “Sistema externo” pode representar um ou mais sistemas.

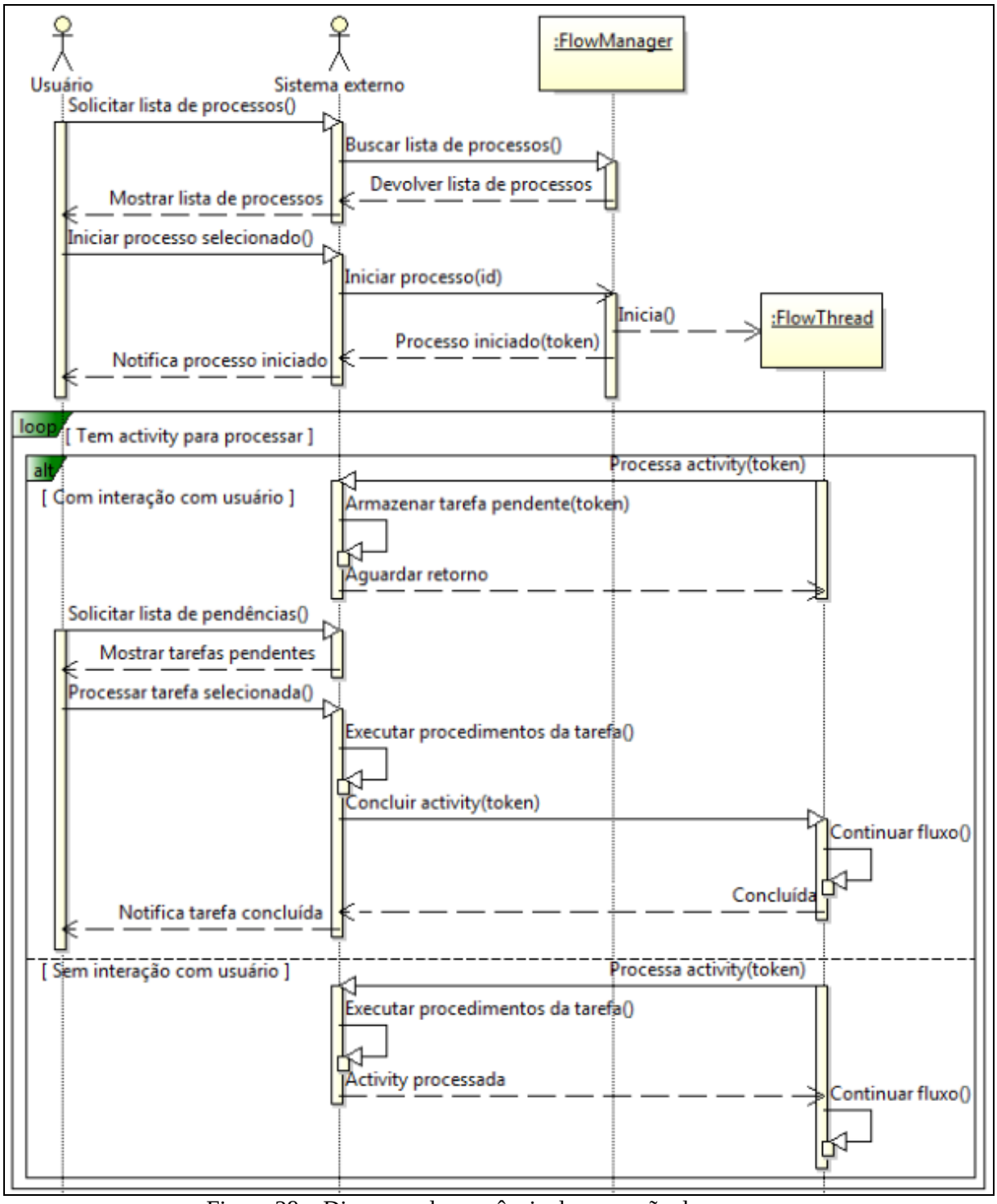


Figura 29 – Diagrama de sequência de execução de processo

A classe `Flow` foi omitida porque ela não exerce um papel ativo no contexto deste diagrama, onde é utilizada apenas como ligação entre as classes `FlowManager` e `FlowThread`. Há uma instância da classe `Flow` para representar cada instância do fluxo do processo em execução. Cada instância de `Flow` agrega uma ou mais instâncias de `FlowThread`, onde cada instância de `FlowThread` representa uma linha de execução dentro do mesmo fluxo. Quando o usuário acessa o sistema externo para iniciar um novo processo, o sistema solicita a lista de

fluxos disponíveis ao `FlowManager`. Assim que o usuário solicita a execução do processo, o sistema externo notifica ao `FlowManager` para que ele dê início à execução, informando o identificador do processo selecionado pelo usuário. Neste momento é criada uma instância de `FlowThread` para cada *start event* contido no fluxo. Cada instância de `FlowThread` permanece em *loop* até que termine o fluxo da linha de execução correspondente. Ao processar uma *activity* contida no fluxo a `FlowThread` solicita que o sistema externo processe a tarefa correspondente. O sistema externo atende à solicitação, considerando:

a) se a tarefa necessita de intervenção de um usuário para ser processada, o sistema externo gera uma pendência para o usuário em questão e a `FlowThread` entra em estado de espera. Quando o usuário solicita a lista de pendências ao sistema externo ele visualiza um registro correspondente à tarefa gerada pela solicitação da `FlowThread`. Quando o usuário trata a tarefa em questão o sistema externo executa os procedimentos referentes a ela e notifica a `FlowThread`. Esta, por sua vez, executa os procedimentos de finalização da *activity* e continua a execução do fluxo. A instância de `FlowThread` que recebe a notificação é identificada pelo *token* correspondente à tarefa, o qual é fornecido pela própria `FlowThread` quando ela solicita o processamento da tarefa;
b) se a tarefa não necessita de intervenção de um usuário para ser processada, o sistema externo realiza os procedimentos referentes a ela e devolve o resultado para a `FlowThread`, a qual continua a execução do fluxo.

A comunicação do sistema externo com as classes `FlowManager` e `FlowThread` pode ser feita diretamente via código Java ou através de *web services*. No segundo caso os *web services* representam uma camada intermediária que atua como ponte entre o sistema externo e o executor de processos.

3.3.4 Diagrama de componentes

Os componentes da ferramenta desenvolvida são representados em três grupos: o pacote do *framework*, o pacote da camada de acesso *web* e as extensões. O *framework* é composto pelas rotinas dos componentes *execution*, *definition*, *extension*, dao e *core*. A camada de acesso *web* é composta pelas aplicações console e editor. As extensões disponíveis são denominadas dao-mssql e ext-*webservice*. A Figura 30 ilustra o diagrama da relação entre

estes componentes. As cores nos símbolos indicam em qual módulo eles são utilizados, sendo que a cor amarela representa o executor, a cor azul representa o editor e a cor verde representa ambos.

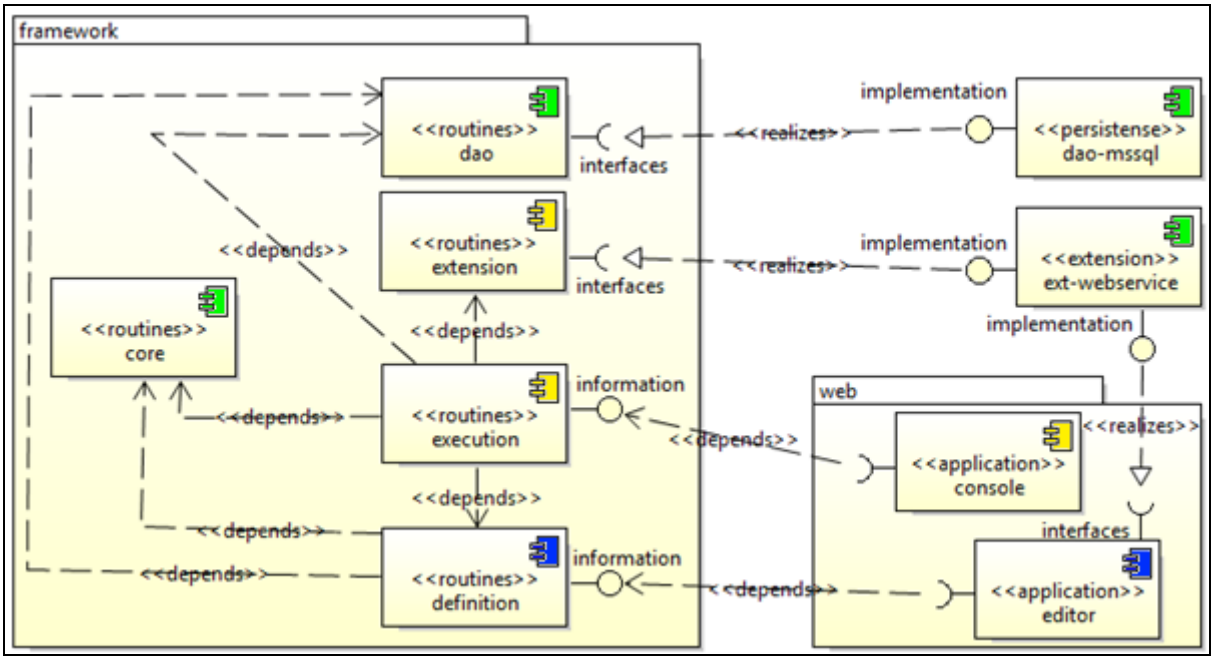


Figura 30 – Diagrama de componentes

Cada componente tem uma finalidade específica, conforme descrito a seguir:

a) *execution*: é o componente mais importante da ferramenta, pois possibilita ao *framework* executar os processos de negócio. Suas rotinas implementam toda a lógica de execução de BPMN conforme considerações da seção 3.1, acionando as rotinas de extensão quando necessário. Também provê informações de administração e monitoramento dos processos. A Figura 31 demonstra os pacotes de código fonte deste componente;

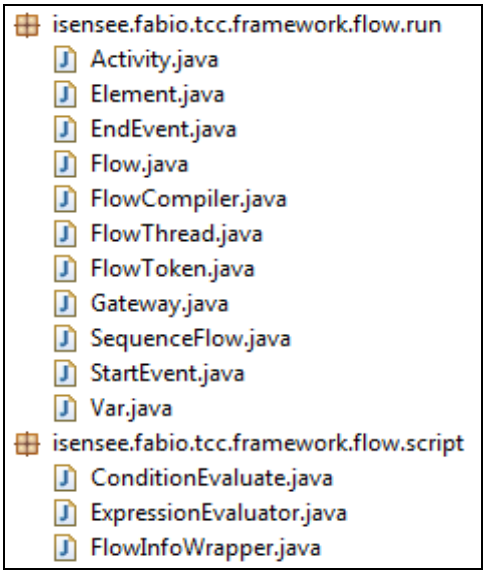


Figura 31 – Pacotes de código fonte do componente *execution*

b) *definition*: contém as rotinas utilizadas pelo editor para gerar a representação executável do fluxo do processo. Utilizando este componente, é possível criar um novo editor utilizando tecnologias diferentes do atual sem alterar nenhuma linha de código no *framework*. A Figura 32 demonstra o pacote de código fonte deste componente;

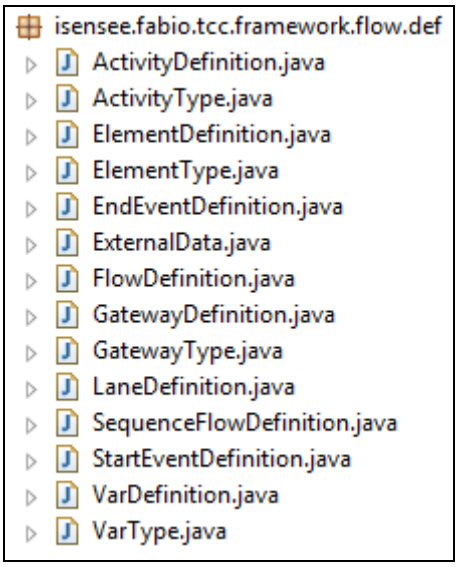


Figura 32 – Pacote de código fonte do componente *definition*

c) *extension*: expõe as interfaces que são implementadas ao desenvolver extensões para o *framework*. Contém também as rotinas de acesso e controle das extensões disponíveis. A Figura 33 demonstra o pacote de código fonte deste componente;

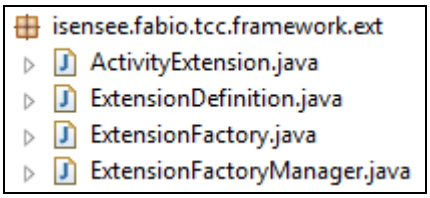


Figura 33 – Pacote de código fonte do componente *extension*

d) dao: expõe as interfaces que são implementadas ao desenvolver as extensões de persistência de dados da ferramenta;

e) *core*: contém rotinas básicas para o funcionamento da ferramenta, como:

- acesso às configurações,
- sincronização de objetos,
- serialização de dados;

f) ext-*webservice*: extensão baseada em SOA que realiza as interfaces do componente *extension*. Incorpora ao *framework* a capacidade de interagir com os sistemas externos através de *web services*, utilizando protocolo SOAP. Estende também o editor para possibilitar a definição do *script* que invoca os *web services*;

g) dao-mssql: extensão que realiza as interfaces do componente dao. Incorpora ao *framework* a capacidade de persistir as informações em banco de dados SQL *Server*;
h) console: aplicação *web* que permite ao administrador interagir com o *framework* através das rotinas de monitoramento e administração providas pelo componente *execution*. Os códigos fonte do console estão disponíveis no pacote `isensee.fabio.tcc.console`;
i) editor: aplicação *web* que possibilita modelar os fluxos dos processos de negócio. A modelagem é feita utilizando recursos de edição gráfica que geram a definição executável do fluxo através do componente *definition*. A implementação do editor divide-se em duas camadas:
   - rotinas do lado servidor, executadas dentro do servidor Java EE, cuja codificação encontra-se no pacote `isensee.fabio.tcc.editor` do projeto Java denominado "web",
   - rotinas do lado cliente, executadas pela máquina Java dentro do navegador *web*, cuja codificação encontra-se no projeto Java denominado "modeler".

3.3.5 Diagramas de classes

Os diagramas apresentados nesta seção representam as principais classes utilizadas pela ferramenta para executar os fluxos de processos. Inicialmente, na Figura 34, são demonstradas as classes do componente *core*. A Figura 35 mostra a classe `FlowManager` e seus relacionamentos. Esta classe é responsável por gerenciar as definições dos modelos dos fluxos e as instâncias dos fluxos em execução, além de prover os métodos para interação com os demais componentes da ferramenta. Na Figura 36 é apresentado o diagrama da estrutura utilizada como base para a definição dos modelos de fluxos, a qual é complementada pela estrutura hierárquica de definição de elementos da BPMN demonstrada na Figura 37. Em seguida, na Figura 38, são apresentadas as classes que representam os elementos da BPMN na sua forma executável e como eles se relacionam com as classes de definição. Estes elementos são executados utilizando a estrutura representada na Figura 39. Por fim são apresentadas as classes da extensão de *web services* do executor, na Figura 40, e as de extensão de DAO para SQL *Server*, na Figura 41. Visando não estender demasiadamente esta seção, os diagramas das classes específicas do *applet* do editor foram omitidos.

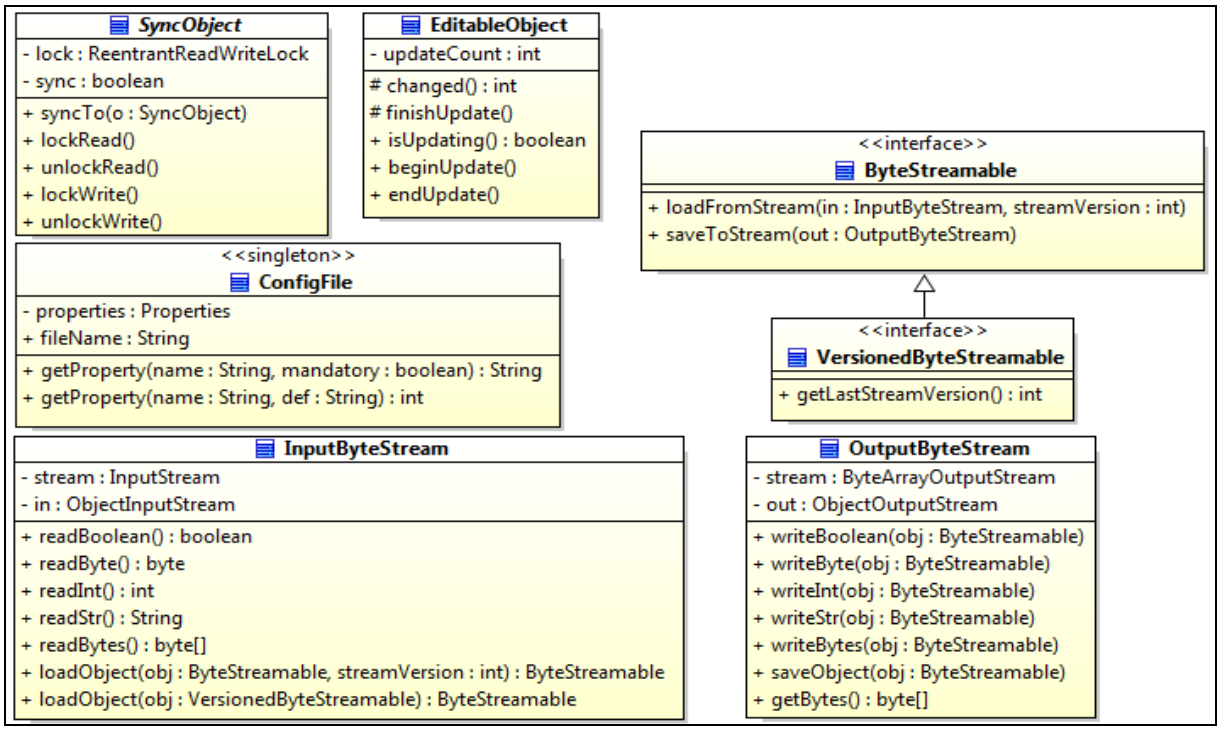


Figura 34 – Diagrama de classes do componente *core*

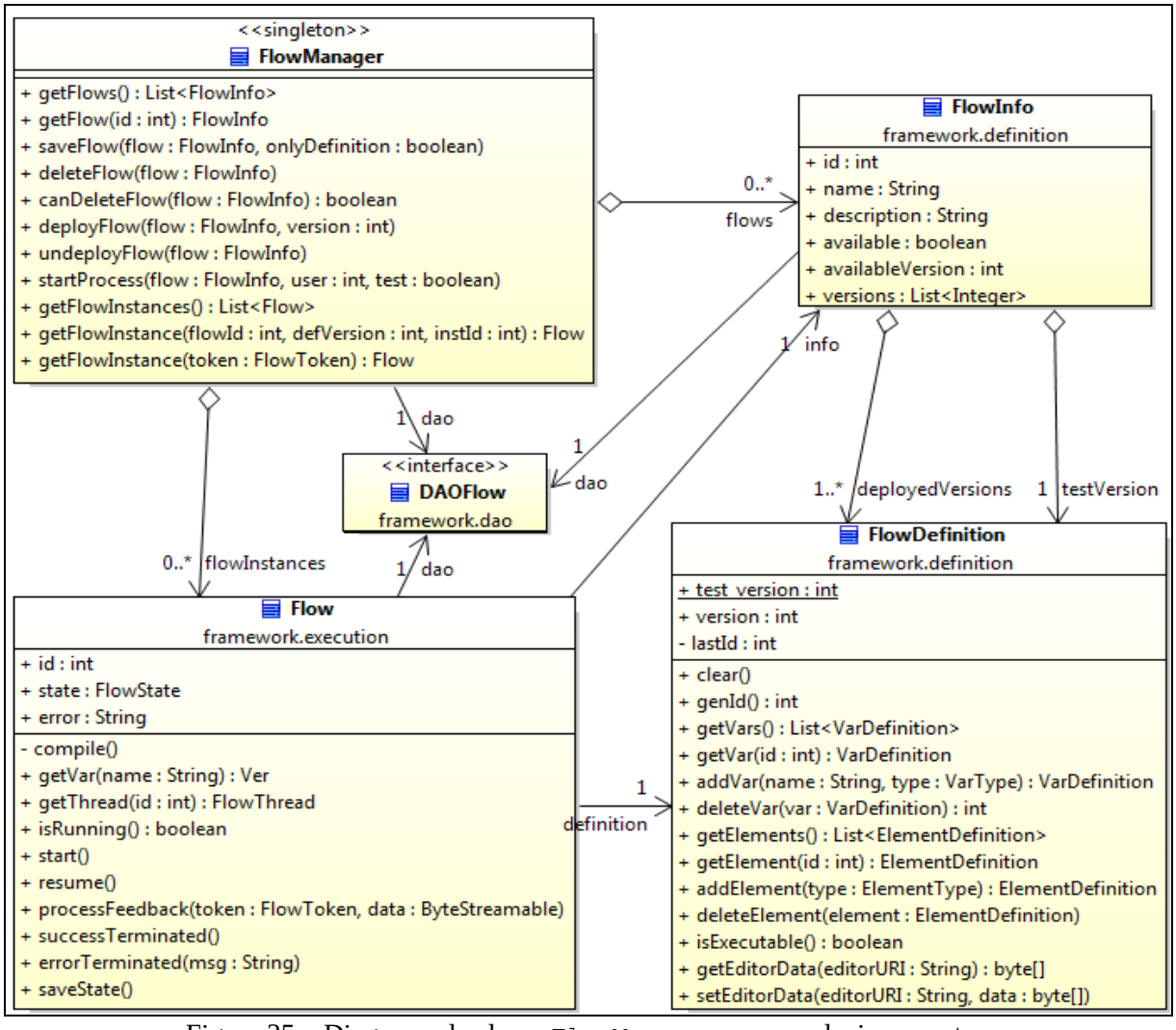


Figura 35 – Diagrama da classe `FlowManager` e seus relacionamentos

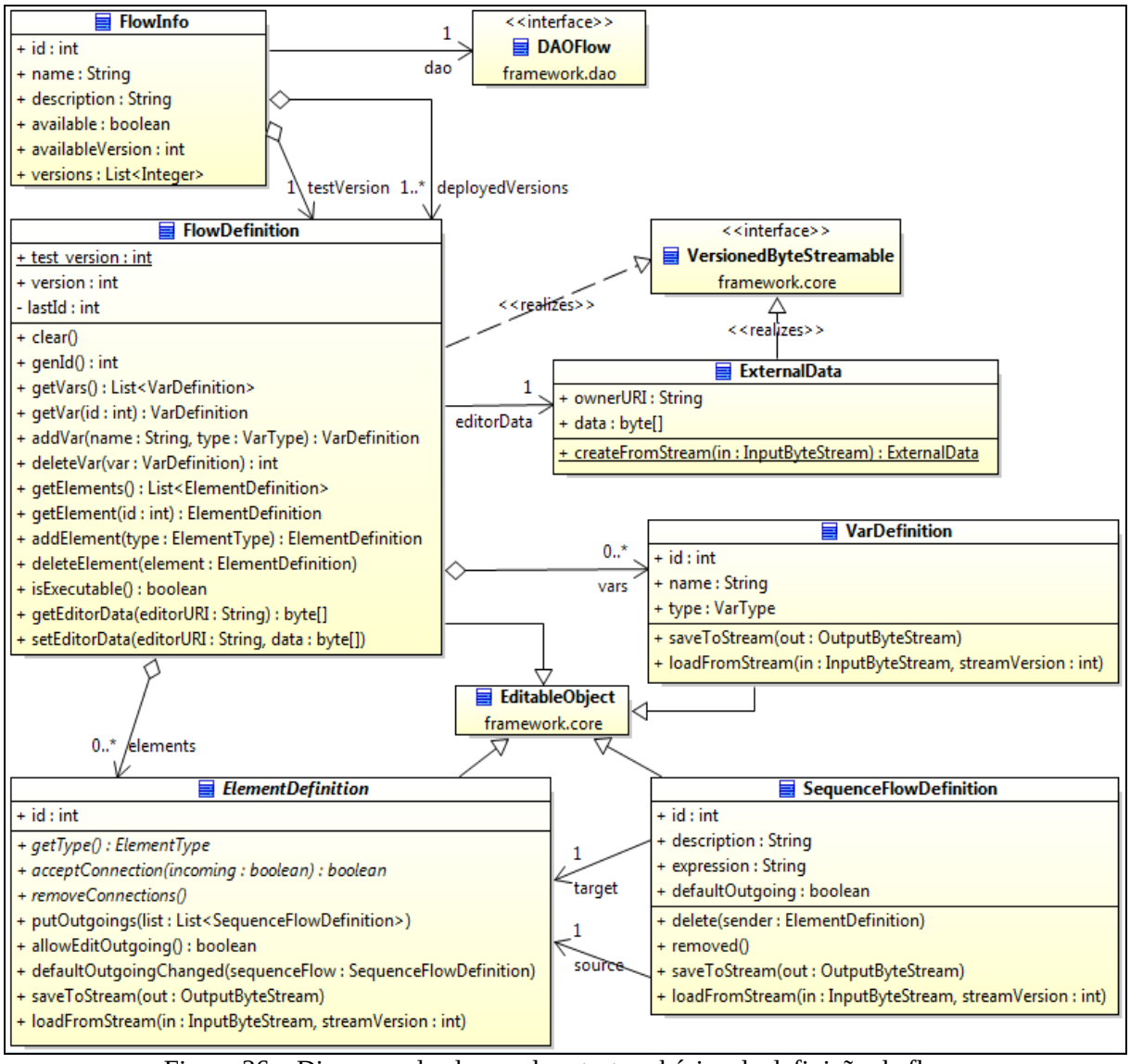


Figura 36 – Diagrama de classes da estrutura básica de definição de fluxo

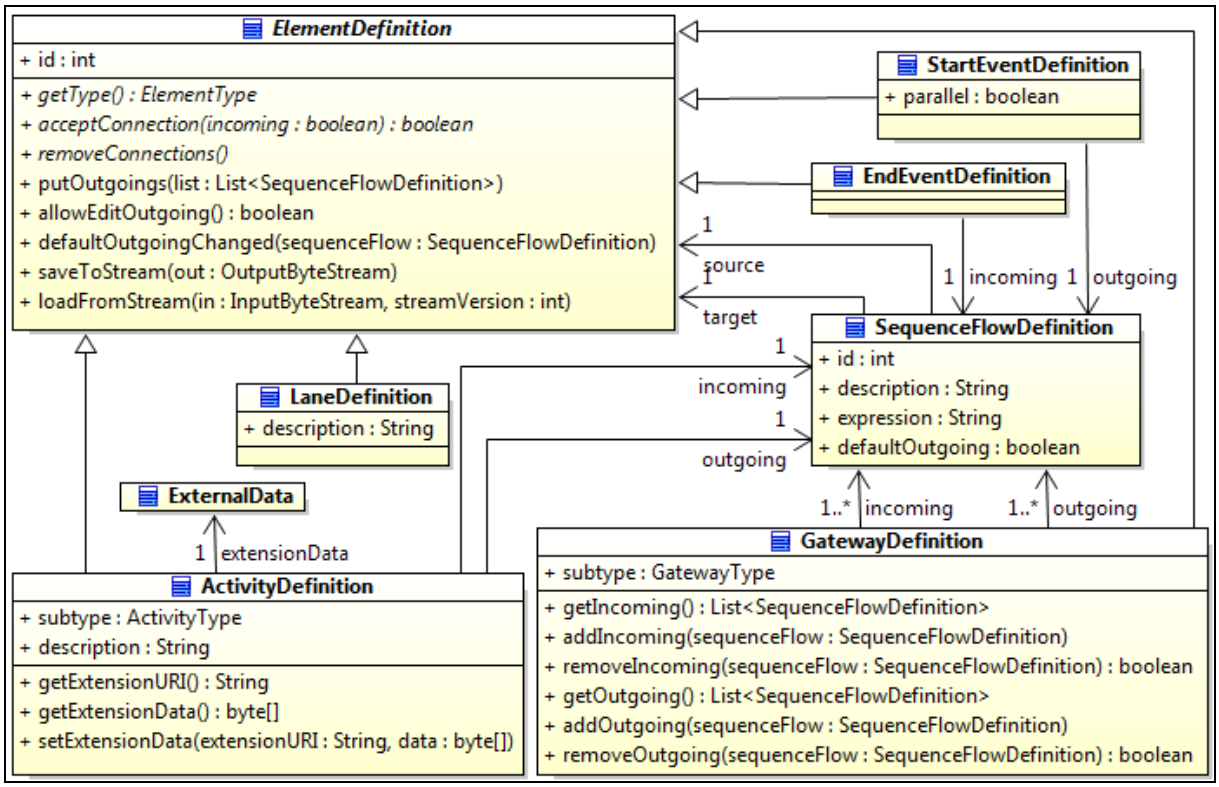


Figura 37 – Diagrama de classes da hierarquia de elementos de definição

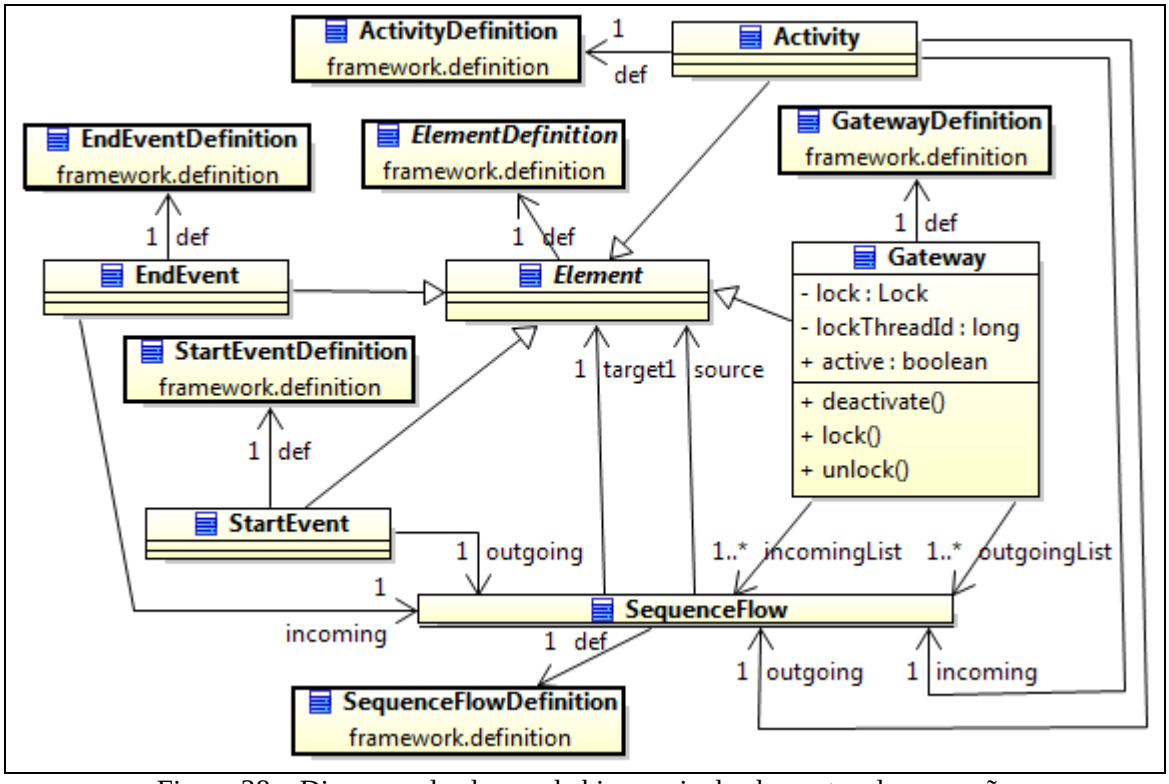


Figura 38 – Diagrama de classes da hierarquia de elementos de execução

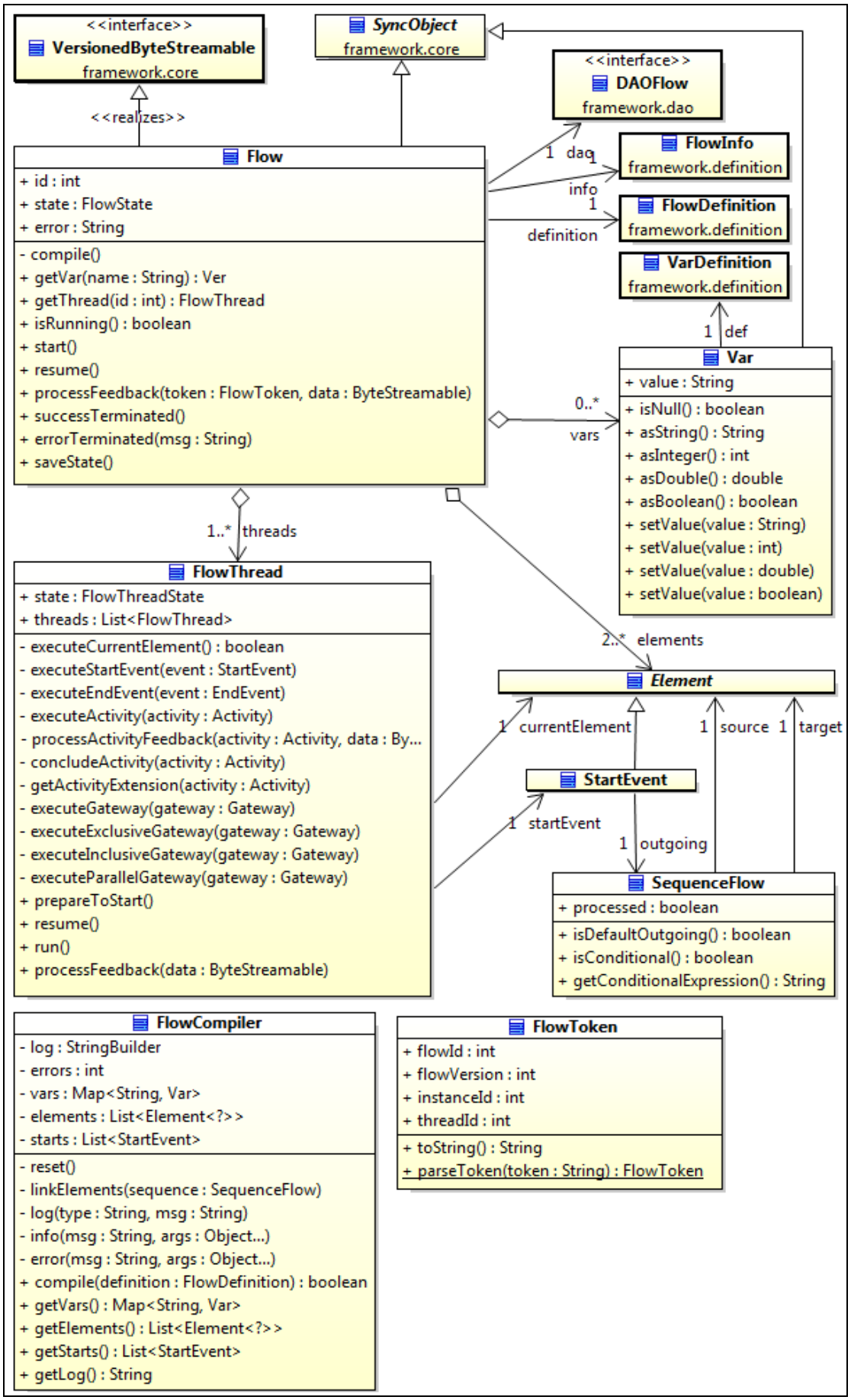


Figura 39 – Diagrama de classes utilizadas para execução de fluxo

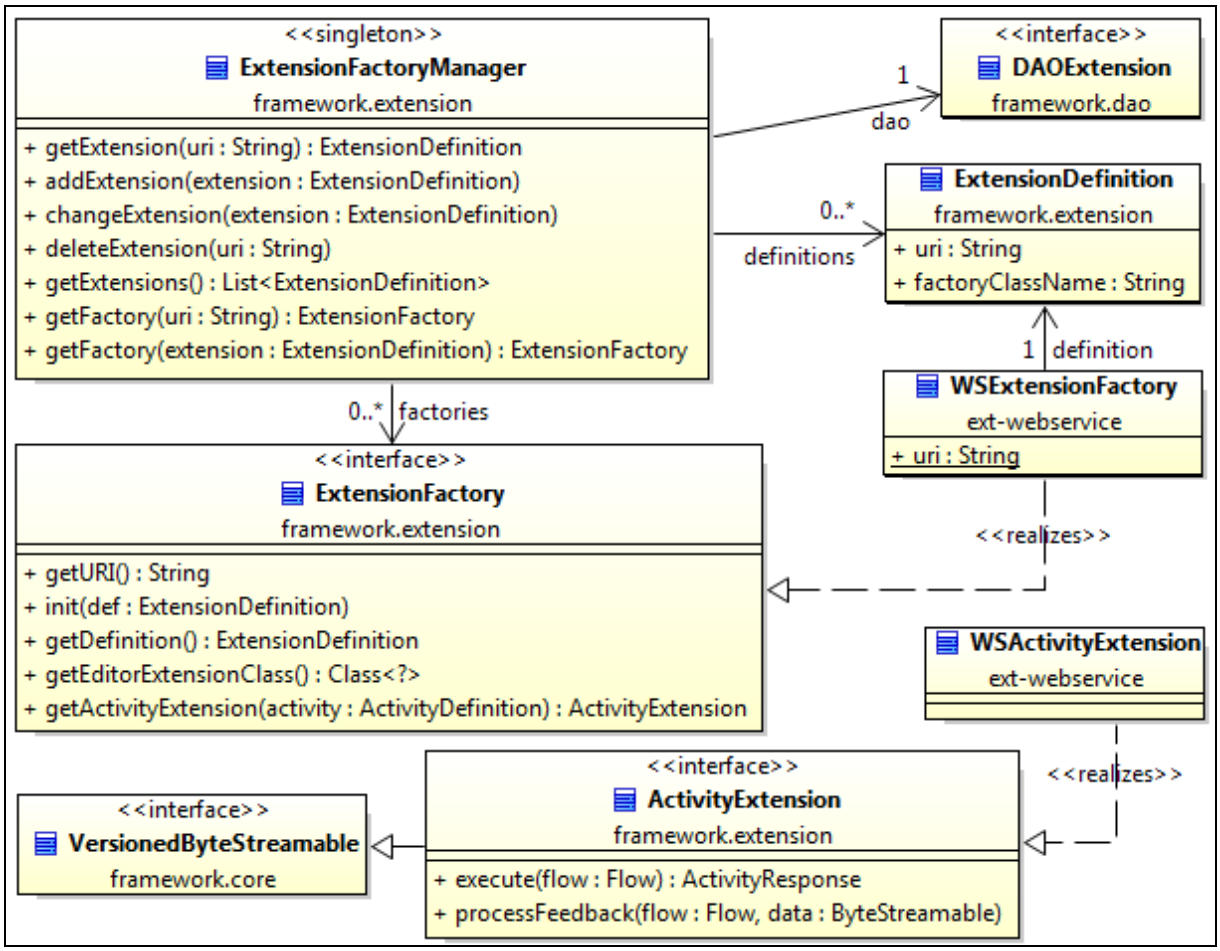


Figura 40 – Diagrama de classes da extensão de *web services* do executor

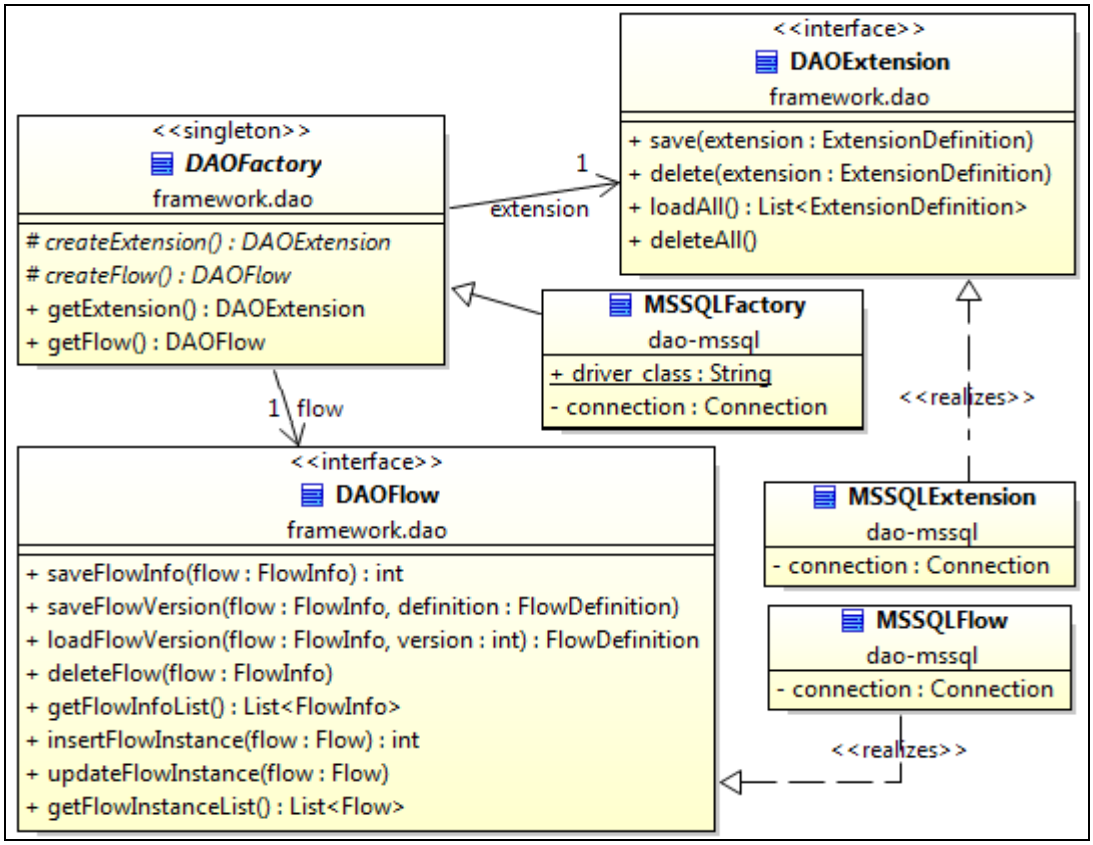


Figura 41 – Diagrama de classes do DAO para SQL *Server*

3.3.6 Diagramas de entidade-relacionamento

O diagrama entidade-relacionamento demonstrado na Figura 42 representa a estrutura de tabelas utilizadas pelo DAO que implementa o suporte ao banco de dados SQL *Server*.

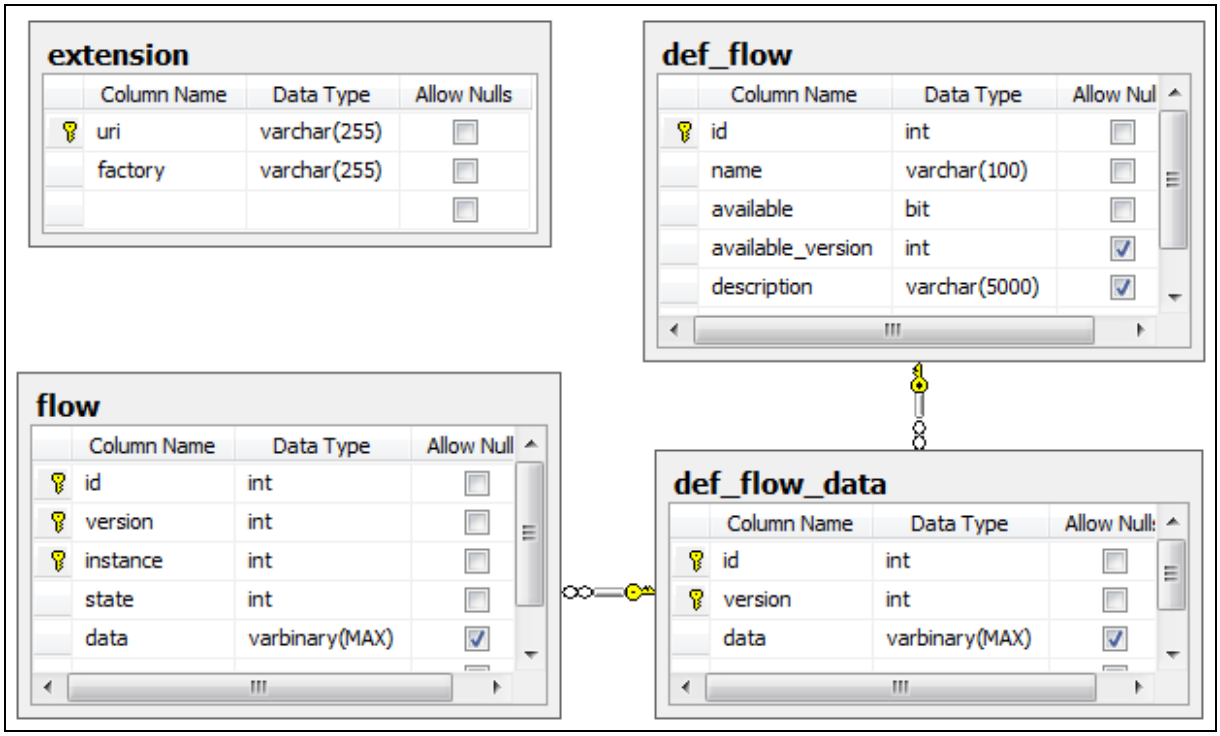


Figura 42 – Diagrama de entidade-relacionamento

Na tabela "*extension*" são armazenados os dados das extensões cadastradas. Na coluna "uri" é armazenado o identificador da extensão. Na coluna "factory" é armazenado o nome da classe da extensão que implementa a interface `isensee.fabio.tcc.framework.ext.ExtensionFactory` do *framework*.

Na tabela "*def_flow*" e "*def_flow_data*" são armazenados os dados dos fluxos modelados através do editor. Para cada fluxo modelado há um registro na tabela "*def_flow*", e para cada versão deste mesmo fluxo há um registro na "*def_flow_data*". Na tabela "*def_flow*", a coluna "*available*" indica se o fluxo está ou não disponível para uso, enquanto a coluna "*available_version*" indica qual é a versão disponível dentre as existentes na tabela "*def_flow_data*". Durante a modelagem, os dados do fluxo são salvos na coluna "*data*" da tabela "*def_flow_data*" cujo valor da coluna "*version*" seja igual a zero. Quando o administrador disponibiliza uma nova versão do fluxo para execução o valor da coluna "*data*" é duplicado, gerando um novo registro da tabela "*def_flow_data*". Esta construção possibilita que o modelo do processo seja alterado sem afetar as instâncias deste mesmo processo que já estão em execução.

A tabela “*flow*” é utilizada para armazenar os dados de execução dos processos. Os dados são serializados e armazenados na coluna “*data*”. Para cada processo iniciado é gerado um novo registro nesta tabela, identificado pela coluna “*instance*”. As colunas “*id*” e “*version*” representam o fluxo e a versão em que ele se encontrava quando ele foi iniciado.

3.4 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.4.1 Técnicas e ferramentas utilizadas

A implementação da ferramenta foi realizada em linguagem Java sobre sistema operacional Windows 7. Foram utilizadas as seguintes ferramentas:

a) Eclipse Java EE, versão Helios *Service Release* 1, como ambiente de desenvolvimento integrado;
b) GlassFish, versão 3.1, como servidor Java EE para publicação e execução dos *web services* e dos aplicativos *web*;
c) Sistema Gerenciador de Banco de Dados (SGBD) Microsoft SQL *Server*, versão 2008 *Release* 2, para persistência dos dados;
d) SQL Server Management Studio para modelagem, criação e manutenção da base de dados;
e) Google Chrome, versão 11.0.696.71, como navegador *web*, além de suas ferramentas para desenvolvedor embutidas que auxiliaram no desenvolvimento das interfaces dos aplicativos;
f) *Ant Build* integrado ao Eclipse para execução dos *scripts* de compilação dos componentes do editor e do executor;
g) Java *Development Kit* (JDK), versão 1.6.0_25 - 64-bit, para execução do Eclipse, do GlassFish e da própria ferramenta desenvolvida;
h) Java *Runtime Environment* (JRE), versão 6.0.250.6 - 32-bit, para execução do *applet* do editor no Google Chrome.

Toda a lógica de modelagem e execução de fluxos de processo foi encapsulada em um *framework*. O *framework* por si só é capaz de executar um fluxo cuja definição é recebida como entrada, passando por todos os passos do processo e avaliando as expressões condicionais para decidir qual caminho tomar. Em sua construção foram utilizadas técnicas de programação concorrente, como *threads* e monitores (vide Quadro 1), para suportar a execução de linhas paralelas dentro de um mesmo fluxo.

```
public class FlowThread implements Runnable {

    public void resume() {
        try {
            if (state == FlowThreadState.processing) {
                if (currentElement == null)
                    throw new RuntimeException(
                            "O elemento corrente está nulo.");

                // inicia uma nova linha de execução
                new Thread(this).start();
            }
          // código omitido por questão de espaço
    }
    private void executeGateway(Gateway gateway) {
        gateway.lock(); // usa monitor para garantir
        try {           // acesso exclusivo ao gateway
            if (gateway.isActive()) {
                switch (gateway.getSubtype()) {
                // código omitido por questão de espaço
                case parallel:
                    executeParallelGateway(gateway);
                // código omitido por questão de espaço
            }
        } finally {
            gateway.unlock(); // libera o acesso ao gateway
        }
    }
     private void executeParallelGateway(Gateway gateway) {
          // código omitido por questão de espaço
            // cria uma thread para cada fluxo de saída
            for (SequenceFlow out : gateway.getOutgoingList()) {
                threads.add(new FlowThread(threads.size() + 1,
                        this, out.getTarget()));
                out.setProcessed();
            }
          // código omitido por questão de espaço
            for (FlowThread t : threads)
                t.resume(); // inicia a execução das threads
        }
```

Quadro 1 – Parte do código que utiliza programação concorrente

A execução das expressões condicionais dos *sequence flows* é realizada com o auxilio da biblioteca Rhino, possibilitando que o *framework* avalie qual caminho seguir ao processar

um *gateway*. O Rhino, biblioteca fornecida pela Mozilla, é uma implementação *open-source* do JavaScript escrita totalmente em Java que possibilita a execução de *scripts* escritos por usuários. O código que processa o *script* da expressão condicional é apresentado no Quadro 2.

```
public class ExpressionEvaluator {

 public boolean evaluate(String expression) {
     ConditionEvaluate eval = new ConditionEvaluate(false);
     // cria o contexto de execução do script
     Context ctx = Context.enter();
     try {
         // põe os objetos "eval" e "flow" no escopo do script:
         // - "eval" possibilita atribuir o resultado da condição;
         // - "flow" provê acesso às variáveis do fluxo.
         Scriptable scope = ctx.initStandardObjects();
         Object evalJS = Context.javaToJS(eval, scope);
         ScriptableObject.putConstProperty(scope, "eval", evalJS);
         Object flowJS = Context.javaToJS(flowWrapper, scope);
         ScriptableObject.putConstProperty(scope, "flow", flowJS);
         // executa o script da expressão
         ctx.evaluateString(scope,expression,"<expression>",1,null);
     } finally {
         Context.exit();
     }
     return eval.getResult();
 }
```

Quadro 2 – Código de avaliação das expressões condicionais

Devido ao nível de abstração do *framework*, que busca aumentar sua reusabilidade, a responsabilidade pelo processamento das *activities* que exigem a execução de alguma tarefa é delegada para suas extensões (vide diagrama de atividade apresentado na Figura 23). Para isto, o *framework* disponibiliza um conjunto de interfaces estáveis que possibilitam a criação de extensões que implementam os protocolos de comunicação com os sistemas externos que interagirão com ele. Utilizando este conjunto de interfaces, foi implementado um componente que utiliza recursos de *web services* para interagir com outros sistemas através do protocolo SOAP. Este componente permite tanto invocar *web services* quanto receber notificações, o que foi implementado da seguinte maneira:

a) para invocar *web services*, o componente utiliza a biblioteca Rhino para processar o *script* escrito pelo modelador na *activity* correspondente. Neste *script* é feita uma chamada ao *stub*[3] do *web service* a ser invocado. Por questão de escopo, não foi implementada a rotina que gera os *stubs*, portanto é necessário que eles sejam previamente gerados e disponibilizados com este componente;

[3] *Stub* é o código cliente gerado a partir da WSDL de um *web service*. Este código encapsula toda a parte de comunicação que invoca o *web service* e processa o seu retorno.

b) para receber notificações, o componente disponibiliza um *web service* implementado com os recursos disponibilizados pelo GlassFish. O Quadro 3 ilustra a parte deste *web service* que recebe a notificação que indica quando uma tarefa terminou de ser processada.

```
import javax.ejb.Remote;
import javax.ejb.Stateless;
import javax.ejb.TransactionAttribute;
import javax.ejb.TransactionAttributeType;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;

@Stateless
@Remote(FeedbackRemote.class)
@WebService(serviceName = "flow-services", name = "Feedback",
        targetNamespace = "http://fabio.isensee.com.br")
@SOAPBinding(style = SOAPBinding.Style.RPC)
public class Feedback implements FeedbackRemote, FeedbackLocal {

    @WebMethod
    @TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
    @WebResult(name = "result")
    public String continueFlow(
            @WebParam(name="token") String token,
            @WebParam(name="params") ServiceParam[] params) {
```

Quadro 3 – Parte do código de implementação de *web service*

A implementação da área de modelagem do editor utiliza a tecnologia de *applets* com a API do Java 2D, permitindo a renderização bidimensional dos elementos da BPMN em um navegador *web*. O Quadro 4 mostra a declaração da classe principal e do método de inicialização do *applet* do editor. Em seguida, no Quadro 5, o uso de Java 2D é demonstrado através do código responsável por desenhar o retângulo que representa graficamente o elemento *activity*. O restante da tela do editor, dentro da qual o *applet* está contido, é implementado em HTML e JavaScript, bem como todas as telas do console de administração. As páginas HTML são geradas dinamicamente por *servlets*, que foi a técnica aplicada para fazer toda a parte de comunicação da camada cliente (que executa no navegador *web*) com a camada servidora (que executa no GlassFish). O Quadro 6 demonstra parte do código do *servlet* que atende as requisições do *applet* do editor, fornecendo acesso às informações disponíveis no servidor.

```
public class EditorApplet extends JApplet {

    @Override
    public void init() {
```

Quadro 4 – Parte do código da classe principal do *applet* do editor

```
@Override
public void draw(Graphics2D graphics) {
    rectangle = new RoundRectangle2D.Double(
            left, top, width, height, 10, 10);
    graphics.setPaint(fill_color);
    graphics.fill(rectangle);
    graphics.setStroke(thin_line);
    graphics.setPaint(border_color);
    graphics.draw(rectangle);
```

Quadro 5 – Parte do código de renderização da *activity*

```
@SuppressWarnings("serial")
public class AppletServlet extends HttpServlet {

    public static final String url_mask = ".apl";

    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        PrintWriter writer = resp.getWriter();

        try {
            String pageName = req.getRequestURI();
            pageName = pageName.substring(0, pageName.indexOf(url_mask));
            pageName = pageName.substring(pageName.lastIndexOf('/') + 1);
            RequestProcessor processor;

            if ("model".equals(pageName))
                processor = new ModelProcessor(req);
```

Quadro 6 – Parte do código do *servlet* que atende as requisições do *applet*

Para auxiliar no entendimento da implementação, a Figura 43 representa a ferramenta em uma estrutura de camadas que contempla as tecnologias e ferramentas mencionadas até o momento nesta seção. Nesta abordagem o *framework* é representado dentro do GlassFish, pois assim foi utilizado neste trabalho. Porém, o que prende a ferramenta ao GlassFish são os *web services*, *servlets* e páginas HTML. O *framework* pode ser utilizado fora de servidores Java EE, desde que seja estendido de maneira adequada.

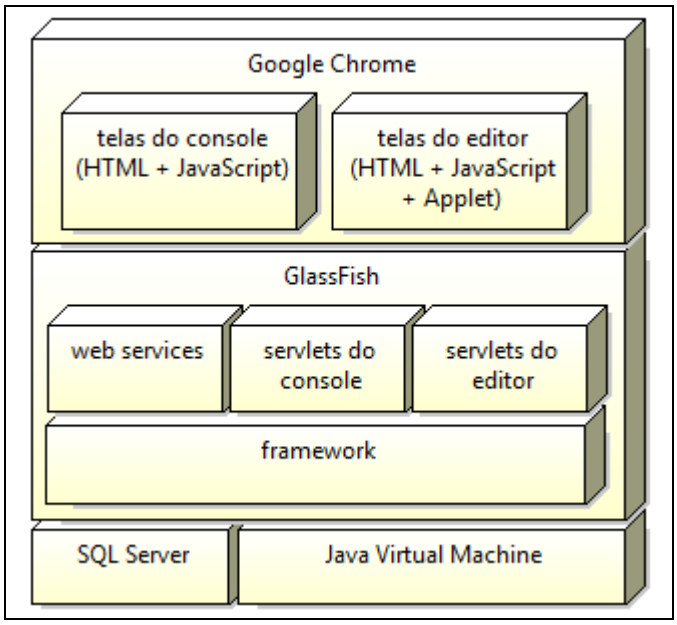


Figura 43 – Representação das camadas da ferramenta

Durante o desenvolvimento foram aplicados os seguintes padrões de projeto:

a) *Data Access Object* (DAO): a utilização deste padrão possibilitou total desacoplamento da ferramenta a algum SGBD ou formato de arquivo específico. Para realização dos testes e estudo de caso foi desenvolvido o DAO que utiliza a API Java *DataBase Connectivity* (JDBC) para persistir os dados em SQL *Server*, cuja implementação está disponível no componente "dao-mssql" descrito na seção 3.3.4. Para utilizar outro SGBD ou ainda persistir os dados em arquivos, basta criar uma extensão que realize as interfaces `DAOExtension` e `DAOFlow` e estenda a classe abstrata `DAOFactory`, todas disponíveis no pacote `isensee.fabio.tcc.framework.dao` do *framework*, sendo que a classe que estende a `DAOFactory` deve ser indicada na configuração "dao.factory". O Quadro 7 demonstra o código responsável por instanciar a classe derivada de `DAOFactory` conforme configuração. O Quadro 8 demonstra o código da interface `isensee.fabio.tcc.framework.dao.DAOFlow`, a qual é realizada pela classe `isensee.fabio.tcc.mssql.MSSQLFlow` ilustrada no Quadro 9;

```
private static DAOFactory instance = null;

public static synchronized DAOFactory getInstance() {
    if (instance == null) {
        String name = ConfigFile.getProperty("dao.factory", true);
        try {
            Class<?> clazz = Class.forName(name);
            instance = (DAOFactory) clazz.newInstance();
```

Quadro 7 – Código de inicialização do *factory* de DAO

```
public interface DAOFlow {

    int saveFlowInfo(FlowInfo flow);

    void saveFlowVersion(FlowInfo flow, FlowDefinition definition);

    FlowDefinition loadFlowVersion(FlowInfo flow, int version);

    void deleteFlow(FlowInfo flow);

    List<FlowInfo> getFlowInfoList();

    int insertRunningFlow(Flow flow);

    void updateRunningFlow(Flow flow);

}
```

Quadro 8 – Código da interface de persistência de dados dos fluxos

```
public class MSSQLFlow implements DAOFlow {

    public MSSQLFlow(Connection connection) {
        this.connection = connection;
    }
    @Override
    public List<FlowInfo> getFlowInfoList() {
        try {
            List<FlowInfo> list = new ArrayList<FlowInfo>();

            if (psSelAllFI == null)
                psSelAllFI = connection.prepareStatement(
                    "select id, name, available, available_version, " +
                    "description from def_flow order by id");

            if (psSelVerFD == null)
                psSelVerFD = connection.prepareStatement(
                    "select version from def_flow_data " +
                    "where id = ? order by version");

            ResultSet rs = psSelAllFI.executeQuery();
            try {
                while (rs.next()) {
                    FlowInfo flow = new FlowInfo();
                    flow.setId(rs.getInt(1));
```

Quadro 9 – Código de leitura dos dados dos fluxos do SQL *Server*

b) *Singleton*: este padrão foi aplicado para manter em memória os objetos que são utilizados durante toda a execução da ferramenta, aumentando sua performance por não ter que instanciá-los a cada utilização. As seguintes classes foram implementadas utilizando este padrão:

- `isensee.fabio.tcc.framework.core.ConfigFile`: provê acesso às configurações da ferramenta,

- `isensee.fabio.tcc.framework.dao.DAOFactory`: *factory* de DAO descrito no item a),
- `isensee.fabio.tcc.framework.ext.ExtensionFactoryManager`: gerencia e provê acesso às extensões do *framework*,
- `isensee.fabio.tcc.framework.flow.FlowManager`: gerencia e provê acesso aos fluxos de processos disponíveis para edição e execução,
- `isensee.fabio.tcc.editor.ext.EditorExtensionManager`: provê acesso às extensões carregadas no editor,
- `isensee.fabio.tcc.editor.util.ClientConnection`: disponibiliza as rotinas utilizadas para fazer a comunicação via HTTP do *applet* do editor que com a camada que é executada no servidor;

c) *Abstract Method*: a tarefa de desenhar os elementos da BPMN no editor exige várias linhas de código, pois envolve a utilização de algoritmos geométricos como distância entre pontos e *bounding box*[4], e outras técnicas de computação gráfica como *text anti-aliasing*[5] e rotação e translação de objetos. Para evitar a criação de uma rotina ou classe única que concentre todos estes algoritmos, o que tornaria o código difícil de entender e dar manutenção, foram criados métodos abstratos nas classes base de todos os componentes modelados no editor para que cada um deles seja responsável por se desenhar e prover informações para interação com a janela de edição. O Quadro 10 demonstra os métodos abstratos da classe `isensee.fabio.tcc.editor.applet.ModelElement`, da qual derivam as classes dos elementos da BPMN. O Quadro 11 demonstra o a implementação destes métodos na classe `isensee.fabio.tcc.editor.applet.ModelGatewayElement`, a qual representa os *gateways*;

```
public abstract class ModelElement<T extends ElementDefinition>
    extends ModelComponent implements ByteStreamable {

    protected abstract void moved();

    protected abstract Rectangle getBoundBox();

    public abstract void draw(Graphics2D graphics);
```

Quadro 10 – Métodos abstratos da classe base dos elementos da BPMN

[4] Caixa invisível que envolve um objeto gráfico, utilizada para determinar seu tamanho e posição na tela.
[5] Filtro que diminui os serrilhados no contorno dos objetos gráficos e suavizam a transição visual destes objetos com o ambiente ao seu redor.

```
@Override
protected void moved() {
    for (SequenceFlowDefinition sf : definition.getIncoming())
        owner.getSequenceFlow(sf).updateConnections();

    for (SequenceFlowDefinition sf : definition.getOutgoing())
        owner.getSequenceFlow(sf).updateConnections();
}
@Override
protected Rectangle getBoundBox() {
    return new Rectangle(left, top, size, size);
}
@Override
public void draw(Graphics2D graphics) {
    polygon = new GeneralPath(GeneralPath.WIND_EVEN_ODD, 4);
    polygon.moveTo(left, top + half_size);
    polygon.lineTo(left + half_size, top);
    // continua...
```

Quadro 11 – Implementação dos métodos abstratos na classe do elemento *gateway*

d) *Abstract Factory*: este padrão foi utilizado como base para a implementação dos pontos de extensão do *framework*. Para que uma extensão possa ser utilizada pelo *framework*, é necessário que ela realize um conjunto de interfaces. Uma delas é a `ExtensionFactory`, demonstrada no Quadro 12, que provê acesso às funcionalidades da extensão referentes a cada recurso que pode ser estendido. O Quadro 13 mostra a parte da implementação da classe que realiza a `ExtensionFactory` na extensão de *web services*. Quando o *framework* executa algum ponto que pode ser estendido, ele solicita ao `ExtensionFactoryManager` quais são as extensões registradas. Quando necessário executar um recurso implementado na extensão, é solicitada uma instância do *factory* da extensão, a qual é criada a partir do nome da classe que foi registrada através de reflexão computacional, conforme demonstrado no Quadro 14.

```
package isensee.fabio.tcc.framework.ext;
public interface ExtensionFactory {

    ActivityExtension getActivityExtension(ActivityDefinition activity);
```

Quadro 12 – Parte da interface do *factory* de extensões

```
package isensee.fabio.tcc.webservice.ext;

import isensee.fabio.tcc.framework.ext.ExtensionFactory;

public class WSExtensionFactory implements ExtensionFactory {

    @Override
    public ActivityExtension getActivityExtension(
            ActivityDefinition activity) {
        InputByteStream stream =
            new InputByteStream(activity.getExtensionData());
        try {
            return (WSActivityExtension)
                stream.loadObject(new WSActivityExtension());
        } catch (IOException e) {
            throw new RuntimeException("Erro carregando extensão de " +
                    activity + ": " + e.getMessage(), e);
        }
    }
```

Quadro 13 – Parte da implementação do *factory* da extensão de *web services*

```
package isensee.fabio.tcc.framework.ext;

import isensee.fabio.tcc.framework.dao.DAOExtension;

public class ExtensionFactoryManager {

    public ExtensionFactory getFactory(ExtensionDefinition extension) {
        ExtensionFactory fac = factories.get(extension);

        if (fac == null) {
            String className = extension.getFactoryClassName();
            Class<?> facClass;
            try {
                facClass = Class.forName(className);
            } catch (ClassNotFoundException e) {
                throw new RuntimeException("Não foi possível carregar " +
                        "o factory da extensão porque a classe \"" +
                        className + "\" não foi encontrada.");
            }
            try {
                fac = (ExtensionFactory) facClass.newInstance();
                fac.init(extension);
                factories.put(extension, fac);
            } catch (Exception e) {
                throw new RuntimeException("Erro carregar o " +
                        "factory da extensão: " + e.getMessage(), e);
            }
        }
        return fac;
    }
```

Quadro 14 – Código de inicialização dos *factories* das extensões

3.4.2 Operacionalidade da implementação

O fluxo ilustrado na Figura 44 foi modelado utilizando o editor desenvolvido neste trabalho para representar o processo de utilização da ferramenta. Cada *lane* representa um tipo de usuário. As *activities* contidas em cada *lane* representam as tarefas realizadas pelo usuário correspondente, considerando:

a) o usuário modelador utiliza o editor de processos para modelar os fluxos;
b) o usuário administrador utiliza o console de administração para tornar os processos disponíveis ou indisponíveis para utilização;
c) o usuário final opera um ou mais sistemas externos, os quais interagem com o executor de processos, para utilizar os processos de negócio.

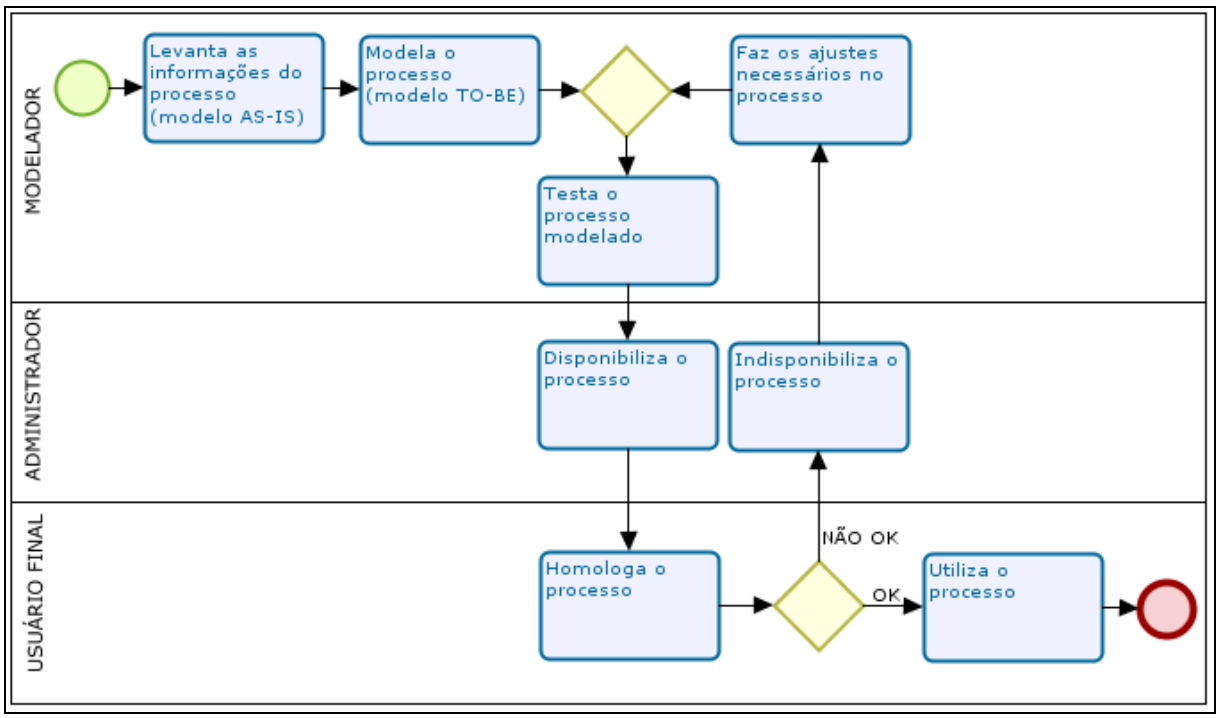


Figura 44 – Fluxo do processo de utilização da ferramenta

A automatização dos processos inicia-se com o levantamento das informações para criação do modelo *AS-IS* do processo, o qual representa o fluxo do processo na sua situação atual. Tendo o modelo *AS-IS*, o modelador verifica os pontos que podem ser melhorados no processo e cria o modelo *TO-BE*, o qual representa o fluxo do processo como ele deve ser. O modelo *TO-BE* é automatizado e testado. Depois que o fluxo modelado foi devidamente testado pelo modelador, o administrador disponibiliza o processo para que os usuários finais o homologuem. Se o processo estiver funcionado corretamente ele é mantido em produção para ser utilizado. Caso contrário, o administrador torna o processo indisponível para que nenhum

usuário possa executá-lo novamente na versão defeituosa. Após o modelador fazer os ajustes necessários e os testes forem executados com sucesso, o administrador disponibiliza uma nova versão para que os usuários finais iniciem um novo ciclo de homologação. Este processo de ajustes, testes e homologação é repetido até que o fluxo do processo esteja funcionando adequadamente.

O funcionamento da ferramenta é demonstrado aplicando o fluxo ilustrado anteriormente no estudo de caso que simula uma situação real em uma empresa. Para a simulação, foi desenvolvido o aplicativo chamado "ApTeste" para representar os sistemas externos que interagem com o executor de processos. Em seguida foi feita a automatização do processo de aumento salarial, desde a modelagem, passando pela sua disponibilização até a sua utilização. Por fim, foi feita uma manutenção no processo de aumento salarial e a disponibilização de uma nova versão.

3.4.2.1 Aplicativo de simulação "ApTeste"

O ApTeste representa os sistemas externos que interagem com o executor de processos, atuando nos seguintes papéis:

a) sistema de recursos humanos: sistema legado que contém regras de negócio básicas de uma solução de recursos humanos. A interação com os usuários é feita através de telas que implementam as regras de negócio;
b) máquina de *workflow*: disponibiliza as seguintes telas para o usuário final:
   - autenticação: solicita o nome do usuário ao acessar o aplicativo,
   - lista de processos: lista de todos os processos disponíveis para que eles possam ser iniciados,
   - lista de tarefas: lista de todas as tarefas pendentes para o usuário que acessou o aplicativo. O clique sobre uma tarefa abre a tela correspondente no sistema de recursos humanos.

O sistema de recursos humanos foi modificado para que as regras de negócio pudessem ser reaproveitadas pela máquina de *workflow*, da seguinte maneira:

a) foram criadas telas de entrada de informações que até então eram trocadas por e-mails ou malotes de papel;
b) os dados do cadastro de funcionários foram disponibilizados para obtenção através de *web services*;

c) quando uma tela é aberta através da máquina de *workflow*, o clique no botão "Gravar" invoca o *web service* do executor de processos que notifica que a tarefa que originou a abertura da tela foi tratada.

3.4.2.2 Modelagem do processo de aumento salarial

A automatização de um processo de negócio é iniciada mapeando como o processo funciona até então. Em seguida é feito um estudo para identificar melhorias que podem ser feitas no processo para otimizá-lo ou enquadrá-lo à realidade da empresa. Após esse levantamento e as melhorias identificadas, definiu-se que o processo de aumento salarial deve funcionar da seguinte maneira:

a) o funcionário que deseja um aumento reúne-se com o líder de sua célula e solicita o aumento, apresentando seus argumentos;
b) caso o líder concorde com o aumento, ele deve iniciar um processo de aumento salarial informando o código do funcionário, o percentual de aumento e as justificativas;
c) solicitação de aumento é então encaminhada ao gerente da área, o qual tem autonomia de conceder aumentos de até 10%;
d) se o gerente aprovar o aumento, considera-se o critério de autonomia:
   - se o aumento for menor ou igual a 10%, a solicitação é enviada diretamente à área de recursos humanos,
   - senão, a solicitação é enviada ao diretor do setor;
e) se o gerente não aprovar o aumento, ele sugere um novo percentual e informa o motivo. Neste caso:
   - a solicitação volta para o líder da célula, o qual avalia a sugestão do gerente,
   - se o líder aceitar a sugestão, a solicitação é encaminha novamente ao gerente para que ele a aprove,
   - se o líder não aceitar a sugestão, ele cancela a solicitação e informa ao seu liderado que o aumento não foi aprovado;
f) caso a solicitação chegue ao diretor, este a avalia e pode aprová-la ou não:
   - se for aprovada, a solicitação é encaminhada à área de recursos humanos,
   - senão, é feito o mesmo procedimento descrito no item e);

g) quando a solicitação é aprovada, os seguintes procedimentos são realizados para concluí-la:

- a área de recursos humanos recebe a solicitação de aumento aprovada para fazer as tratativas legais e aplicar o reajuste ao cadastro do funcionário,
- o líder do funcionário, o qual abriu a solicitação, recebe a tarefa de notificar o aumento ao funcionário.

Com base na definição do processo, o modelador acessa o editor de processos e cria um novo fluxo, conforme ilustrado na Figura 45, e modela o processo conforme ilustrado na Figura 46.

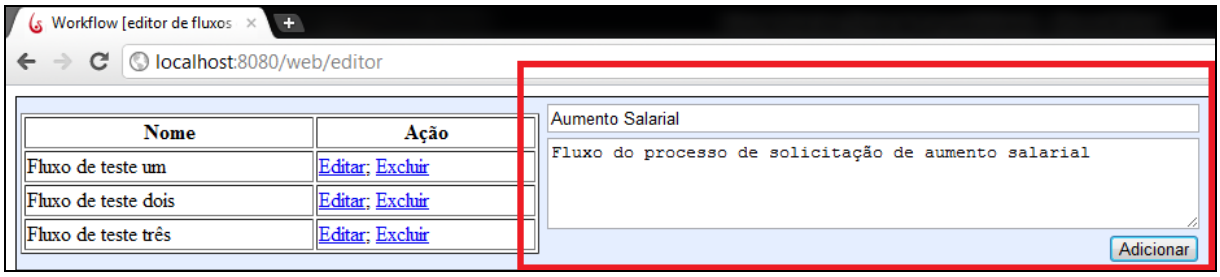


Figura 45 – Criação do fluxo de aumento salarial

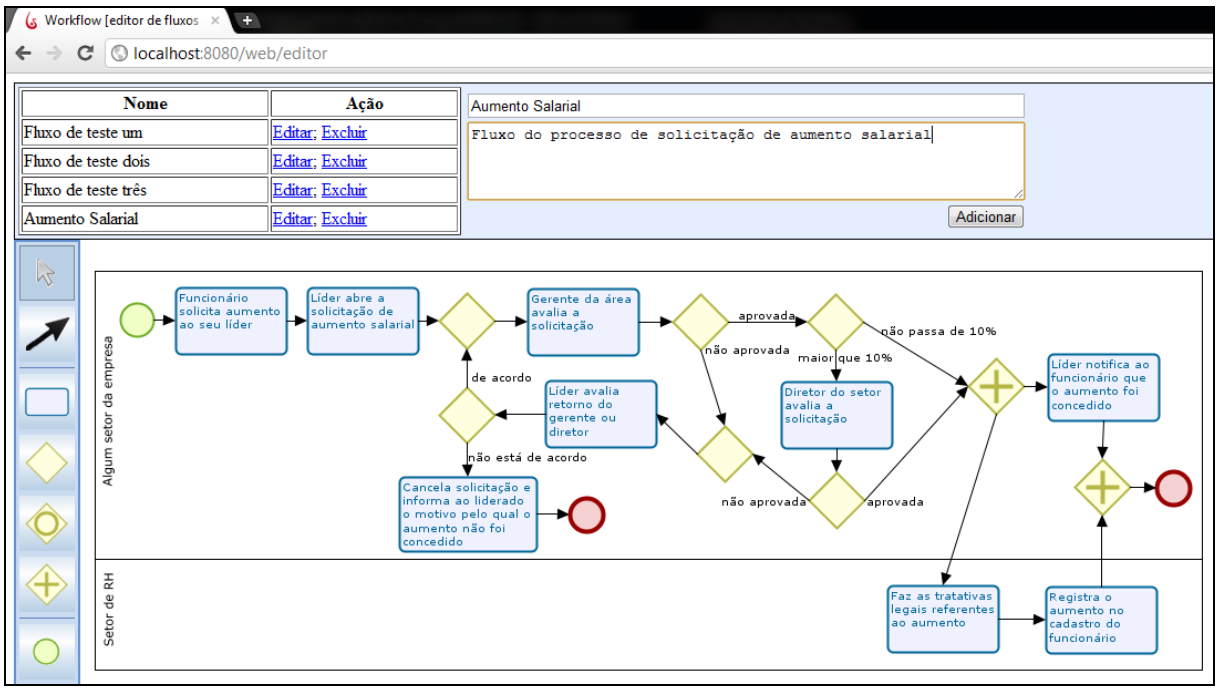


Figura 46 – Modelagem do fluxo de aumento salarial

Após ter o fluxo modelado e aprovado é feito o processo de automatização. Com o fluxo aberto no editor, o modelador realiza as seguintes operações:

a) define as variáveis utilizadas na execução do processo, conforme Figura 47;

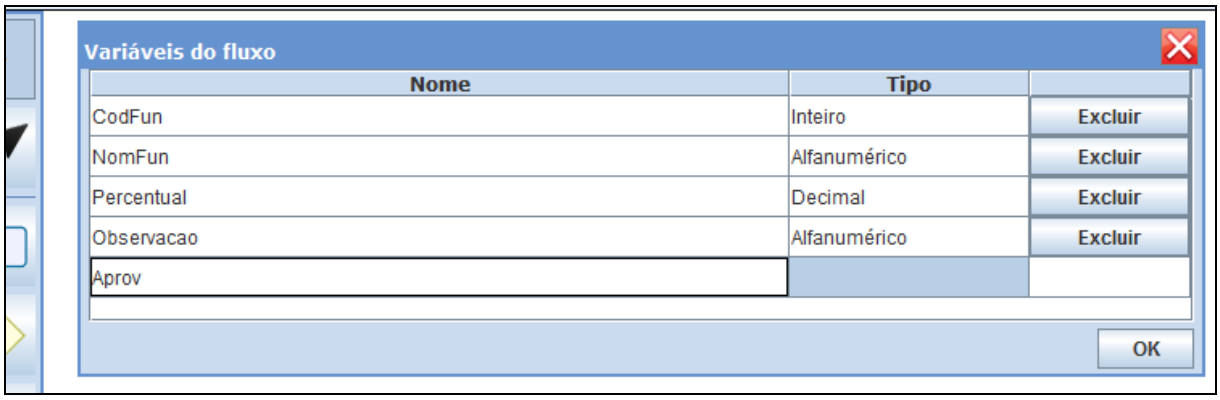


Figura 47 – Edição de variáveis do fluxo de aumento salarial

b) define as expressões de validação dos *sequence flows* dos *gateways* divergentes, conforme Figura 48 e Figura 49;

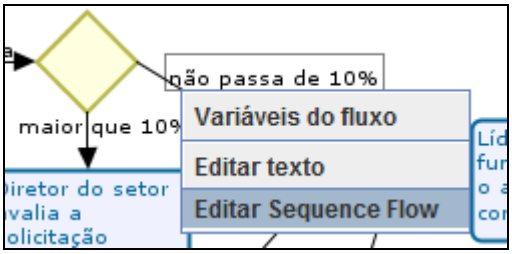


Figura 48 – Menu de edição de *sequence flow*

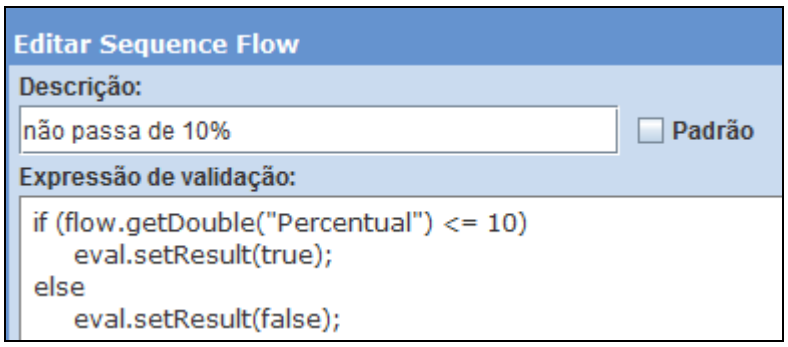


Figura 49 – Edição da expressão condicional do *sequence flow*

c) define os *scripts* de execução das *activities* que exigem processamento, onde são executados os *web services* do ApTeste passando os valores de entrada e processando os de retorno, conforme Figura 50 e Figura 51.

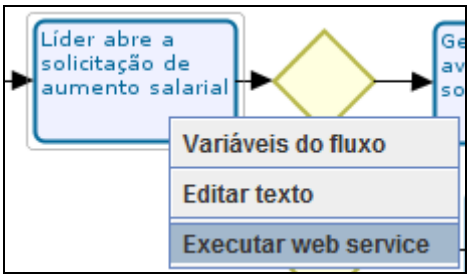


Figura 50 – Menu de edição de *activity*

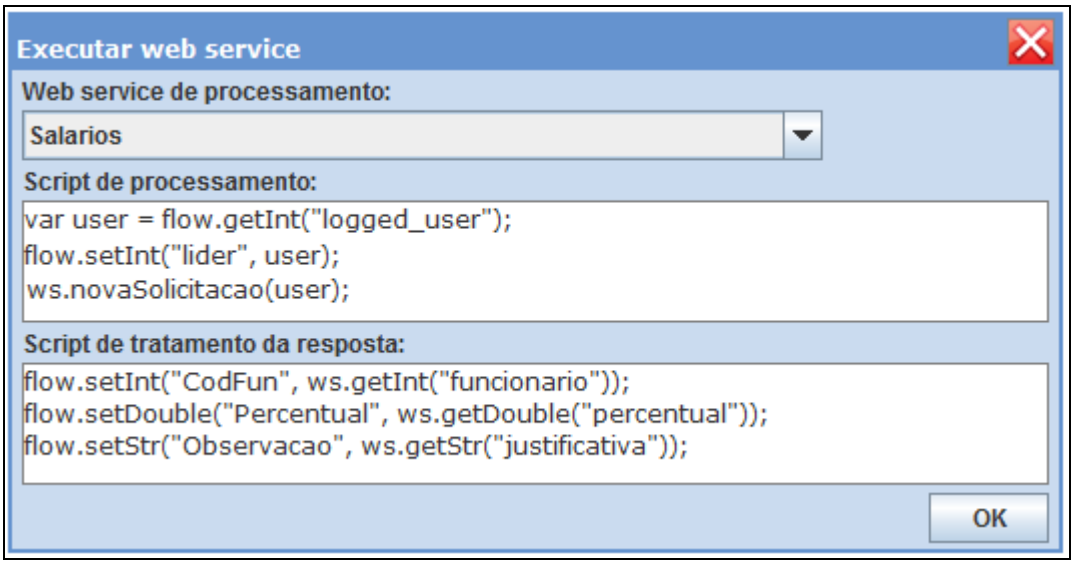


Figura 51 – Edição de *scripts* de execução de *web service*

O fluxo modelado e automatizado é então testado. Com o editor aberto, o modelador seleciona o fluxo e clica na opção "Testar". A execução do processo é iniciada e o modelador passa a utilizar o processo através do ApTeste simulando as possíveis ações dos usuários finais descritas na seção 3.4.2.4.

3.4.2.3 Disponibilização do processo de aumento salarial

Após modelado e devidamente testado, o fluxo é disponibilizado para ser utilizado pelos usuários. A disponibilização é feita pelo administrador através da tela de fluxos existentes disponível no console de administração. O administrador seleciona o fluxo de aumento salarial e clica na opção "Disponibilizar", conforme Figura 52. Esta mesma tela permite que o administrador torne o fluxo indisponível, clicando na opção "Indisponibilizar", caso os usuários finais identifiquem inconsistências durante a fase de homologação.

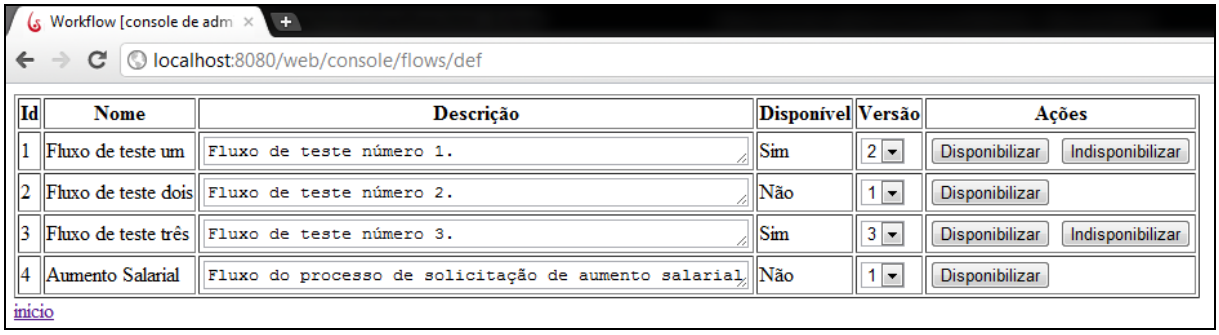


Figura 52 – Tela do console de administração que mostra os fluxos existentes

3.4.2.4 Utilização do processo de aumento salarial

A utilização do fluxo do processo de aumento salarial é demonstrada a seguir através da simulação de uma solicitação de aumento. Por se tratar de um aplicativo criado apenas para a realização do estudo de caso, as telas do ApTeste são ilustradas apenas quando forem importantes para o entendimento da simulação. Ao final desta simulação é exibido o log de execução gerado pelo ApTeste, demonstrando as interações feitas com os usuários e com o executor de processos.

Para simular a execução, considera-se que os funcionários e usuários estejam cadastrados no ApTeste, formando a hierarquia ilustrada na Figura 53. Além desta hierarquia, há ainda o usuário "Marge", definido como responsável por tratar os aumentos salariais aprovados. A simulação procede da seguinte maneira:

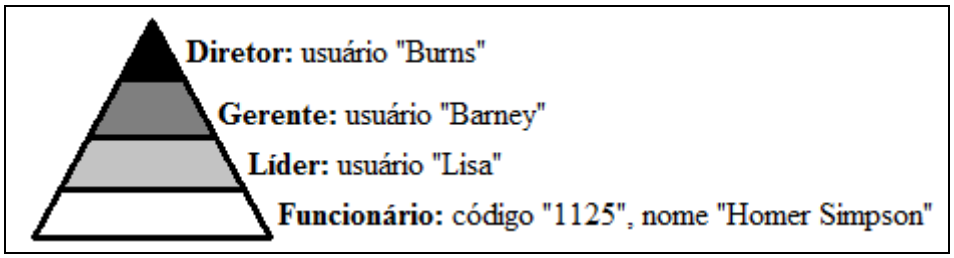


Figura 53 – Hierarquia de usuários para simulação do fluxo

a) o funcionário Homer Simpson solicita um aumento para seu líder. Concordando com a solicitação, o líder acessa o ApTeste e inicia um processo de aumento salarial clicando no botão "Iniciar", ilustrado na Figura 54;

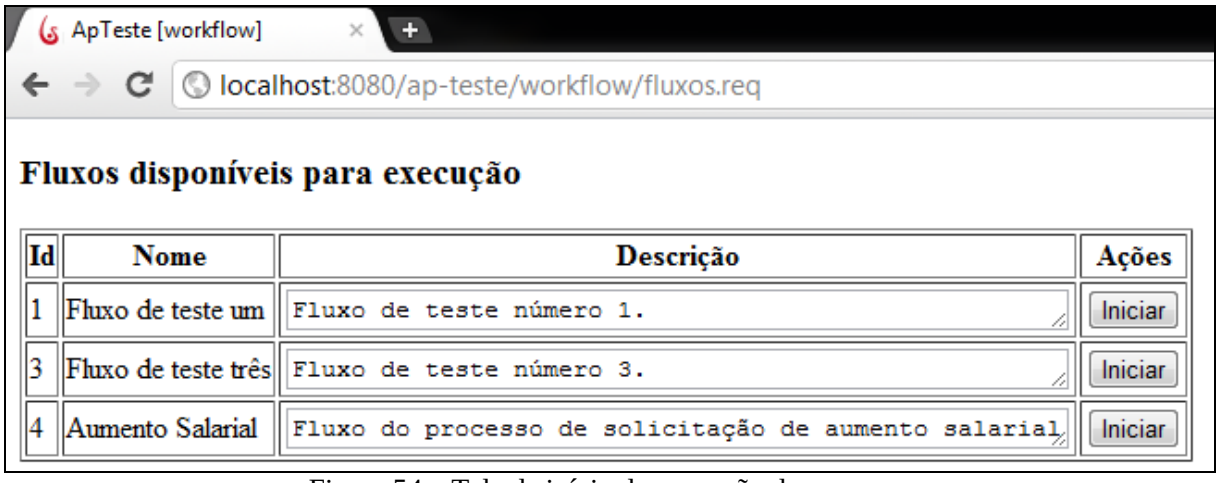


Figura 54 – Tela de início de execução de processo

b) o líder acessa sua lista de tarefas pendentes e visualiza o registro correspondente à tarefa gerada pela ferramenta para que ele insira os dados da solicitação, conforme ilustrado na Figura 55;

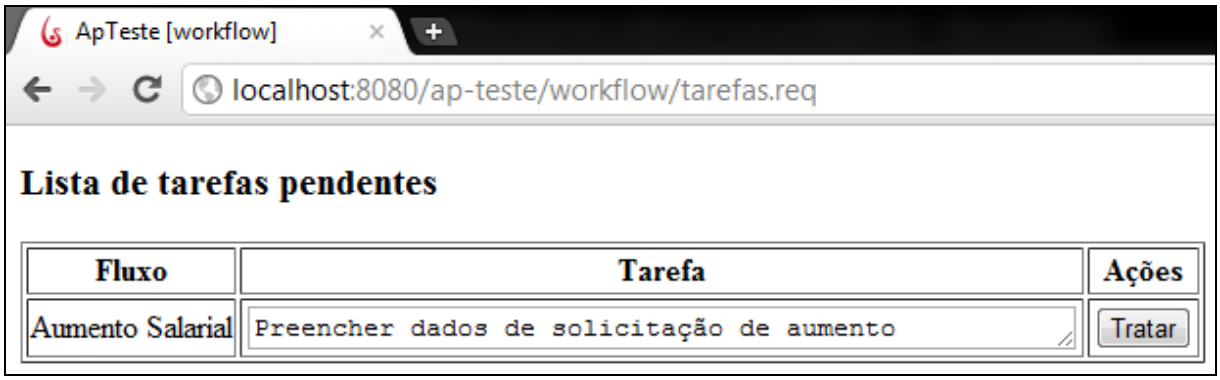


Figura 55 – Tela de visualização de tarefas

c) o líder clica no botão “Tratar” da tarefa, fazendo com que seja aberta a tela na qual ele insere os dados demonstrados na Figura 56;

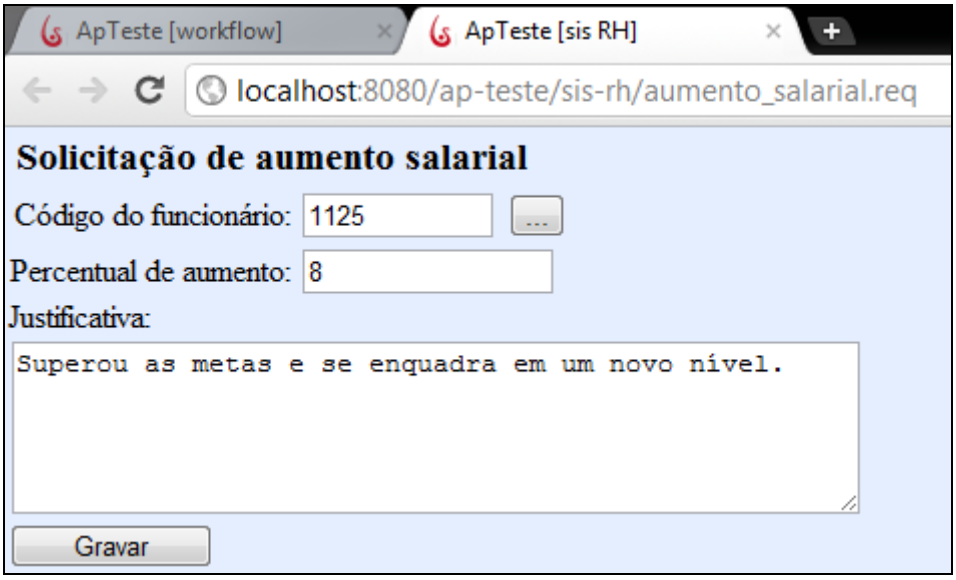


Figura 56 – Tela de entrada dos dados da solicitação de aumento

d) o líder clica no botão “Gravar”, fazendo com que a pendência seja enviada para o gerente;

e) o gerente acessa sua lista de tarefas pendentes no ApTeste, onde visualiza o registro correspondente à solicitação aberta pelo líder;

f) o gerente clica no botão “Tratar” da tarefa, fazendo com que seja aberta a tela na qual ele visualiza os dados da solicitação, marca a opção indicando que ele aprova e clica em “Gravar”;

g) o usuário Marge acessa sua lista de tarefas pendentes no ApTeste, onde visualiza o registro correspondente à solicitação aprovada pelo gerente Barney;

h) o usuário Marge clica no botão “Tratar” da tarefa, fazendo com que seja aberta a tela que contém os campos que devem ser preenchidos para efetivar o aumento no sistema. Os campos cujas informações foram inseridas no decorrer da execução do fluxo já aparecem preenchidos;

i) enquanto o usuário Marge toma as medidas legais para efetivação do aumento, o líder (usuário Lisa) acessa na sua lista de tarefas pendentes o registro correspondente à solicitação aprovada;

j) o líder informa ao funcionário Homer que o aumento foi aprovado;

k) o líder abre a tarefa que está pendente, assinala a opção indicando que o funcionário foi notificado e clica em “Gravar”;

l) o usuário Marge finaliza o tratamento de sua tarefa e clica em “Gravar”;

m) o fluxo é finalizado com sucesso, resultando no log exibido no Quadro 15, no qual foram registrados os eventos originados das ações dos usuários e das mensagens trocadas entre a extensão de *web services* e o ApTeste.

Problems | Console | Servers | Progress | Tasks | Search

C:\Java\glassfish3\glassfish\domains\domain1\logs\server.log

```
INFO: ApTeste.Workflow -> solicitando lista de fluxos disponíveis
INFO: Web service -> processando [nome: Flow; método: getExecutableList]
INFO: ApTeste.Workflow -> mostrando lista de fluxos disponíveis
INFO: ApTeste.Workflow -> solicitando início de execução do fluxo
INFO: Web service -> processando [nome: Flow; método: startFlow; fluxo: 4]
INFO: ApTeste.Workflow -> fluxo iniciado
INFO: Web service -> solicitando processamento [token: 4:1:1:1]
INFO: ApTeste.SisRH (ws Salarios) -> gerando pendência para o usuário Lisa
INFO: ApTeste.Workflow -> retornando lista de tarefas do usuário Lisa
INFO: ApTeste.SisRH -> abrindo tarefa do usuário Lisa
INFO: ApTeste.SisRH -> usuário Lisa tratou tarefa 'preencher_dados_aumento'
INFO: ApTeste.SisRH -> notificando tratamento da tarefa
INFO: Web service -> processando [nome: Feedback; método: continueFlow; token: 4:1:1:1]
INFO: Web service -> solicitando processamento [token: 4:1:1:1]
INFO: ApTeste.SisRH (ws Salarios) -> gerando pendência para o usuário Barney
INFO: ApTeste.Workflow -> retornando lista de tarefas do usuário Barney
INFO: ApTeste.SisRH -> abrindo tarefa do usuário Barney
INFO: ApTeste.SisRH -> usuário Barney tratou tarefa 'aprovar_aumento'
INFO: ApTeste.SisRH -> notificando tratamento da tarefa
INFO: Web service -> processando [nome: Feedback; método: continueFlow; token: 4:1:1:1]
INFO: Web service -> solicitando processamento [token: 4:1:1:2]
INFO: ApTeste.SisRH (ws Salarios) -> gerando pendência para o usuário Lisa
INFO: Web service -> solicitando processamento [token: 4:1:1:3]
INFO: ApTeste.SisRH (ws Salarios) -> gerando pendência para o usuário Marge
INFO: ApTeste.Workflow -> retornando lista de tarefas do usuário Marge
INFO: ApTeste.SisRH -> abrindo tarefa do usuário Marge
INFO: ApTeste.SisRH -> abrindo tarefa do usuário Lisa
INFO: ApTeste.SisRH -> abrindo tarefa do usuário Lisa
INFO: ApTeste.SisRH -> usuário Lisa tratou tarefa 'notificar_aumento'
INFO: ApTeste.SisRH -> notificando tratamento da tarefa
INFO: Web service -> processando [nome: Feedback; método: continueFlow; token: 4:1:1:2]
INFO: ApTeste.SisRH -> usuário Marge tratou tarefa 'formalizar_aumento'
INFO: ApTeste.SisRH -> notificando tratamento da tarefa
INFO: Web service -> processando [nome: Feedback; método: continueFlow; token: 4:1:1:3]
```

Quadro 15 – Log da simulação de execução do fluxo de aumento salarial

3.4.2.5 Manutenção do processo de aumento salarial

Para demonstrar os procedimentos realizados para dar manutenção em um fluxo de processo que está em uso na empresa, foi simulada uma situação onde os usuários do processo de aumento salarial identificam a necessidade de modificar o fluxo. A seguir são descritos os eventos ocorridos na simulação:

a) ao identificar a necessidade de mudança, a empresa aciona o modelador para que ele altere o fluxo;
b) enquanto o fluxo atual continua sendo utilizado pelos usuários, o modelador utiliza o editor de processos para fazer as alterações na versão de testes (a versão de produção não é afetada);
c) o modelador finaliza as alterações e notifica o administrador;
d) o administrador acessa o console de administração, seleciona a versão 2 do fluxo de aumento salarial e clica em "Disponibilizar", conforme ilustrado na Figura 57;

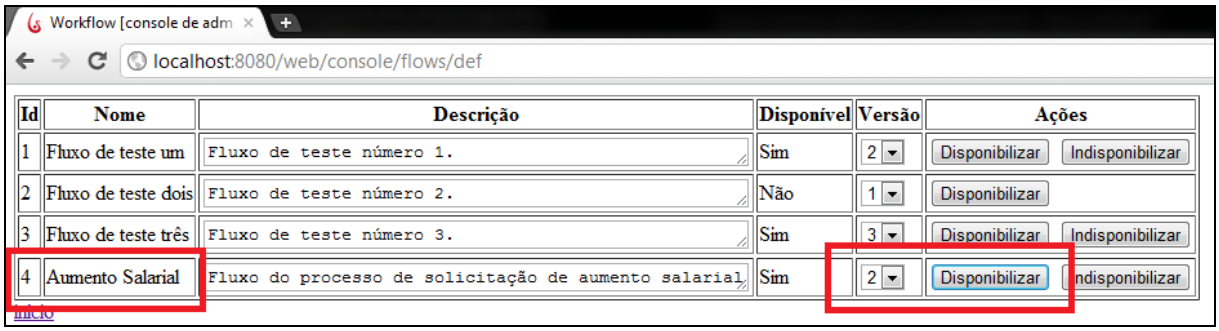


Figura 57 – Disponibilização de nova versão do fluxo de aumento salarial

e) os usuários utilizam a nova versão do fluxo (sempre que um processo é iniciado, o executor utiliza a versão atualmente disponível);
f) os usuários identificam que as alterações feitas no fluxo não estão corretas;
g) o administrador acessa o console de administração, seleciona a versão 1 do fluxo de aumento salarial e clica em "Disponibilizar";
h) os usuários voltam a utilizar a versão 1 do fluxo, a qual não continha erros;
i) o modelador faz as correções no fluxo;
j) o administrador acessa novamente o console de administração, seleciona a versão 3 do fluxo de aumento salarial e clica em "Disponibilizar";
k) os usuários passam a utilizar a nova versão do fluxo;
l) os usuários identificam que o fluxo está correto e continuam a utilizá-lo.

3.5 RESULTADOS E DISCUSSÃO

A forma como a ferramenta foi desenvolvida, com a implementação de um *framework* que encapsula toda a lógica de modelagem e execução dos fluxos de processos, teve um impacto positivo nos resultados alcançados, permitindo destacar os seguintes aspectos:

a) a modularidade exigida para permitir extensibilidade deixa o código muito mais organizado, legível e de fácil manutenção. Também aumenta as possibilidades de reutilização de cada funcionalidade, pois cada rotina é escrita para fazer algo específico;

b) com o recurso de extensão, as expectativas quanto às possibilidades de integração com outros sistemas e máquinas de *workflow* foram superadas. A integração através de código fonte (extensão) apresenta maior performance e estabilidade, pois elimina as camadas de conversão de protocolos, conforme ilustrado na Figura 58. Além disso, é possível interagir com sistemas que não suportam *web services*;

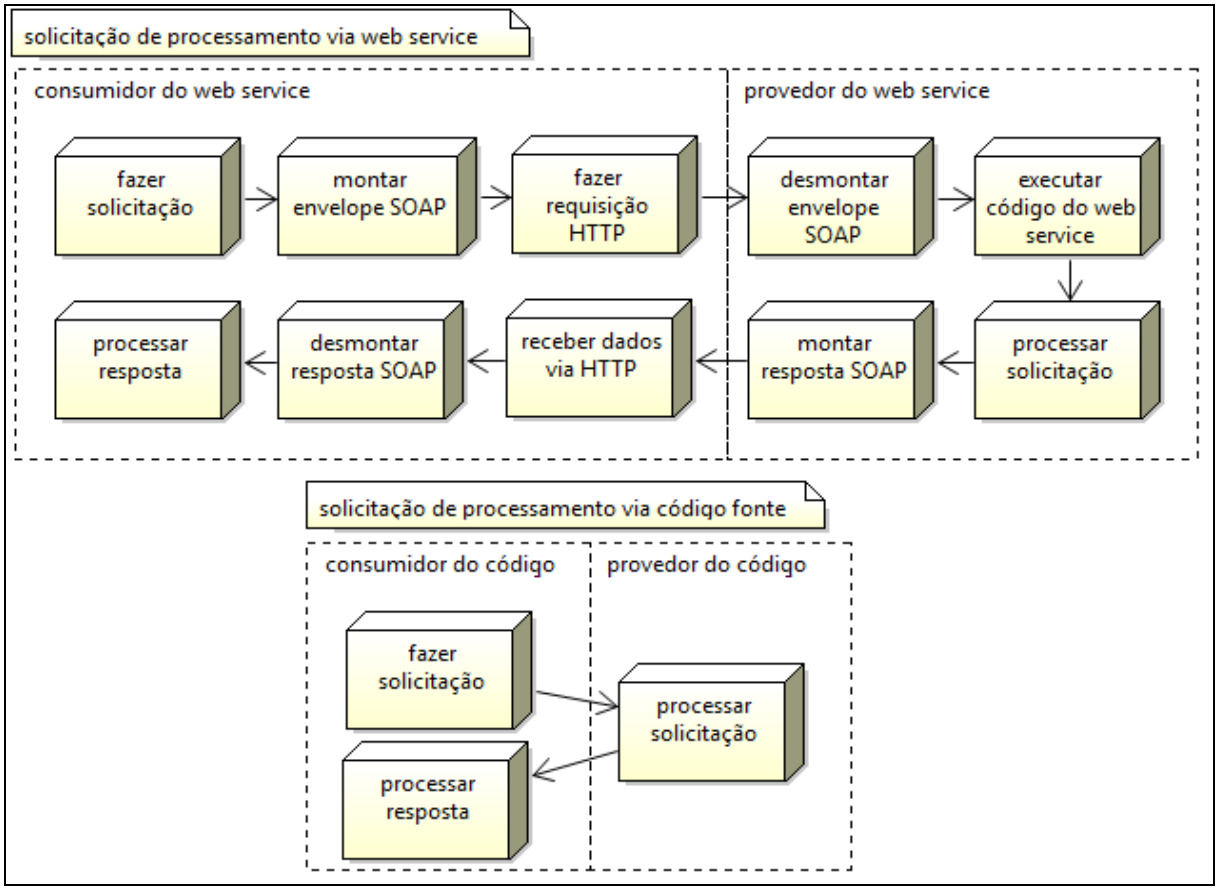


Figura 58 – Comparativo de execução de *web service* com código fonte

c) a possibilidade de estender as rotinas do editor permite criar telas especializadas que editam as *activities* de forma visual, sem utilizar *script*.

Em contrapartida, desenvolver um *framework* é uma decisão que deve ser tomada com cautela, pois aumenta exponencialmente o custo de desenvolvimento devido ao tempo exigido com tarefas como definir protocolos e interfaces, construir *factories*, realizar as interfaces e estudar a melhor forma de modularizar os códigos.

Outra questão que impactou no tempo de desenvolvimento foi a utilização de applet para desenvolver o editor. Após avaliar as tecnologias de renderização de componentes disponíveis para modelar a BPMN, optou-se pela utilização de *applet*. O fato de *applets* serem completamente implementados em Java traz, além dos benefícios da própria linguagem, a possibilidade de utilizar a API do Java 2D. O Java 2D facilitou a tarefa de renderização devido à quantidade de primitivas geométricas que ela disponibiliza, pecando apenas pelo fato de não possuir primitivas para linhas direcionadas com seta, exigindo muito esforço para representar os *sequence flows*. Um inconveniente do *applet* é o fato de seu código ser processado no navegador *web* da máquina cliente ao invés do servidor. Em sua fase inicial, a estrutura de classes do *framework* foi implementada considerando que todo o processamento do editor seria realizado na camada servidora, tendo acesso direto às informações do *framework*. Devido à inviabilidade de enviar todo o código do *framework* juntamente com o *applet* para o navegador *web*, foi necessário remodelar a estrutura de classes de definição de fluxo para que elas não referenciassem nenhuma classe que não estivesse no mesmo pacote que elas ou no *core*. Esta remodelagem permitiu com que as classes de definição e do *core* fossem empacotadas com as classes do *applet* que são enviadas para o navegador. Por fim, a utilização de *applet* neste caso teria economizado muito tempo se a estrutura de classes tivesse sido inicialmente projetada para utilizar esta tecnologia.

A utilização de padrões de projeto como *abstract method* e *abstract factory* facilitou consideravelmente a implementação da extensibilidade. Um bom resultado disto é a facilidade de implementar suporte a novos componentes da BPMN, tanto no editor quanto no executor. Criando derivações das classes abstratas `Element`, `ElementDefinition` e `ModelElement` novos elementos são disponibilizados sem alterar a codificação dos já existentes. Feito isso, basta alterar as classes `FlowThread` e `FlowCompiler` para que o executor saiba como processar os elementos em questão.

A vantagem desta ferramenta com relação ao BPEL *Designer* e ao BPEL *Service Engine* vem da possibilidade de automatizar o fluxo do processo diretamente sobre a BPMN. Além de dispensar a necessidade de conhecimentos específicos em BPEL, o fato de não ter uma linguagem intermediária diminui a quantidade de tempo necessário para automatizar o processo. Também garante que alterações feitas no modelo em BPMN serão corretamente

aplicadas ao processo automatizado, enquanto o BPEL está suscetível ao esquecimento ou outro erro humano no momento de passar as alterações do BPMN para o BPEL. No mais, tanto o BPEL quanto a ferramenta desenvolvida baseiam-se na execução de *web services* para comunicar-se com outros sistemas e máquinas de *workflow*. A vantagem desta ferramenta é que ela permite também a integração nativa, via código fonte, com máquinas de *workflow* desenvolvidas em Java.

Em comparação com o *Enterprise Architect*, pode-se observar que ele contempla os elementos da BPMN não implementados nesta ferramenta. Porém, por ser mais abrangente, ele também é menos produtivo. Ao adicionar um elemento ao modelo, é necessário abrir a tela de propriedades do elemento em questão, a qual possui vários campos que podem ser preenchidos com informações que, para esta ferramenta, são desnecessárias. Outra desvantagem do *Enterprise Architect* é que ele não executa o fluxo, apenas modela. Mesmo possuindo recurso de exportação para BPMN, ele ainda depende de um executor de BPEL para que o fluxo exportado seja efetivamente automatizado.

A ferramenta deste trabalho possui muitas semelhanças com o sistema desenvolvido por Reinert (2006), já que ambas baseiam-se nos conceitos de automatização de processos com máquinas de *workflow*. O módulo de configuração de seu sistema tem papel equivalente ao módulo editor deste trabalho, o que ocorre também com os módulos de execução de ambos. O Quadro 16 exibe um comparativo entre os dois trabalhos.

Sistema de Reinert	**Ferramenta deste trabalho**
Foco na automatização do processo de desenvolvimento de software	Abrange a automatização de qualquer processo de negócio
Por ter um foco específico, implementa as rotinas de execução do processo como um todo, sem depender de outros sistemas	Por ser focado na execução da BPMN e não no processamento das tarefas em si, depende de máquinas de *workflow* e/ou outros sistemas para iniciar a execução dos processos e para processar as tarefas
Não suporta paralelismo	Suporta paralelismo
Modelagem do processo baseada em fluxograma	Modelagem do processo baseada em BPMN, que é mais completa e específica para automatização de processos de negócio, além de ser um padrão popular no mercado
Implementado em linguagem Java com base na arquitetura Java EE	Implementado em linguagem Java com base na arquitetura Java EE
Framework Mentawai como servidor *web*	GlassFish como servidor *web*
Módulo de configuração como aplicativo que executa na máquina do usuário e módulo de execução com acesso *web*	Editor e executor com acesso *web*
Utiliza JSP para as páginas *web*	Utiliza HTML e JavaScript para as páginas *web*
Utiliza o *framework* JGraph para a modelagem dos processos	Utiliza *applet* e Java 2D para a modelagem dos processos
Utiliza *Hibernate* e banco de dados Apache Derby para a persistência de dados	Utiliza SQL e banco de dados SQL *Server* para a persistência de dados. A camada de acesso aos dados é baseada no padrão de projetos DAO, possibilitando a portabilidade para outras formas de persistência.
Edição de processo baseada em fluxograma	Edição de processo baseada em BPMN

Quadro 16 – Comparação com o sistema de Reinert

4 CONCLUSÕES

A otimização e automatização dos processos de negócio são necessidades que vêm crescendo nas empresas à medida que a informática evolui e o mercado se torna mais competitivo. A modelagem destes processos auxilia a defini-los, a encontrar possíveis falhas e, consequentemente, a otimizá-los. A utilização de uma notação de fácil compreensão como a BPMN é importante para facilitar o entendimento do processo por parte dos envolvidos. Porém, pô-lo em prática ainda é muito suscetível a erros humanos. A utilização de máquinas de *workflow* minimiza estes erros e auxilia a otimizar ainda mais os processos através da automatização.

O desenvolvimento de uma ferramenta que possibilita automatizar os processos de negócio diretamente sobre sua modelagem mostrou-se funcional e atingiu todos os objetivos previamente formulados. O aumento na produtividade foi alcançado ao eliminar a necessidade da utilização de uma linguagem intermediária para realizar a automatização. Este aumento fica ainda mais evidente quando é necessário fazer uma modificação no processo, onde basta editar o seu modelo e disponibilizá-lo para utilização. A aplicação dos conceitos de SOA no desenvolvimento da extensão de comunicação via *web services* diminui o custo da automatização dos processos, pois possibilita total reaproveitamento de informações e regras de negócio existentes em sistemas que as disponibilizam na forma de *web services*, fazendo com que elas não tenham que ser duplicadas ou reimplementadas dentro da ferramenta ou do modelo do processo. Por se basear em tecnologias básicas e bem difundidas como HTTP e XML, a utilização do protocolo SOAP possibilita que até mesmo sistemas que não foram projetados para esta finalidade possam disponibilizar serviços e também consumir os serviços disponibilizados pelo executor de processos, independente de plataforma ou linguagem de programação. Ao disponibilizar suas funcionalidades como serviços, os sistemas tendem a abstrair a lógica de funcionamento interna, possibilitando que cada serviço seja reutilizado individualmente em diferentes contextos.

Uma grande vantagem da automatização dos processos de negócio é a facilidade de identificar pontos críticos do processo. Através de recursos de monitoramento e geração de estatísticas pode-se, por exemplo, identificar e eliminar pontos de gargalo do processo, o que auxilia a tomada de ações de melhoria. Já pensando no desenvolvimento de recursos de monitoramento, o editor desta ferramenta foi desenvolvido para facilitar o reaproveitamento das rotinas de renderização dos elementos da BPMN na implementação dos recursos de

monitoramento e administração, independente de estes recursos utilizarem uma arquitetura *web* ou cliente-servidor.

Como a BPMN é uma notação abrangente, consequentemente ela é muito extensa. Devido a isto, este trabalho e a ferramenta desenvolvida não contemplam todos os elementos da notação.

4.1 EXTENSÕES

As seguintes extensões são sugeridas para este trabalho:

a) disponibilizar recursos de *Business Activity Monitoring* (BAM) para prover acesso em tempo real aos indicadores da execução do fluxo, permitindo medir a performance e identificar pontos críticos no processo de negócio;
b) implementar os outros componentes de modelagem de processos da BPMN que são passíveis de automatismo, dentre os quais pode-se destacar as *activities* dos tipos *sub-process* e *call*, os *events* baseados em *messages* e *timers*, incluindo os *intermediate events*;
c) desenvolver extensões para integração especializada com sistemas comerciais específicos, demonstrando a capacidade de reutilização do *framework*. Caso os sistemas comerciais escolhidos disponibilizem suas funcionalidades como *web services*, pode-se optar por melhorar a extensão de *web service* desenvolvida neste trabalho para tornar sua interface mais amigável e produtiva, incluindo rotinas de importação de WSDL;
d) implementar uma máquina de *workflow* para interagir com o executor, disponibilizando telas para administração e utilização dos fluxos de processos de forma semelhante ao que foi feito no aplicativo utilizado para o estudo de caso;
e) desenvolver mecanismos para importar modelos BPMN e implementações em BPEL de outras ferramentas disponíveis no mercado.

REFERÊNCIAS BIBLIOGRÁFICAS

BITENCOURT, Maurício. **Modelagem de processos com BPMN**. [S.l.], [2007?]. Disponível em: <http://www.projeler.com.br/download/pdf/artigo_bpmn_projeler_mauricio_bitencourt.pdf>. Acesso em: 12 set. 2010.

FAYAD, Mohamed; SCHMIDT, Douglas C. Object-oriented application frameworks. **Communications of the ACM**, New York, v. 40, n. 10, p. 32-38, Out. 1997.

HAN, Dong S.; KIM, Kwang H. **Performance and scalability analysis on client-server**: workflow architecture. [S.l.], [2001?]. Disponível em: <http://ctrl.kyonggi.ac.kr/homepage/paper/upfile/C200106001.pdf>. Acesso em: 20 set. 2010.

MONTEIRO, Sérgio. **Windows workflow foundation:** parte 01. [S.l.], ago. 2007. Disponível em: <http://imasters.uol.com.br/artigo/6794/dotnet/windows_workflow_foundation_parte_01/>. Acesso em: 05 set. 2010.

OBJECT MANAGEMENT GROUP. **Business Process Model and Notation (BPMN):** version 2.0. [S.l.], jun. 2010. Disponível em: <http://www.omg.org/cgi-bin/doc?dtc/10-06-04.pdf>. Acesso em: 12 set. 2010.

OBJECT MANAGEMENT GROUP. **BPMN information home**. [S.l.], [2011]. Disponível em: <http://www.bpmn.org>. Acesso em: 07 jul. 2011.

ORACLE. **Lesson: overview to the Java 2D API concepts:** the Java tutorials. [S.l.], [2011]. Disponível em <http://download.oracle.com/javase/tutorial/2d/overview/index.html>. Acesso em: 05 jun. 2011.

REINERT, Roberto. **Sistema de workflow para modelagem e execução de processos de software.** 2006. 75 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação). Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

ROSE INDIA TECHNOLOGIES PVT. LTD. **Definition of applet, what is Java applets, introduction to Java applet**. [S.l.], abr. 2007. Disponível em: <http://www.roseindia.net/java/example/java/applet/applet.shtml>. Acesso em: 05 jun. 2011.

SAMPAIO, Cleuton. **SOA e web services em Java**.Rio de Janeiro: Brasport, 2006.

SPARX SYSTEMS. **Enterprise Architect – UML design tools and UML CASE tools for software development.** [S.l.], 2010. Disponível em: <http://www.sparxsystems.com.au/products/ea/index.html>. Acesso em: 16 set. 2010.

SPARX SYSTEMS. **Enterprise Architect user guide.** [S.l.], [2004?]. Documento eletrônico disponibilizado com a ferramenta Enterprise Architect 8.0.

SUN MICROSYSTEMS. **BPEL designer and service engine user's guide.** Santa Clara, 2009. Disponível em: <http://dlc.sun.com/pdf/821-0539/821-0539.pdf>. Acesso em: 16 set. 2010.

WHITE, Stephen A. **Introduction to BPMN**. [New York], 2004. Disponível em: <http://www.bpmn.org/Documents/Introduction_to_BPMN.pdf>. Acesso em: 06 set. 2010.

WORLD WIDE WEB CONSORTIUM. **SOAP specifications**. [S.l.], 2007. Disponível em: <http://www.w3.org/TR/soap/>. Acesso em: 07 set. 2010.

WULONG, Tang. **What is workflow?:** definition from whatis.com. [S.l.], out. 1998. Disponível em: <http://searchcio.techtarget.com/definition/workflow>. Acesso em: 04 jun. 2011.

APÊNDICE A – Detalhamento dos atores e casos de uso especificados

Neste apêndice é apresentado o detalhamento dos atores e casos de uso definidos para especificar este trabalho, conforme diagramas ilustrados na seção 3.3.1. O Quadro 17 contém todos os atores com sua respectiva descrição. Do Quadro 18 até o Quadro 29 são detalhados os casos de uso.

ATOR	**DESCRIÇÃO**
Sistema externo	O sistema externo representa qualquer sistema ou ferramenta que interage com o executor de processos. É responsável por: 1. iniciar a execução dos processos; 2. processar as tarefas a ele atribuídas quando solicitado pelo executor; 3. possibilitar interação com o usuário. **Nota:** Em um cenário convencional, esse ator representa a máquina de *workflow* e os sistemas que disponibilizam suas regras de negócio como serviços.
Usuário	O usuário é a pessoa que utiliza os processos disponíveis. Ele utiliza o executa operações simples como iniciar novos processos, tratar tarefas a ele atribuídas e acompanhar o andamento dos processos dos quais ele participou. **Nota:** Esse ator interage com o executor de processos através dos recursos providos pelos sistemas externos, nunca diretamente.
Administrador	O administrador é um usuário com permissões elevadas que pode executar operações de administração dos processos existentes como, por exemplo, tornar uma versão disponível ou indisponível para os demais usuários, colocar um fluxo de testes em produção ou cancelar a execução de um processo.
Modelador	O modelador é a pessoa responsável pela modelagem dos processos de negócio. Utiliza o editor para criar e/ou alterar os fluxos dos processos que serão disponibilizados para os usuários.
Desenvolvedor	O desenvolvedor é a pessoa que cria e mantém as extensões que agregam novas funcionalidades ao editor e ao executor de processos, possibilitando a interação com os sistemas externos.

Quadro 17 – Descrição dos atores especificados

Caso de uso – UC01 - Modelar fluxo do processo
Ator: Modelador **Objetivo:** Modelar o fluxo de do processo de negócio **Pré-condições:** Levantamento detalhado do processo a ser modelado, especificação dos serviços providos pelos sistemas externos que serão utilizados **Pós-condições:** BPD do processo pronto para ser testado **Cenário Principal:** **1.** Modelador cria ou altera o fluxo do processo **2.** Modelador insere as variáveis que serão utilizadas na execução do processo **3.** Modelador insere os elementos do processo na área de edição **4.** Modelador define as expressões de validação dos *gateways* de tipo *exclusive* e *inclusive* **5.** Modelador define os scripts de cada *activity* que representa uma execução de serviço de sistema externo **6.** Modelador clica no botão de salvar **7.** Ferramenta salva o fluxo

Quadro 18 – Descrição do `UC01 - Modelar fluxo do processo`

Caso de uso – UC02 - Criar fluxo do processo
Ator: Modelador **Objetivo:** Criar um novo fluxo de processo **Pré-condições:** Levantamento detalhado do processo a ser modelado, especificação dos serviços providos pelos sistemas externos que serão utilizados **Pós-condições:** BPD do processo pronto para ser testado **Cenário Principal:** **1.** Modelador insere um nome e uma descrição no editor **2.** Modelador clica em "Adicionar" **3.** Ferramenta salva as informações do novo processo com o fluxo em branco **4.** Ferramenta exibe a área de edição para modelagem do processo **5.** Continua a execução do UC01 **Cenário Alternativo:** No passo 1, caso o nome informado esteja em branco: **1.** Ferramenta exibe mensagem de erro para o modelador **2.** Modelador informa um nome para o fluxo **3.** Volta para o cenário principal **Cenário Alternativo:** No passo 1, caso o nome informado já exista: **1.** Ferramenta exibe mensagem de erro para o modelador **2.** Modelador informa um novo nome para o fluxo **3.** Volta para o cenário principal

Quadro 19 – Descrição do `UC02 - Criar fluxo do processo`

Caso de uso – UC03 - Alterar fluxo do processo
Ator: Modelador **Objetivo:** Alterar o fluxo de um processo previamente criado **Pré-condições:** Fluxo do processo criado **Pós-condições:** BPD do processo pronto para ser testado **Cenário Principal:** **1.** Modelador solicita lista de fluxos existentes **2.** Ferramenta exibe tabela com todos os fluxos modelados **3.** Modelador clica na opção "Editar" correspondente ao fluxo a ser alterado **4.** Ferramenta exibe o modelo do fluxo solicitado **5.** Continua a execução do UC01 **Cenário Alternativo:** No passo 2,caso não haja nenhum fluxo modelado: **1.** Ferramenta exibe uma tabela vazia **2.** O caso de uso é encerrado

Quadro 20 – Descrição do `UC03 - Alterar fluxo do processo`

Caso de uso – UC04 - Testar fluxo de processo
Ator: Modelador **Objetivo:** Testar um fluxo de processo para garantir seu funcionamento **Pré-condições:** Fluxo modelado conforme UC01 **Pós-condições:** Fluxo executado com sucesso ou com erro **Cenário Principal:** **1.** Modelador solicita lista de fluxos existentes **2.** Ferramenta exibe tabela com todos os fluxos modelados **3.** Modelador clica na opção "Testar" correspondente ao fluxo a ser testado **4.** Ferramenta dispara o início da execução do processo **Cenário Alternativo:** No passo 2,caso não haja nenhum fluxo modelado: **1.** Ferramenta exibe uma tabela vazia **2.** O caso de uso é encerrado **Cenário Alternativo:** No passo 3, caso o fluxo modelado esteja inconsistente: **1.** Ferramenta exibe mensagem de erro ao modelador **2.** O caso de uso é encerrado

Quadro 21 – Descrição do `UC04 - Testar fluxo de processo`

Caso de uso – UC05 - Excluir fluxo de processo
Ator: Modelador **Objetivo:** Excluir um fluxo de processo que não será utilizado **Pré-condições:** Fluxo modelado conforme UC01 **Pós-condições:** Dados do fluxo excluídos **Cenário Principal:** **1.** Modelador solicita lista de fluxos existentes **2.** Ferramenta exibe a tabela com todos os fluxos modelados **3.** Modelador clica na opção "Excluir" correspondente ao fluxo a ser excluído **4.** Ferramenta exclui o fluxo **Cenário Alternativo:** No passo 2,caso não haja nenhum fluxo modelado: **1.** Ferramenta exibe uma tabela vazia **2.** O caso de uso é encerrado **Cenário Alternativo:** No passo 3, caso o fluxo já possua alguma versão disponível para os usuários: **1.** Ferramenta exibe mensagem de erro ao modelador **2.** O caso de uso é encerrado

Quadro 22 – Descrição do `UC05 - Excluir fluxo de processo`

Caso de uso – UC06 - Disponibilizar fluxo de processo
Ator: Administrador **Objetivo:** Disponibilizar um fluxo de processo para que os usuários possam utilizá-lo **Pré-condições:** Fluxo modelado conforme UC01 **Pós-condições:** Fluxo disponível para utilização (pode ser executado) **Cenário Principal:** **1.** Administrador acessa a tela de fluxos existentes **2.** Ferramenta exibe a tabela com todos os fluxos modelados **3.** Administrador seleciona o fluxo **4.** Administrador seleciona a versão que deseja disponibilizar **5.** Administrador clica na opção "Disponibilizar" **6.** Ferramenta torna o fluxo disponível para uso **Cenário Alternativo:** No passo 2,caso não haja nenhum fluxo modelado: **1.** Ferramenta exibe tabela vazia **2.** O caso de uso é encerrado

Quadro 23 – Descrição do `UC06 - Disponibilizar fluxo de processo`

Caso de uso – UC07 - Indisponibilizar fluxo de processo
Ator: Administrador **Objetivo:** Indisponibilizar um fluxo de processo para que ele não possa ser utilizado pelos usuários **Pré-condições:** Fluxo disponibilizado conforme UC06 **Pós-condições:** Fluxo indisponível para uso (não pode ser executado) **Cenário Principal:** **1.** Administrador acessa a tela de fluxos existentes **2.** Ferramenta exibe a tabela com todos os fluxos modelados **3.** Administrador seleciona o fluxo **4.** Administrador clica na opção "Indisponibilizar" **5.** Ferramenta torna o fluxo indisponível para uso **Cenário Alternativo:** No passo 2,caso não haja nenhum fluxo modelado: **1.** Ferramenta exibe tabela vazia **2.** O caso de uso é encerrado **Cenário Alternativo:** No passo 4,caso o fluxo já esteja indisponível: **1.** Ferramenta desabilita a opção "Indisponibilizar" **2.** Administrador seleciona outro fluxo **3.** Volta para o cenário principal

Quadro 24 – Descrição do UC07 - Indisponibilizar fluxo de processo

Caso de uso – UC08 - Visualizar processos em execução
Ator: Administrador **Objetivo:** Visualizar todos os processos que estão sendo executados no momento **Pré-condições:** Processos iniciados conforme UC10 **Pós-condições:** Exibição dos processos em execução **Cenário Principal:** **1.** Administrador solicita informações dos processos em execução **2.** Ferramenta exibe tabela com todos os processos que estão com execução em andamento

Quadro 25 – Descrição do UC08 - Visualizar processos em execução

Caso de uso – UC09 - Cancelar execução de processo
Ator: Administrador **Objetivo:** Cancelar a execução de um processo para que ele não seja continuado **Pré-condições:** Processo iniciado conforme UC10 **Pós-condições:** Execução do processo cancelada **Cenário Principal:** **1.** Administrador visualiza os processos em execução (vide UC08) **2.** Administrador clica na opção "Cancelar execução" **3.** Ferramenta cancela a execução do processo

Quadro 26 – Descrição do UC09 - Cancelar execução de processo

Caso de uso – UC10 - Iniciar execução de processo
Ator: Usuário, Sistema externo **Objetivo:** Iniciar a execução de um processo **Pré-condições:** Processo disponibilizado conforme UC06 **Pós-condições:** Processo iniciado **Cenário Principal:** **1.** Usuário acessa o sistema externo **2.** Usuário solicita lista de processos disponíveis **3.** Sistema externo solicita a lista de processos disponíveis **4.** Ferramenta retorna lista de processos disponíveis **5.** Sistema externo exibe lista de processos disponíveis **6.** Usuário seleciona o processo **7.** Usuário solicita que o processo seja iniciado **8.** Sistema externo solicita que o processo seja iniciado **9.** Ferramenta inicia execução do processo **Nota:** A forma como o usuário interage com o sistema externo é definida na especificação do sistema externo

Quadro 27 – Descrição do `UC10 - Iniciar execução de processo`

Caso de uso – UC11 - Processar tarefa

Ator: Usuário, Sistema externo
Objetivo: Processar uma tarefa decorrente da execução de uma *activity* do fluxo de um processo em execução
Pré-condições: Processo iniciado conforme UC10
Pós-condições: Continuidade da execução do fluxo do processo

Cenário Principal:
1. Ferramenta inicia processamento de uma *activity* definida durante a modelagem (vide UC01) como sendo a execução de uma tarefa de sistema externo
2. Ferramenta solicita que o sistema externo execute a tarefa
3. Sistema externo gera uma pendência para o usuário
4. Usuário acessa o sistema externo
5. Usuário solicita lista de tarefas pendentes
6. Sistema externo exibe lista de tarefas pendentes
7. Usuário seleciona a tarefa a ser processada
8. Sistema externo exibe tela com campos que precisam ser informados pelo usuário
9. Usuário informa os dados necessários
10. Usuário clica na opção de concluir a tarefa
11. Sistema externo processa as informações inseridas pelo usuário
12. Sistema externo notifica a ferramenta que a tarefa foi processada
13. Ferramenta conclui execução da tarefa
14. Ferramenta continua fluxo de execução do processo

Cenário Alternativo:
No passo 3, caso a tarefa não dependa de interação com o usuário:
1. Sistema externo processa a tarefa
2. Volta para o cenário principal, a partir do passo 13.

Nota: A forma como o usuário interage com o sistema externo é definida na especificação do sistema externo

Quadro 28 – Descrição do `UC11 - Processar tarefa`

Caso de uso – UC12 - Desenvolver extensão
Ator: Desenvolvedor **Objetivo:** Desenvolver extensão para agregar novas funcionalidades ao editor e executor de processos **Pré-condições:** Ambiente de desenvolvimento montado com as ferramentas descritas na seção 3.4.1. **Pós-condições:** Extensão pronta para ser instalada em ambiente de execução **Cenário Principal:** **1.** Desenvolvedor entra no ambiente de desenvolvimento **2.** Desenvolvedor cria um novo projeto Java **3.** Desenvolvedor cria uma classe que implementa a interface `ActivityExtension` **4.** Desenvolvedor cria uma classe que estende a classe `EditorExtension` **5.** Desenvolvedor cria uma classe que estende a classe `ExtensionMenuItemHandler` **6.** Desenvolvedor cria uma classe que estende a classe `ComponentEditorListener` **7.** Desenvolvedor repete os passos 5 e 6 para cada item a ser exibido no menu de contexto do elemento *Activity* durante o uso do editor de processos **8.** Desenvolvedor cria uma classe que implementa a interface `ExtensionFactory` **9.** Desenvolvedor exporta o projeto Java criado no passo 2 para a pasta de bibliotecas do servidor Java EE **10.** Desenvolvedor exporta as classes criadas nos passos 4, 5 e 6 para a pasta "*web*" do projeto Java do editor **11.** Desenvolvedor republica o projeto do editor no servidor Java EE **12.** Desenvolvedor acessa a página de cadastro de extensões do console de administração **13.** Desenvolvedor informa o nome completo da classe criada no passo 8 **14.** Desenvolvedor clica na opção "Adicionar" **15.** Ferramenta salva a extensão **16.** Desenvolvedor testa o uso da extensão **17.** Desenvolvedor distribui os arquivos gerados nos passos 9 e 10 **Cenário Alternativo:** No passo 14, caso não seja possível carregar as classes da extensão pelo nome informado: **1.** Ferramenta exibe mensagem de erro ao desenvolvedor **2.** Desenvolvedor informa o nome correto **3.** Volta ao cenário principal

Quadro 29 - Descrição do `UC12 - Desenvolver extensão`