O uso de programação reflexiva para o desenvolvimento de aplicações comerciais adaptativas executando em nuvem.

R S BERNINI e A R CAMOLESI

rsbernini@gmail.com; camolesi@femanet.com.br

RESUMO: Constitui em utilizar programação reflexiva e conceitos de Tecnologia Adaptativa para resolução de problemas relacionados nas aplicações comerciais que executam em nuvem.

PALAVRA-CHAVE: Tecnologia Adaptativa, Reflexão, Nuvem

ABSTRACT: It is to use reflective programming concepts and Adaptive Technology to solve problems in business applications running in the cloud.

KEYWORDS: Adaptive technology, Reflection, Cloud

1. Introdução

Sistemas devem ser construídos para atender as demandas das empresas. Neste contexto o desenvolvimento de softwares a cada dia mais tem se deparado com sistemas complexo e que tem que ser adaptados às exigências mais fundamentais segundo as características de cada empresa.

Aplicações complexas são caracterizadas por componentes e aspectos cuja estrutura e comportamento, comumente, podem modificar-se (Camolesi, 2004a). Tais aplicações possuem um comportamento inicial definido por um conjunto de ações que desempenham suas funções elementares, e durante a execução podem ter o seu comportamento modificado para dar suporte a novas

funcionalidades. Tais modificações são decorrentes dos estímulos de entrada a que são submetidos no sistema e/ou da ocorrência de suas ações internas.

Uma técnica utilizada para auxiliar os projetistas na modelagem de aplicações com comportamento modificável é a tecnologia adaptativa (NETO, 1993). A tecnologia adaptativa envolve um dispositivo não-adaptativo (subjacente) já existente em uma camada adaptativa que permite realizar mudanças no comportamento da aplicação definida (Pistori, 2003).

Atualmente com os avanços da internet, a evolução das redes de computadores, e facilidade de aquisição de dispositivos móveis e o surgimento da computação em nuvem depara-se com a necessidade de aplicações que sejam ubíquas e possam ser executadas em diversos ambientes e locais. Neste contexto, é sugerido neste trabalho que estudos sejam realizados com o foco no desenvolvimento de aplicações que permitam a mudança de comportamento em tempo de execução utilizando-se dos conceitos de Programação Reflexiva e possam ser executados em ambiente WEB, na nuvem.

2. Tecnologia Adaptativa

Com a evolução necessita entender que a inteligência humana tem parâmetros relacionado a problemas cada vez mais complexo impossibilitando que apenas uma única área seja responsáveis por tais atos existentes.

Yourdon (1990) relatou que o sistema assemelha a um conjunto estruturado ou de certa forma ordenado em devidas partes de um elemento que se completa e que se mantem integrado a uma ação reciproca em busca de conseguimento de uma demanda muito grande de objetivos, entretanto, tornando de modo mais simples definir um módulo sistemático com um conjunto de elementos interdependentes, ou partes que formam interagindo entre si um unitário complexo e completo, definindo previamente que existe muitas partes menores completando e complexando uma parte maior.

Sendo assim os sistemas devem de modo geral ser construídos para atender a empresas de um modo ao seu todo, com um desenvolvimento de software cada dia mais complexos e melhor adaptado tecnologicamente as grande demanda de exigências fundadas em cada necessidade específica.

Camolesi (2004) cita que as aplicações complexas são caracterizadas necessariamente de componentes e seus aspectos cuja a estrutura comportamental relata-se a adaptar e modificar a sua necessidade especial. Essas aplicações possuem um temperamento inicial definido por um conjunto de ações modificando para suportar uma infinidade de novas possibilidades que possa atender, sendo assim tais modificações decorre a estímulos de entrada de dados ou informações repassados pelo sistema decorrente de uma função internas acionárias. Utilizando se de técnicas projetadas na modelagem comportamental da aplicação, sendo modificável.

Pistori, (2003) cita que a tecnologia adaptativa envolve um dispositivo não-adaptativo (subjacente) já existente em uma camada adaptativa que permite realizar mudanças no comportamento da aplicação definida. Estudos ao qual já foram realizados com o objetivo de se criar uma ferramenta de aplicações adaptativas, CASACHI (2011) realizou um estudo que teve por objetivo apresentar o uso da Programação Orientada a Aspectos (KICZALES, 1997) no desenvolvimento de aplicações que utilizam os conceitos de Tecnologia Adaptativa. Tal estudo permitiu a implantação da Tecnologia Adaptativa de uma forma fácil e segura, além de permitir que novas funcionalidades (aspectos) sejam adicionadas ao software sem a necessidade de modificar o código fonte já produzido, ou seja, a aplicação não sofre modificações em sua total estrutura, e sim utiliza partes de uma sistema ao qual trata de auxiliar de modo dinâmica e adaptativa certo processo complexo da aplicação. O programador deve apenas acrescentar novos aspectos ao software e o mesmo se adapta ao código já existente. Porém neste trabalho verificou-se que os conceitos de Tecnologia Adaptativa não são implementados completamente, uma vez que os programas que utilizam Programação Orientada a Aspectos são combinados antes da compilação e os programas produzidos não acabam se tornando adaptativos em tempo de execução.

3. Tecnologia Adaptativa auto adaptação

O trabalho proposto neste projeto tem como base principal a Tecnologia Adaptativa, sendo esta, uma tecnologia com a característica de auto adaptação. Neste sentido espera que o desenvolvimento desta pesquisa possa contribuir com publicações para esta área direcionado para o ramo comercial corporativo

a fim de sanar necessidades complexas. O principal diferencial na Tecnologia Adaptativa é a forma razoavelmente simples que podemos transformar teorias já existentes, bem como fazer um reaproveitamento e estruturação destas para melhorar sua capacidade de respostas e interação. A Tecnologia Adaptativas faz com que seja facilitada a interação com o usuário, sem a necessidade de uma base de dados contemplando todas as possíveis reações de um jogador, porque cada regra inicial pode ser modificada de acordo com o ambiente ao qual está sendo trabalhado de forma que não necessita de alguém programando isso a cada nova ocorrência encontrada. Este conceito de auto-modificação da Tecnologia Adaptativa, torna a Inteligência Artificial, possa ser trabalhada de forma mais simples, e eficiente desde que seja feito de forma correta o seu desenvolvimento, seguindo passos que por mais simples que pareçam, mostrem uma complexidade no que diz a atenção dispensada para não ter um sistema com falhas futuras.

GONÇALVES E CAMOLESI (2012) apresentam o desenvolvimento de uma ferramenta para geração automática de aplicações comerciais a partir do modelo de Banco de Dados projetado para a aplicação. O desenvolvedor utilizando-se de um assistente gera automaticamente, de forma rápida e simples, uma aplicação para realizar manutenção dos dados desejados. A aplicação gerada permite ao usuário da mesma realizar operações de manutenção (inclusão, alteração, recuperação e remoção) dos dados. Além das operações básicas de manutenção a ferramenta permite que relatórios básicos também sejam gerados de forma automática.

4. Mudança de comportamento

Com a evolução crescente da internet, e das redes de computadores, com facilidade de aquisição de dispositivos móveis juntamente com o surgimento da computação em nuvem depara-se com a necessidade de aplicações que estejam todo o tempo conectado em toda a parte e possam ser executadas em diversos ambientes e locais. Os programas geralmente são escritos para trabalhar com dados. Eles em geral, leem, escrevem, manipulam e exibem dados (a geração de Gráficos a partir de bases de dados pré-existentes são um exemplo típico de programas desse tipo). Os tipos que você, como o programador, criar e usar é projetado para estes fins, e é você, em tempo de

projeto, que deve compreender as características dos tipos que você usa. Para alguns tipos de programas, no entanto, os dados que eles manipulam não são números, textos ou gráficos, mas são as informações sobre programas e tipos de programa. Tais informações são conhecidas como metadados (Uma informação sobre a informação). Os Atributos são um mecanismo para a adição de tais informações, tais como instruções do compilador e outros dados sobre seus dados, métodos e classes, para o próprio programa. Um programa pode olhar para os metadados de outros conjuntos ou de si mesmo, enquanto ele está executando.

Nos atuais sistemas de informação existentes no mercado apresentam um comportamento pragmaticamente estático com estruturas pré-definidas, nos quais são informados apenas os seus parâmetros inicias e estes são utilizados durante o tempo de execução, mantendo um padrão evolutivo e estativo atualmente. No mundo empresarial o dinamismo é fundamental, novas funcionalidades e estratégias de cálculos surgem no mundo dos negócios o que acarretam em mudanças que devem ser realizadas nos sistemas existentes, e priorizar custo, funcionalidade, tempo e melhoras significativamente o crescimento aplicando sempre novas tecnologias de crescimento fundada e aspectos adaptativos. A análise relacionada ao comportamento de tais sistemas ser estático torna impossível, na maioria dos casos, mudanças automáticas ou sem a necessidade de recodificação de trechos e partes do sistema já desenvolvido. Tais mudanças além de demandar tempo e a alocação de profissionais para a sua realização, pode gerar problemas imprevistos nas aplicações e, até mesmo, terem altos custos para serem realizadas. O uso da Tecnologia Adaptativa neste tipo de aplicação tem por objetivo a atender uma necessidade da aplicação utilizando uma tabela de decisão que possa ter seu comportamento alterado com a obtenção de novas informações comparando-as e apresentando respostas em seu tempo de execução.

5. Desenvolvimento da aplicação

5.1 Tecnologias

A tecnologia que para se criar a aplicação é muito importante, pois é com ela que tais conceitos apresentados será provado e permitirá o usuário ter

acesso de modo mais simplificado e funcional. A opção foi de se utilizar apenas tecnologia Open Sources, que são as tecnologia que pode se tirar proveito livremente sem nenhum custo para o programador.

Sendo assim a utilização da linguagem Java para desenvolvimento Web, uma das ferramentas gratuitas mais utilizadas no mercado atual, o motivo obvio além da liberdade e do poder de desenvolvimento é que possui uma gama muito grande de materiais disponíveis nos principais meio, como livros, fóruns, blogs, vídeos, e em diversos formatos, sendo de uso multiplataforma, podendo ser desenvolvido em qualquer sistema operacional, Windows, Linux, Mac ou Unix e utilizado como aplicação nas mesmas.

5.2 Computação em nuvem

Computação em nuvem é uma tendência recente de tecnologia cujo objetivo é proporcionar serviços de Tecnologia da Informação (TI) sob demanda com pagamento baseado no uso. Tendências anteriores à computação em nuvem foram limitadas a uma determinada classe de usuários ou focadas em tornar disponível uma demanda específica de recursos de TI, principalmente de informática. O que torna hoje a computação em nuvem uma tecnologia muito interessante é o avanço tecnológico na velocidade das conexões disponíveis e o ilimitado armazenamento de dados.

MILLER (2008) destaca que, por se tratar de um novo paradigma, existem muitas contradições. Entretanto, a maioria dos pesquisadores considera que essa nova abordagem deva proporcionar economia de escala, uma vez que possibilitará que usuários domésticos, a partir de um computador com capacidades reduzidas ou até mesmo um televisor de alta definição, possa utilizar serviços especializados, oferecidos por companhias.

Basicamente há três partes que compõe os elementos de uma computação em nuvem: Clientes, Data Center e Servidores Distribuídos. Cada elemento desenvolve um papel especifico na solução.

Data Center é um conjunto de servidores onde são armazenados os aplicativos que são acessados pela internet. Uma técnica que vem crescendo na área de TI é a virtualização de servidores. O que permite que vários servidores

sejam instalados em apenas uma máquina física, vale lembrar que o número de servidores instalados em uma única máquina dependerá do tamanho e da velocidade do servidor e quais aplicações estão funcionando no servidor virtual. Nesta técnica o software pode ser instalado permitindo que vários servidores virtuais sejam utilizados. VELTE (2011, p.07).

Não há a necessidade dos servidores estarem alocados juntos. Normalmente estes servidores estão espalhados pelo mundo, mas para o usuário, estes servidores trabalham como se estivessem no mesmo local. Isto proporciona ao fornecedor mais flexibilidade e opções de segurança. Se acontecer a perda de comunicação de algum servidor, o serviço ainda poderá ser acessado de outro servidor disponível. VELTE (2011, p.08).

Há muito mais coisas acontecendo por trás das nuvens do que simplesmente igualá-la a internet e armazenamento de informações, atualmente a computação em nuvem é classificada em três modelos de serviços. Software como Serviço (SaaS), Plataforma como um serviço (PaaS) e Infraestrutura como um serviço (IaaS).

O Software como Serviço (SaaS) nada mais é do que um aplicativo oferecido como um serviço aos clientes que acessam através da internet. Este tipo de software custa menos que comprar todo o aplicativo, resumindo, quanto menos ele for usado, mais ele será em conta.

Outra característica do Software como Serviço (SaaS) é que o cliente fica livre da obrigação de manter sua infraestrutura atualizada e funcionando, além de fornecerem uma proteção mais eficiente.

A Plataforma como um Serviço (PaaS) bem como os SaaS é outro modelo de aplicação, na Plataforma como um Serviço (PaaS) é fornecido todos os recursos e ferramentas necessárias para que desenvolvedores de software construa novos aplicativos e serviços diretamente da internet sem precisar instalá-las em seu computador pessoal.

O fator de definição que torna PaaS exclusiva é que permite que desenvolvedores desenvolvam e implementem aplicativos da Web em uma infraestrutura hospedada. Ou seja, PaaS permite aproveitar os recursos de

computação aparentemente infinitos de uma infraestrutura de nuvem (ORLANDO, 2011).

Enquanto os SaaS e PaaS oferecem aplicações aos clientes, a Infraestrutura como um Serviço (IaaS) fornece apenas recursos de Hardware, geralmente na forma de virtualização, podem ser fornecidos recursos como: Espaço Físico para servidor;

Memória; Equipamentos de rede; Ciclos de CPU; Espaço de Armazenamento.

Na forma de virtualização, vários usuários podem utilizar os recursos de uma única máquina simultaneamente. O pagamento para a utilização destes recursos é de acordo com a quantidade utilizada, ou seja, ela é ajustada de acordo com a necessidade do cliente.

5.3 Computação Reflexiva

No desenvolvimento de sistemas comerciais, o analista vai programá-lo para solucionar problemas, independentemente da quantidade de problemas. Esse sistema jamais vai modificar seu comportamento ou estrutura para solucionar um problema, pois ele irá realizar tudo o que foi programado sem fazer alterações em si mesmo, não permitindo que o sistema consiga se autoanalisar conforme novas necessidades que aparecerem.

Porém ao utilizar reflexão computacional é possível o sistema raciocinar sobre si mesmo, podendo solucionar problemas em tempo de execução, os quais somente poderiam ser resolvidos com adição de novos comandos na programação.

Segundo Barth, F. J. (2000), o conceito básico do paradigma de reflexão computacional não é exatamente nova. Este conceito originou-se em lógica matemática e recentemente, mecanismos de alto nível tornam o esquema de reflexão um aliado de características operacionais ou não funcionais a módulos já existentes.

A reflexão pode ser definida como qualquer ação executada por um sistema computacional sobre si próprio, onde tal conceito foi apresentado pela primeira vez por Maes (1987), pode-se dizer que reflexão é a capacidade de um

programa reconhecer detalhes internos em tempo de execução que não estavam disponíveis no momento da compilação do programa.

Reflexão Computacional tem dois significados distintos. Um é introspecção, que se refere ao ato de examinar a si próprio, o segundo é intercessão, onde está ligado ao redirecionamento da luz, ou seja, o termo reflexão computacional na área da informática tem a capacidade de examinar ou conhecer a si mesmo ou estrutura, podendo fazer alterações no seu comportamento através do redirecionamento ou de interceptação de operações efetuadas [Conceitos Básicos].

Segundo Barth, F. J. (2000), a reflexão Computacional define em uma nova arquitetura de software. Este modelo de arquitetura é composto por um meta-nível, onde se encontram estruturas de dados e ações a serem realizadas sobre o sistema objeto, localizadas no nível base. Neste contexto, a arquitetura de meta-nível é explorada para expressar propriedades não funcionais do sistema, de forma independente do domínio da aplicação.

A reflexão Computacional define conceitualmente uma arquitetura em níveis, denominada arquitetura reflexiva. Em uma arquitetura reflexiva, um sistema computacional é visto como incorporando dois componentes: um representando os objetos (domínio da aplicação), e outro a parte reflexiva (domínio reflexivo ou autodomínio) [Arquitetura Reflexiva].

No nível-base são encontradas as funções dos objetos, essas computações dos objetos têm a funcionalidade de resolver problemas e retornar informações sobre o domínio externo, enquanto o nível reflexivo encontra-se no meta-nível, resolvendo os problemas do nível-base, e retorna informações sobre a computação do objeto, podendo adicionar funcionalidades extras a este objeto.

Em uma arquitetura reflexiva, tem uma visão que um sistema computacional tem incorporado uma parte objeto e outra parte reflexiva. Na parte da computação objeto é realizadas funções para resolver problemas e retornar informações sobre um domínio externo, e quanto ao nível reflexivo tem a funcionalidade de resolver problemas e retornar informação sobre a computação do objeto.

5.4 Modelo da Aplicação

O sistema consiste em analisar a Tabela do Simples Nacional procurando contornar o modelo complexo lógico que torna a implementação mais lenta e por vezes falha. Com a Tecnologia Adaptativa o comportamento da aplicação se torna mais ágil e dinâmica e inteligente.

ANEXO I **(Vigência a Partir de 01.01.2012)**

Receita Bruta em 12 meses (em R$)	**Alíquota**	**IRPJ**	**CSLL**	**Cofins**	**PIS/Pasep**	**CPP**	**ICMS**
Até 180.000,00	4,00%	0,00%	0,00%	0,00%	0,00%	2,75%	1,25%
De 180.000,01 a 360.000,00	5,47%	0,00%	0,00%	0,86%	0,00%	2,75%	1,86%
De 360.000,01 a 540.000,00	6,84%	0,27%	0,31%	0,95%	0,23%	2,75%	2,33%

Figura 1. TABELA DO SIMPLES NACIONAL - Alíquotas e Partilha do Simples Nacional – Comércio

Autômato que simplifica o processo que será realizado pelo sistema

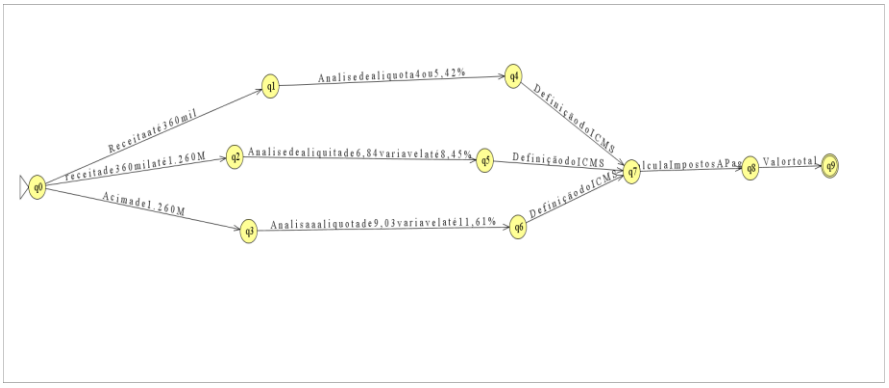

Fig 2. Automato simples da TABELA DO SIMPLES NACIONAL

Modelo Complexo do Autômato que irá representar o sistema comportamental em seu todo.

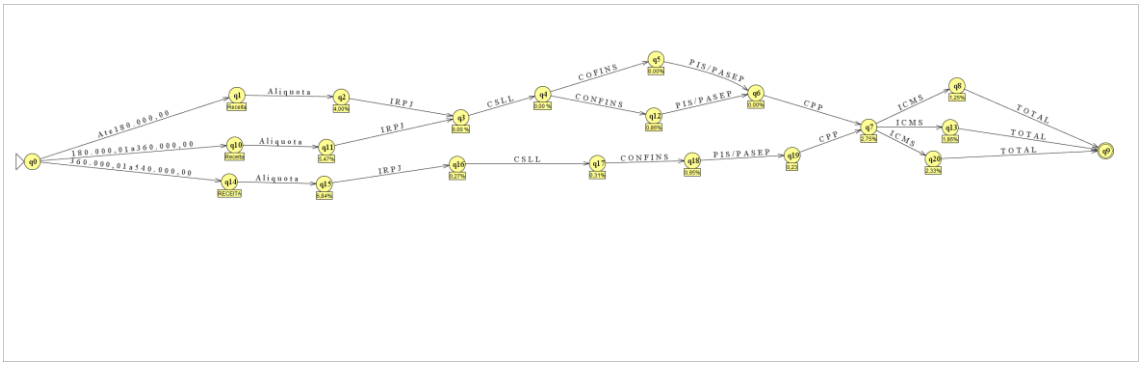

Fig 3. Automato simples da TABELA DO SIMPLES NACIONAL

Seguindo o Modelo Relacional, nossa aplicação obedece um processo padrão para armazenamento de dados passando de cliente/servidor.

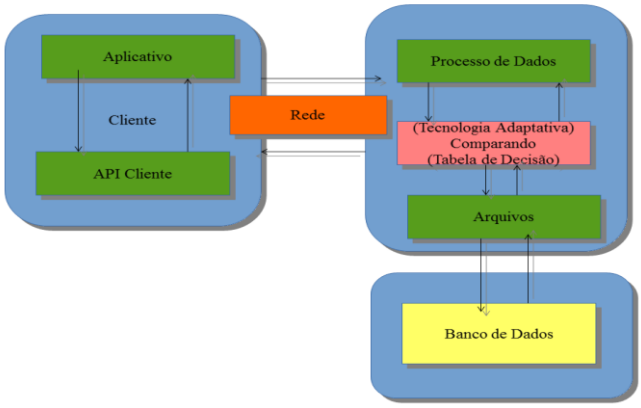

Fig 4. Modelo relacional da aplicação

Código 1. Classe CriaLançamento

```
import java.math.BigDecimal;
import java.util.Calendar;

import javax.persistence.EntityManager;
import javax.persistence.EntityTransaction;

import com.fema.financeiro.model.Lancamento;
import com.fema.financeiro.model.Pessoa;
import com.fema.financeiro.model.TipoLancamento;
import com.fema.financeiro.util.JpaUtil;

public class CriaLancamentos {
```

```
public static void main(String[] args) {
                EntityManager manager = JpaUtil.getEntityManager();
                EntityTransaction trx = manager.getTransaction();
                trx.begin();

                Calendar dataVencimento1 = Calendar.getInstance();
                dataVencimento1.set(2013, 10, 1, 0, 0, 0);

                Calendar dataVencimento2 = Calendar.getInstance();
                dataVencimento2.set(2013, 12, 10, 0, 0, 0);

                Pessoa cliente = new Pessoa();
                cliente.setNome("Rafael Sebastião Bernini");

                Pessoa fornecedor = new Pessoa();
                fornecedor.setNome("Desenvolvimento e Infra");

                Lancamento lancamento1 = new Lancamento();
                lancamento1.setDescricao("Venda de serviços");
                lancamento1.setPessoa(cliente);

lancamento1.setDataVencimento(dataVencimento1.getTime());

lancamento1.setDataPagamento(dataVencimento1.getTime());
                lancamento1.setValor(new BigDecimal(103_000));
                lancamento1.setTipo(TipoLancamento.RECEITA);

                Lancamento lancamento2 = new Lancamento();
                lancamento2.setDescricao("Venda anual");
                lancamento2.setPessoa(cliente);

lancamento2.setDataVencimento(dataVencimento1.getTime());

lancamento2.setDataPagamento(dataVencimento1.getTime());
                lancamento2.setValor(new BigDecimal(15_000));
                lancamento2.setTipo(TipoLancamento.RECEITA);

                Lancamento lancamento3 = new Lancamento();
                lancamento3.setDescricao("Treinamento");
```

```
                                        lancamento3.setPessoa(fornecedor);

                        lancamento3.setDataVencimento(dataVencimento2.getTime());
                                        lancamento3.setValor(new BigDecimal(68_000));
                                        lancamento3.setTipo(TipoLancamento.DESPESA);

                                        manager.persist(cliente);
                                        manager.persist(fornecedor);
                                        manager.persist(lancamento1);
                                        manager.persist(lancamento2);
                                        manager.persist(lancamento3);

                                        trx.commit();
                                        manager.close();
                        }

}
```

Código 2. Classe CriaLançamentoBean

```
package com.fema.financeiro.controller;

import java.io.Serializable;
import java.util.List;

import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;
import javax.faces.event.AjaxBehaviorEvent;
import javax.inject.Inject;
import javax.inject.Named;

import com.fema.financeiro.model.Lancamento;
import com.fema.financeiro.model.Pessoa;
import com.fema.financeiro.model.TipoLancamento;
import com.fema.financeiro.repository.Lancamentos;
import com.fema.financeiro.repository.Pessoas;
import com.fema.financeiro.service.CadastroLancamentos;
import com.fema.financeiro.service.NegocioException;

@Named
```

```
@javax.faces.view.ViewScoped
public class CadastroLancamentoBean implements Serializable {

		private static final long serialVersionUID = 1L;

		@Inject
		private CadastroLancamentos cadastro;

		@Inject
		private Pessoas pessoas;

		@Inject
		private Lancamentos lancamentos;

		private Lancamento lancamento;
		private List<Pessoa> todasPessoas;

		public void prepararCadastro() {
				this.todasPessoas = this.pessoas.todas();

				if (this.lancamento == null) {
						this.lancamento = new
Lancamento();
				}
		}

		public List<String> pesquisarDescricoes(String descricao) {
				return
this.lancamentos.descricoesQueContem(descricao);
		}

		public void dataVencimentoAlterada(AjaxBehaviorEvent event) {
				if (this.lancamento.getDataPagamento() == null) {

		this.lancamento.setDataPagamento(this.lancamento.getDataVencimento(
));
				}
		}

		public void salvar() {
```

```
                                        FacesContext context =
FacesContext.getCurrentInstance();

                                        try {

                        this.cadastro.salvar(this.lancamento);

                                                                this.lancamento = new
Lancamento();
                                                                context.addMessage(null, new
FacesMessage("Lançamento salvo com sucesso!"));
                                        } catch (NegocioException e) {

                                                                FacesMessage mensagem = new
FacesMessage(e.getMessage());

                        mensagem.setSeverity(FacesMessage.SEVERITY_ERROR);
                                                                context.addMessage(null,
mensagem);
                                        }
                        }

                        public List<Pessoa> getTodasPessoas() {
                                        return this.todasPessoas;
                        }

                        public TipoLancamento[] getTiposLancamentos() {
                                        return TipoLancamento.values();
                        }

                        public Lancamento getLancamento() {
                                        return lancamento;
                        }

                        public void setLancamento(Lancamento lancamento) {
                                        this.lancamento = lancamento;
                        }

}
```

Código 3. Classe ConverteLançamento

```
package com.fema.financeiro.converter;

import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.convert.Converter;
import javax.faces.convert.FacesConverter;
import javax.inject.Inject;

import com.fema.financeiro.model.Lancamento;
import com.fema.financeiro.repository.Lancamentos;

@FacesConverter(forClass = Lancamento.class)
public class LancamentosConverter implements Converter {

		@Inject
		private Lancamentos lancamentos;

		public Object getAsObject(FacesContext context, UIComponent
component, String value) {
				Lancamento retorno = null;

				if (value != null && !"".equals(value)) {
						retorno =
this.lancamentos.porId(new Long(value));
				}

				return retorno;
		}

		public String getAsString(FacesContext context, UIComponent
component, Object value) {
				if (value != null) {
						Lancamento lancamento =
((Lancamento) value);
						return lancamento.getId() == null ?
null : lancamento.getId().toString();
				}
				return null;
		}
```

```
}
```

Código 4. Classe Filtro de Lançamento

```
package com.fema.financeiro.filter;

import java.io.IOException;

import javax.inject.Inject;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.fema.financeiro.controller.Usuario;

@WebFilter("*.xhtml")
public class AutorizacaoFilter implements Filter {

		@Inject
		private Usuario autenticacao;

		public void doFilter(ServletRequest req, ServletResponse res,
							FilterChain chain) throws
IOException, ServletException {
				HttpServletResponse response =
(HttpServletResponse) res;
				HttpServletRequest request = (HttpServletRequest)
req;

				if (!autenticacao.isLogado() &&
!request.getRequestURI().endsWith("/Login.xhtml")
					&&
!request.getRequestURI().contains("/javax.faces.resource/")) {
```

```
                response.sendRedirect(request.getContextPath() + "/Login.xhtml");
                        } else {
                                chain.doFilter(req, res);
                        }
                }

                public void init(FilterConfig config) throws ServletException {
                }

                public void destroy() {
                }

}
```

Código 5. Classe Lançamento

```
package com.fema.financeiro.model;

import java.io.Serializable;
import java.math.BigDecimal;
import java.util.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import org.hibernate.validator.constraints.NotEmpty;

import com.fema.financeiro.validation.DecimalPositivo;
```

```
@Entity
@Table(name = "lancamento")
public class Lancamento implements Serializable {

		private static final long serialVersionUID = 1L;

		private Long id;
		private Pessoa pessoa;
		private String descricao;
		private BigDecimal valor;
		private BigDecimal valoriss;
		private TipoLancamento tipo;
		private Date dataVencimento;
		private Date dataPagamento;

		@Id
		@GeneratedValue
		public Long getId() {
				return id;
		}

		public void setId(Long id) {
				this.id = id;
		}

		@NotNull
		@ManyToOne(optional = false)
		@JoinColumn(name = "pessoa_id")
		public Pessoa getPessoa() {
				return pessoa;
		}

		public void setPessoa(Pessoa pessoa) {
				this.pessoa = pessoa;
		}

		@NotEmpty
		@Size(max = 80)
		@Column(length = 80, nullable = false)
```

```
public String getDescricao() {
		return descricao;
}

public void setDescricao(String descricao) {
		this.descricao = descricao;
}

@DecimalPositivo
@Column(precision = 10, scale = 2, nullable = false)
public BigDecimal getValor() {
		return valor;
}

public void setValor(BigDecimal valor) {
		this.valor = valor;
}

@NotNull
@Enumerated(EnumType.STRING)
@Column(nullable = false)
public TipoLancamento getTipo() {
		return tipo;
}

public void setTipo(TipoLancamento tipo) {
		this.tipo = tipo;
}

@NotNull
@Temporal(TemporalType.DATE)
@Column(name = "data_vencimento", nullable = false)
public Date getDataVencimento() {
		return dataVencimento;
}

public void setDataVencimento(Date dataVencimento) {
		this.dataVencimento = dataVencimento;
}
```

```
	@Temporal(TemporalType.DATE)
	@Column(name = "data_pagamento", nullable = true)
	public Date getDataPagamento() {
		return dataPagamento;
	}

	public void setDataPagamento(Date dataPagamento) {
		this.dataPagamento = dataPagamento;
	}

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((id == null) ? 0 :
id.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Lancamento other = (Lancamento) obj;
		if (id == null) {
			if (other.id != null)
				return false;
		} else if (!id.equals(other.id))
			return false;
		return true;
	}

}
```

Código 6. Classe Lançamento Repositório

```
package com.fema.financeiro.repository;

import java.io.Serializable;
import java.util.List;

import javax.inject.Inject;
import javax.persistence.EntityManager;
import javax.persistence.TypedQuery;

import com.fema.financeiro.model.Lancamento;

public class Lancamentos implements Serializable {

		private static final long serialVersionUID = 1L;

		private EntityManager manager;

		@Inject
		public Lancamentos(EntityManager manager) {
				this.manager = manager;
		}

		public Lancamento porId(Long id) {
				return manager.find(Lancamento.class, id);
		}

		public List<String> descricoesQueContem(String descricao) {
				TypedQuery<String> query = manager.createQuery(
						"select distinct
descricao from Lancamento "
						+ "where
upper(descricao) like upper(:descricao)",
						String.class);
				query.setParameter("descricao", "%" + descricao +
"%");
				return query.getResultList();
		}

		public List<Lancamento> todos() {
```

```
                                        TypedQuery<Lancamento> query =
manager.createQuery(
                                                                        "from
Lancamento", Lancamento.class);
                                        return query.getResultList();
                }

                public void adicionar(Lancamento lancamento) {
                                        this.manager.persist(lancamento);
                }

                public Lancamento guardar(Lancamento lancamento) {
                                        return this.manager.merge(lancamento);
                }

                public void remover(Lancamento lancamento) {
                                        this.manager.remove(lancamento);
                }

}
```

Os código acima representam a criação da inserção ao bando de dados e o processo de cadastro e lançamento do cliente até o servidor, até que possa ser gravados em nuvem.

6. Conclusão

Havendo capacidade intuitiva decorrente de sua experiência, pode criar um ato mais direcionada ao seu objetivo no atendimento das necessidades de suprir com forma eficiente e eficaz, sendo visionário em diminuir o impacto das incertezas, através da utilização de processos sistemáticos de tomada de decisão.

Tal modelo de projeto ainda está em fase de desenvolvimento experimental, e a confiabilidade do modelo adaptativo deverá ser em tempo alcançado pelo formalismo e suas exigências matemático proposta. Desta maneira, abre espaço para maiores avanços em pesquisas e desenvolvimento na área comercia juntamente com a tecnologia adaptativa tornando um novo método de trabalho e sendo generosamente satisfatório, tanto nos aspectos

práticos de aplicação da quanto nos processos comportamentais da Tecnologia Adaptativa.

7. Referencias

BARTH, F.J. Utilização da Reflexão Computacional para implementação de aspectos não funcionais em um gerenciador de arquivos distribuídos. Trabalho de Conclusão de Curso, Universidade Regional de Blumenau. 2000

CAMOLESI, A.R.; NETO, J.J. Modelagem Adaptativa de Aplicções Complexas. XXX Conferencia Latinoamericana de Informática - CLEI'04. Arequipa - Peru, Setiembre 27 - Octubre 1, 2004a.

GONÇALVES, J.S. CAMOLESI, A.R. O uso de programação reflexiva para o desenvolvimento de aplicações comerciais adaptativas. Relatório Final de Projeto de Iniciação Científica, FEMA, Assis/SP, 2012.

LIMA, J. C.S. Um Estudo Sobre a Reconfiguração da Função de Compras em Empresas do Setor Automotivo. Tese de Doutorado - Escola Politécnica da USP, 2004.

LUZ, J.C. Tecnologia adaptativa aplicada à otimização de código em compiladores. Dissertação de Mestrado, USP, São Paulo, 2004.

LUZ, J.C.; NETO, J.J. Tecnologia adaptativa aplicada à otimização de código em compiladores. IX Congreso Argentino de Ciencias de la Computación, La Plata,

MAES, P. Concepts and experiments in computational reflection. ACM Sigplan Notices, v. 22, n. 12, p. 147-155, Dec. 1987

MILLER, Michael. Cloud Computing: Web-Based Applications That Change the Way You Work and Collaborate Online. Que Editor, ISBN: 0789738031, 2008.

NETO J.J. Introdução a Compilação. EDITORA: LTC, Rio de Janeiro, 1987

NETO, J.J. Adaptive Rule-Driven Devices - General Formulation and Case Study. Lecture Notes in Computer Science. Watson, B.W. and Wood, D. (Eds.): Implementation and Application of Automata 6th International Conference, CIAA 2001, Springer-Verlag, Vol.2494, pp. 234-250, Pretoria, South Africa, July 23-25, 2001.

NETO, J.J. Contribuições à metodologia de construção de compiladores. Tese de Livre Docência, USP, São Paulo, 1993.

NETO, J.J. Cross-Assemblers para Microprocessadores - Geração Automática Através do SPD. I CONAI - Congresso Nacional de Automação Industrial, pp. 501-509, São Paulo, 1983.

NETO, J.J. e MAGALHÃES, M.E.S. Um Gerador Automático de Reconhecedores Sintáticos para o SPD. VIII SEMISH - Seminário de Software e Hardware, pp. 213-228, Florianópolis, 1981.

NETO, J.J. Uma Solução Adaptativa para Reconhecedores Sintáticos. Anais EPUSP - Engenharia de Eletricidade - série B, vol. 1, pp. 645-657, São Paulo, 1988.

NETO, J.J.; ALMEIDA Jr.J.R.; NOVAES, J.M. Synchronized Statecharts for Reactive Systems. Proceedings of the IASTED International Conference on Applied Modelling and Simulation, pp.246-251, Honolulu, Hawai, 1998.

NETO, J.J.; IWAI, M.K. Adaptive Automata for Syntax Learning. CLEI 98 - XXIV Conferencia Latinoamericana de Informatica, MEMORIAS. pp. 135-149, Quito, Equador, 1998.

NETO, J.J.; SILVA, P.S.M. An adaptive framework for the design of software specification language. Proceedings of the International Conferenceon Adaptive and Natural ComputINing Algorithms (ICANNGA), Springer Verlag editor, pp. 349-352, Coimbra University, Coimbra, Portugal, 21 - 23 march 2005.

NOVAES, J.M. Um formalismo adaptativo com mecanismo de sincronização para aplicações concorrentes. Dissertação de Mestrado, USP, São Paulo, 1997.

ORLANDO, Dan. Modelos de Serviços de Computação em Nuvem, Parte 2: Plataforma como Serviço. Enterprise RIA Consultant, Vision Media Group. Disponível em: <http://www.ibm.com/developerworks/br/cloud/library/cl-cloudservices2paas/>. Acesso em Julho de 2015.

PISTORI, H. Tecnologia Adaptativa em Engenharia de Computação: Estado da Arte e Aplicações. Tese de Doutorado, USP, São Paulo, 2003.

PISTORI, H.; NETO, J.J.; COSTA, E.R. Utilização de Tecnologia Adaptativa na Detecção da Direção do Olhar. SPC Magazine, v.2., n.2, pp.22-29, Lima, Perú, Maio, 2003.

VELTE, Anthony T; VELTE, Toby J.; ELSENPETER, Robert. Computação em Nuvem. Rio de Janeiro: Alta Books Editora, 2010.

Documentação do Java Disponível em <http://www.oracle.com/technetwork/pt/java/javase/documentation/index.html > Acessada Fevereiro 2015

Documentação do MySQL Disponível em <http://dev.mysql.com/doc/refman/5.6/en/> Acesso Março 2015

Documentação do JAVA. Disponível em <http://www.oracle.com/technetwork/pt/java/javase/documentation/index.html> Acesso Julho 2015.

Documentação da Tabela Simples Nacional – Alíquotas e Partilhas do Simples Nacional – Comércio <http://www.normaslegais.com.br/legislacao/simples-nacional-anexoI.html> Acesso Maio 2015