

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO
TOCANTINS CAMPUS PALMAS**

WILKINSON SATURNINO DA SILVA

SCPREVI – SISTEMA DE COBRANÇA PREVIDENCIÁRIO

PALMAS - TO

2012

WILKINSON SATURNINO DA SILVA

SCPREVI – SISTEMA DE COBRANÇA PREVIDENCIÁRIO

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do Grau de Tecnólogo em Sistemas Para Internet pelo Instituto Federal de Educação, Ciência e Tecnologia do Tocantins, Campus Palmas.

ORIENTADOR: Prof. Esp. Marinaldo Oliveira Santos.

PALMAS - TO

2012

WILKINSON SATURNINO DA SILVA

SCPREVI – SISTEMA DE COBRANÇA PREVIDENCIÁRIO

Este Trabalho de Conclusão de Curso foi julgado e aprovado como cumprimento às exigências legais do currículo do Curso Superior de Tecnologia em Sistemas Para Internet pela Coordenação da Área de Tecnologia da Informação no Instituto Federal de Educação, Ciência e Tecnologia do Tocantins – Unidade Sede.

Palmas, 23 de março de 2012.

Supervisor do Trabalho de Conclusão de Curso

Responsável Técnico

Banca Examinadora:

Presidente e Orientador

Membro de Banca Examinadora

Membro de Banca Examinadora

AGRADECIMENTOS

Agradeço a todos que me ajudaram a concluir este projeto,

Aos meus pais e minha irmã, que foram meu maior incentivo em toda esta caminhada. Agradeço por todo o esforço e apoio desde o início. Sem sombra de dúvidas, este projeto não teria sido concluído sem a vossa ajuda.

Aos professores do Instituto Federal de Educação, Ciência e Tecnologia do Tocantins por compartilhar todo seu conhecimento e ensinamentos, oferecendo esta oportunidade para seguir a carreira de Tecnólogo. Aos amigos, por estarem sempre ao meu lado me apoiando. Em especial um agradecimento à minha companheira, por toda a paciência e compreensão de minha ausência durante o desenvolvimento deste projeto. Muito obrigado!

RESUMO

Este projeto tem como objetivo demonstrar como foi desenvolvido o SCPREVI (Sistema de Cobrança Previdenciário), sistema com tecnologia *web* que foi desenvolvido com base nas necessidades do Instituto de Previdência Social do Município de Palmas – TO (Previpalmas). Todo o desenvolvimento do projeto foi baseado em técnicas expressadas no Processo Unificado (PU).

Este projeto apresenta não só a fase de desenvolvimento do sistema como também introduz a problemática na qual foi o incentivo para o trabalho. Apresenta a especificação do sistema, onde é explanada a parte teórica do desenvolvimento. Não menos importante, é demonstrado também, quais tecnologias e ferramentas foram utilizadas para se chegar ao produto final.

Palavras-chave: Boleto, Cobrança, Cedente, Servidor, Java, Sistema.

ABSTRACT

This project have to demonstrate how we developed the SCPREVI (Sistema de Cobrança Previdenciário, translated into English as: Charging System Pension), system with web technology that was developed based on the needs of Instituto de Previdência Social do Município de Palmas – TO, translated into English as: Social Security Institute in the city of Palmas - TO (Previpalmas). All development of the project was based on techniques expressed in the Unified Process (UP).

This project has not only the development phase of the system but also introduces the problem in what was the incentive to work. Displays the system specification, which is explained the theoretical development. No less important, is also shown, what technologies and tools were used to reach the final product.

Key-words: *Billet, Collection, Assignor, Server, Java, System.*

LISTA DE ILUSTRAÇÕES

Figura 1 - Fases do Processo Unificado.....	23
Figura 2 - Diagrama Entidade Relacionamento – SCPREVI.	43
Figura 3 - Ambiente Java.	44
Figura 4 - Arquitetura do <i>Hibernate</i>	46
Figura 5 - Arquivo de configuração do <i>Hibernate</i> do SCPREVI.	47
Figura 6 - Arquivo de configuração do <i>Data Source</i> de conexão do SCPREVI.	48
Figura 7 - Configuração do <i>Data Source</i> no arquivo “web.xml” do SCPREVI.....	49
Figura 8 - Classe “ <i>InterfaceHibernateDAO.java</i> ” do SCPREVI.....	50
Figura 9 - Classe “ <i>HibernateDAO.java</i> ” do SCPREVI.....	50
Figura 10 - Classe “ <i>HibernateUtil.java</i> ” do SCPREVI.	51
Figura 11 – Parte da classe “ <i>DAOFactory.java</i> ” do SCPREVI.....	52
Figura 12 - Representação da arquitetura entre camadas	52
Figura 13 - Classe “ <i>PessoaRN.java</i> ” do SCPREVI.....	53
Figura 14 - Arquitetura do padrão de projeto MVC.....	55
Figura 15 - Filtro “ <i>ConexaoHibernateFilter.java</i> ” do SCPREVI.	56
Figura 16 - Ciclo de vida do JSF.	59
Figura 17 - Demonstração de configuração do componente “<p:dataTable>”.....	60
Figura 18 - <i>Templates Facelets</i> do SCPREVI.	61
Figura 19 - Invocação de <i>template</i> SCPREV.....	61
Figura 20 - Exemplo de implementação de <i>template</i> SCPREVI.	62
Figura 21 - Configuração do “div” rodapé da <i>template</i> “proprietário_principal.xhtml” do SCPREVI.....	62
Figura 22 - Funcionamento do <i>Spring Security</i>	63
Figura 23 - Compartilhamento de conexão via <i>Data Source</i> JNDI.....	64
Figura 24 - Arquivo “applicationContext.xml” do SCPREVI.	64
Figura 25 - Arquivo “applicationContext-security.xml” do SCPREVI.	64
Figura 26 - Configuração do <i>Spring Security</i> no arquivo “web.xml” do SCPREVI.....	65
Figura 27 - Restrição de conteúdo por usuário no arquivo “principal.xhtml” do SCPREVI...	66
Figura 28 - Configuração de cálculo de atualização monetária para servidor cedido – SCPREVI.....	68
Figura 29 - Continuação do método “atualizacaoMonetaria” para servidor cedido – SCPREVI.....	69
Figura 30 - Invocação do método “BoletoViwer” do <i>Bopepo</i> para geração do boleto SCPREVI.....	69
Figura 31 - Método “geraBoleto” de servidor cedido – SCPREVI.	70
Figura 32 - Estrutura de <i>Layout</i> de arquivo XML do <i>Texgit</i>	72
Figura 33 - <i>Layout</i> de arquivo remessa do SCPREVI.....	73
Figura 34 - Método para setar dados de acordo com <i>layout</i> XML – SCPREVI.....	74

LISTA DE ABREVIATURAS E SIGLAS

AJAX	<i>Asynchronous JavaScript and XML</i>
API	<i>Application Programming Interface</i>
CRUD	<i>Create, Read, Update and Delete</i>
CSS	<i>Cascading Style Sheets</i>
CSV	<i>Comma Separated Values</i>
DAO	<i>Data Access Object</i>
EJB	<i>Enterprise Java Beans</i>
HTML	<i>Hypertext Markup Language</i>
HTTP	<i>HyperText Transfer Protocol</i>
IDE	<i>Integrated Development Environment</i>
JCP	<i>Java Community Process</i>
JDK	<i>JavaSE Development Kit</i>
JNDI	<i>Java Naming and Directory Interface</i>
JPA	<i>Java Persistence API</i>
JRE	<i>Java Runtime Environment</i>
JSF	<i>Java Server Faces</i>
JSP	<i>Java Server Pages</i>
JSTL	<i>Java Standard Tag Library</i>
JVM	<i>Java Virtual Machine</i>
LIP	<i>Licença por Interesses Particulares</i>
MVC	<i>Model, View and Controller</i>
ORM	<i>Object Relational Mapping</i>
PDF	<i>Portable Document Format</i>
POJO	<i>Plain and Old Java Object</i>
PU	<i>Unified Process</i>

RGPS	Regime Geral de Previdência Social
RPPS	Regime Próprio de Previdência Social
RUP	<i>Rational Unified Process</i>
UI	<i>User Interface</i>
UML	<i>Unified Modeling Language</i>
URL	<i>Uniform Resource Location</i>
XHTML	<i>Extensible Hypertext Markup Language</i>
XLS	<i>Excel Spreadsheet</i>
XML	<i>Extensible Markup Language</i>

SUMÁRIO

1 INTRODUÇÃO	13
1.1 TEMA DA PESQUISA	14
1.1.1 Delimitação do Tema.....	14
1.2 PROBLEMA	14
1.3 HIPÓTESES	16
1.4 OBJETIVOS	18
1.4.1 Objetivo Geral	18
1.4.2 Objetivos Específicos	18
1.5 JUSTIFICATIVA	18
1.6 ESTRUTURA DO TRABALHO.....	19
2 ESPECIFICAÇÃO DO SISTEMA	21
2.1 VISÃO GERAL DO PRODUTO	21
2.2 DOCUMENTO DE REQUISITOS.....	23
2.2.1 Requisitos Funcionais	23
2.2.2 Requisitos Não Funcionais	26
2.2.3 Regras de Negócio.....	26
2.2.4 Modelagem de Casos de Uso	27
2.2.5 Linguagem UML	34
2.2.5.1 Diagrama de Casos de Uso.....	34
2.2.5.2 Diagrama de Atividades.....	35
2.2.5.3 Diagrama de Classes de Domínio	35
2.2.5.4 Diagrama de Sequência	36
2.2.6 Glossário	37
3 APORTE TEÓRICO.....	41
3.1 Base de Dados	41
3.1.1 MySQL Workbench	42
3.2 Linguagem Java.....	43
3.2.1 <i>Framework</i> de Persistência <i>Hibernate</i>	45
3.2.2 Padrão de Projeto DAO	49
3.2.3 Padrão de Projeto MVC.....	54
3.2.4 Técnica Open Session In View.....	55
3.2.5 Framework de desenvolvimento <i>JavaServer Faces</i>	57
3.2.6 Biblioteca de componentes <i>PrimeFaces</i>	59

3.2.7 Utilizando <i>Templates</i> com <i>Facelets</i> e Linguagem CSS	61
3.2.8 Framework de Segurança Spring Security	62
3.2.9 Projeto <i>JRimum - Bopepo</i>	66
3.2.10 Projeto <i>JRimum - Texgit</i>	71
3.2.11 Leitura de Arquivo Retorno.....	74
3.4 Servidor Web Apache Tomcat	76
3.5 Ambiente de Desenvolvimento NetBeans IDE	77
4 CONCLUSÕES E PROJETOS FUTUROS	78
4.1 CONCLUSÕES	78
4.2 PROJETOS FUTUROS	78
REFERÊNCIAS	80
ANEXO A – TELAS DO CASO DE USO MANter COBRANÇA DE SERVIDOR SCPREVI	82
A.1 TELA 01 – SCPREVI-CDU-MCSERV	82
A.2 TELA 02 – SCPREVI-CDU-MCSERV	82
A.3 TELA 03 – SCPREVI-CDU-MCSERV	83
A.4 TELA 04 – SCPREVI-CDU-MCSERV	83
A.5 TELA 05 – SCPREVI-CDU-MCSERV	84
A.6 TELA 06 – SCPREVI-CDU-MCSERV	84
A.7 TELA 07 – SCPREVI-CDU-MCSERV	85
A.8 TELA 08 – SCPREVI-CDU-MCSERV	85
A.9 TELA 09 – SCPREVI-CDU-MCSERV	86
A.10 TELA 10 – SCPREVI-CDU-MCSERV	86
A.11 TELA 11 – SCPREVI-CDU-MCSERV	87
A.12 TELA 12 – SCPREVI-CDU-MCSERV	87
A.13 TELA 13 – SCPREVI-CDU-MCSERV	88
A.14 TELA 14 – SCPREVI-CDU-MCSERV	88
A.15 TELA 15 – SCPREVI-CDU-MCSERV	88
A.16 TELA 16 – SCPREVI-CDU-MCSERV	89
A.17 TELA 17 – SCPREVI-CDU-MCSERV	89
A.18 TELA 18 – SCPREVI-CDU-MCSERV	90
A.19 TELA 19 – SCPREVI-CDU-MCSERV	90
A.20 TELA 20 – SCPREVI-CDU-MCSERV	91
A.21 TELA 21 – SCPREVI-CDU-MCSERV	91
A.22 TELA 22 – SCPREVI-CDU-MCSERV	92
A.23 TELA 23 – SCPREVI-CDU-MCSERV	92
ANEXO B – DIAGRAMAS DO SCPREVI.....	93
B.1 Diagrama de Casos de Uso SCPREVI.....	93
B.2 Diagrama de Atividades SCPREVI.....	94
B.3 Diagrama de Classes de Domínio SCPREVI	95

B.4 Diagrama de Sequência SCPREVI I	96
B.5 Diagrama de Sequência SCPREVI II	96

1 INTRODUÇÃO

Um dos cargos mais disputados no Brasil e Tocantins é o de servidor público, pois com ele, tem-se como garantia a estabilidade e boa remuneração, de acordo com o caráter do cargo. Dessa forma, existem inúmeros direitos e deveres desses servidores, estando na posição de efetivos do município. Neste âmbito, se encaixa perfeitamente um dos princípios da seguridade social, o princípio da obrigatoriedade¹. Ou seja, os servidores dignos de um cargo efetivo devem, por obrigação e, na forma da lei, contribuir parte de seu salário à previdência, com o intuito de gozar de benefícios futuramente, servindo assim, de entusiasmo para esses servidores.

A contribuição desses servidores está ligada diretamente a um regime de previdência social. Neste caso, o Regime Próprio de Previdência Social (RPPS), o qual é “comandado” pelo ente gestor, responsável por garantir a proteção dos contribuintes. A entidade do qual se trata a afirmação chama-se Previpalmas, responsável por garantir a seguridade de todos os servidores efetivos do município, de acordo com o disposto no *caput* da lei nº 1.414/2005 de Palmas – TO². Dessa forma, esses servidores contribuintes passam a ser chamados de beneficiários que, por sua vez, são classificados em segurados e dependentes.

Os beneficiários na qualidade de segurados apresentam benefícios os quais assim como as obrigações, são baseados em leis fundamentadas e específicas a cada município. Assim, os segurados dispõem de alguns benefícios como, por exemplo, tratar de interesses particulares durante um tempo determinado ou prestar serviços a outra entidade, ambos sem perder o vínculo com o Previpalmas. Entretanto, esses benefícios são contornados por regras estabelecidas em lei e que, portanto, devem ser cumpridas, caso contrário serão impostas consequências negativas ao segurado, com o principal intuito de seguir os princípios da seguridade social. Dessa forma, uma das principais dificuldades das entidades de previdência como o Previpalmas é manter a proteção de seus segurados de acordo com seus comportamentos perante a previdência.

¹ “A participação dos membros da coletividade nas ações de seguridade social, para que esta efetivamente atinja os seus objetivos, deve ser imposta obrigatoriamente”. (DIAS e MACÊDO, 2010, p. 95).

² “Dispõe sobre o Regime Próprio de Previdência Social do Município e sobre a entidade de previdência e dá outras providências”. (Lei nº 1.414/2005 de Palmas – TO).

1.1 TEMA DA PESQUISA

SCPREVI – Sistema de Cobrança Previdenciário.

1.1.1 Delimitação do Tema

Desenvolvimento de um sistema com tecnologia *web* para o Previpalmas, a fim de gerenciar os servidores de Licença por Interesses Particulares (LIP) e cedidos, mantendo o controle e a cobrança de suas contribuições e, também, o gerenciamento dos servidores exonerados.

1.2 PROBLEMA

Os servidores efetivos do município, os chamados segurados (ou seja, contribuintes ao RPPS), apresentam como um de seus direitos básicos de servidor público, a possibilidade de entrar de licença para o trato de interesses particulares, bem como podem ser cedidos ou afastados para outras entidades da União, dos Estados, do Distrito Federal e de Municípios. Dessa forma, entende-se que, mesmo afastados, esses servidores não perdem o vínculo com a entidade responsável pela gestão de seu regime (RPPS), neste caso o Previpalmas, pois de acordo com o artigo 17 da lei nº 1.414/2005 de Palmas – TO:

Permanece filiado ao Regime Próprio de Previdência, na qualidade de segurado, o servidor ativo que estiver:
I – afastado ou licenciado temporariamente do exercício do cargo efetivo sem recebimento de subsídio ou remuneração, nas hipóteses e nos prazos para afastamento ou licenciamento previstos em Lei;
II – cedido a órgão ou entidade da administração direta ou indireta da União, dos Estados, do Distrito Federal e de Municípios.

O Previpalmas tem como responsabilidades não só o recolhimento de contribuições ao RPPS dos servidores efetivos e ativos, como também, as contribuições dos servidores de LIP. Ou seja, existem servidores de LIP, que, mesmo sem o recebimento de subsídio ou remuneração durante o período de licença, e, portanto, sem a obrigação de continuar contribuindo ao regime, continuam mantendo essa contribuição. Entretanto, caso não continuem, para fins de benefícios futuros, será descontado para o cálculo do tempo de licença, o tempo de contribuição, de acordo com o artigo 17, § 1º e art. 48, parágrafo único da lei nº 1.414/2005 de Palmas – TO:

Art. 17, § 1º - “O servidor, na hipótese do inciso I deste artigo, poderá promover o recolhimento tempestivo das contribuições previdenciárias

próprias e das relativas ao órgão ou entidade de vinculação, para cumprimento do disposto no parágrafo único do art. 48”.

Art. 48, parágrafo único - O servidor afastado ou licenciado do cargo efetivo sem recebimento de remuneração de que trata o art. 17, inciso I, somente contará o respectivo tempo de afastamento ou licenciamento, para fins de aposentadoria, mediante o recolhimento mensal das contribuições previdenciárias próprias e das relativas ao órgão ou entidade de vinculação.

Assim, seria possível o Previpalmas gerenciar essas contribuições para ter certeza que o servidor LIP está em dia com suas obrigações e, caso não esteja, realizar a cobrança de forma informatizada?

Juntamente a isto, o Instituto também tem a necessidade de gerenciar os servidores cedidos a outros órgãos, pois não há o controle informatizado sobre suas contribuições. Logo, o Previpalmas não apresenta o registro e gerenciamento das informações de cada servidor que está à disposição de outros órgãos, mas ainda vinculados ao instituto, estando estes em dia ou não com suas contribuições. Assim, entende-se que o recolhimento das contribuições desses servidores é repassado à entidade que gerencia o regime de origem, neste caso o RPPS, pelo órgão cessionário, ou seja, a entidade a qual o servidor está cedido. Dessa forma, existe uma possibilidade de gerenciar essas contribuições e cobrar o servidor de forma informatizada, caso esteja em atraso?

Seguindo esta linha de raciocínio, a partir de uma pesquisa realizada, sobretudo no setor de Contabilidade do Instituto, foram obtidas as informações de que, todo este processo de gerenciamento de contribuições de servidores LIP e cedidos é feito de forma manual, através de *softwares* como *Microsoft Office Excel*, por exemplo. Isto torna dificultoso o controle das contribuições desses servidores e implica em falta de segurança das transações, pois não há solução informatizada para este fim no Previpalmas.

O Instituto deve proceder também, com a restituição de contribuições dos servidores exonerados, efetivos do estado. Ou seja, deve ser feita a compensação previdenciária de cada servidor exonerado, com o fim de repassar o valor contribuído pelo servidor ao Regime Geral de Previdência Social (RGPS) para o RPPS, pois o servidor contribuiu para o regime, mas não gozou de nenhum benefício, de acordo com o artigo 4º da lei nº 9.798/1999, que diz:

Cada regime próprio de previdência de servidor público tem direito, como regime instituidor, de receber do Regime Geral de Previdência Social (RGPS), enquanto regime de origem, compensação financeira, observado o disposto neste artigo.

Todo este procedimento é realizado atualmente por dois sistemas no Previpalmas: o SIPREV³, utilizado para proceder com o Cálculo Atuarial e o COMPREV⁴, utilizado para proceder com a Compensação Previdenciária. Entretanto, o Previpalmas não apresenta uma maneira de registrar os dados dos servidores exonerados além desses dois sistemas citados, que executam apenas os cálculos necessários. Dessa forma, seria possível auxiliar estes dois procedimentos, dando mais segurança ao tráfego de informações, tornando o processo mais rápido e produtivo e possibilitando a consulta e exportação de dados desses servidores?

1.3 HIPÓTESES

De acordo com pesquisa realizada no Previpalmas, algumas soluções foram encontradas com base nas problemáticas citadas anteriormente. Todo o problema em si, poderia ser resolvido com o desenvolvimento e posterior implantação de um sistema com tecnologia *web*, que possibilite o gerenciamento de dados dos servidores exonerados e cobrança de contribuições dos servidores LIP e cedidos.

Para realizar a cobrança desses servidores (LIP e cedidos) seria interessante o desenvolvimento de uma solução no sistema que gere automaticamente boletos de cobrança com base no valor da contribuição e dos juros aplicados. Atualmente, de acordo com a Febraban⁵, existem duas formas de cobrança por boletos bancários:

- **Boletos Registrados:** este tipo de cobrança permite a comunicação da aplicação do cedente⁶ com o banco, a fim de realizar a conferência dos dados, que é feita através de um arquivo denominado *arquivo remessa*, vide **Tabela 5**, TG07. Dessa forma, no momento em que é gerado o boleto de cobrança, o banco já tem todas suas informações e, de acordo com configuração efetuada pelo próprio cedente, caso o

³ Sistema Integrado de Informações Previdenciárias.

⁴ Sistema de Compensação Previdenciária entre o RGPS e o RPPS.

⁵ Federação Brasileira de Bancos. Instituição responsável pela elaboração do padrão de arquivos para a troca de informações entre cliente e banco. Padrão este, seguido por todos os bancos obrigatoriamente.

⁶ Vide **Tabela 05**, TG02.

sacado (neste caso um servidor LIP ou cedido, vide **Tabela 05**, TG12), atrase o pagamento do boleto, será possível executar o protesto do título;

- **Boletos Não Registrados:** este tipo de cobrança não permite comunicação com o banco. Após a geração do boleto, todo o processo de segurança da transação é de total responsabilidade do cedente. Assim, o banco somente terá as informações do sacado quando o mesmo proceder com o pagamento da dívida, seja este pagamento executado em dia ou não.

Para este projeto foi escolhida então, a cobrança por boletos registrados que, como dito na descrição, permite comunicação com o banco do cedente. Ou seja, o banco fica a par de todas as informações do sacado no momento em que é gerado o seu boleto de cobrança. Isto por que a aplicação deve gerar não só o boleto de cobrança, como também um arquivo remessa (arquivo que serve para comunicação com o banco), que segue um padrão exigido pela Febraban, o que torna a transação segura.

Vale lembrar que o desenvolvimento da comunicação das duas aplicações é de total obrigação do banco do cedente, pois o cedente como cliente do banco tem esse direito. Caberia também, outra funcionalidade do sistema em relação aos boletos de cobrança dos servidores: desenvolvimento do lado cliente no sistema, para permitir o acesso dos próprios servidores (LIP ou cedidos) aos seus boletos, e visualizá-los ou baixá-los diretamente no *browser* para posteriormente proceder com o pagamento, caso o boleto em questão ainda esteja com a situação marcada como “pendente”, por exemplo.

Para auxiliar o Previpalmas na compensação previdenciária e cálculo atuarial, feita com base nos dados dos servidores exonerados, seria interessante também, o desenvolvimento de uma funcionalidade no sistema que gerencie os dados dos servidores exonerados. Assim, após informação buscada no diário oficial do município, de que um servidor foi exonerado, o funcionário do Previpalmas poderia cadastrar as informações no SCPREVI. Dessa forma, no momento em que for executar o procedimento de compensação (feito no COMPREV), por exemplo, seria possível fazer uma consulta específica pelo servidor no SCPREVI e gerar relatórios

de informações do servidor selecionado. Isto tornaria mais produtivo o processo além de garantir segurança das informações dos servidores exonerados.

1.4 OBJETIVOS

1.4.1 Objetivo Geral

Este projeto tem como objetivo principal o desenvolvimento de um sistema com tecnologia *web* que esteja pronto para comunicação com o banco do cedente. Ou seja, como foi dito anteriormente, esta comunicação entre as aplicações é de total obrigação do banco. Assim, este sistema irá suprir as necessidades do Previpalmas em relação ao controle de cobranças de servidores LIP e cedidos, e não menos importante, servir de apoio à compensação previdenciária e atualização monetária através de consulta e geração de relatórios de servidores exonerados.

1.4.2 Objetivos Específicos

- Gerenciar as contribuições dos servidores LIP e cedidos, efetivos do município;
- Caso esses servidores atrasem suas contribuições, aplicar cálculos de atualização monetária (juros, multas e moras) e gerar os boletos de cobrança com esses valores atualizados;
- Disponibilizar os boletos gerados para *download* no navegador;
- Gerar arquivo remessa após a geração dos boletos com o objetivo de manter a comunicação com o banco do cedente;
- Estar pronto para receber o arquivo retorno e verificar se o título está nos padrões e se foi aceito e registrado pelo banco ou não;
- Permitir cadastro e consulta específica de servidores exonerados;
- Geração e exportação de relatórios de servidores exonerados.

1.5 JUSTIFICATIVA

O Previpalmas na posição de entidade gestora do RPPS em Palmas, deve proceder com a atualização de valores de dívidas dos servidores (ou simplesmente, atualização monetária), caso esses servidores (LIP ou cedidos) atrasem suas contribuições que será conferida sempre aos dias quinze (15) de cada mês subsequente ao da competência. Isso pode ser constatado no § 6º do artigo 17 e no artigo 86 da lei nº 1.414/2005 de Palmas – TO:

Art. 17, § 6º - No caso de atraso no recolhimento das contribuições previstas nos parágrafos 1º e 2º deste artigo, aplica-se o disposto no parágrafo único do art. 86;

Art. 86 – A responsabilidade pelo recolhimento e repasse das contribuições dos segurados em atividade e do Município, de suas autarquias e fundações para a Coordenação de Previdência - Previpalmas será do dirigente máximo do órgão ou entidade em que o segurado estiver vinculado e deverá ocorrer até o 15º (décimo quinto) dia do mês subsequente ao da competência. *Parágrafo Único* – Em caso de mora no recolhimento ou no repasse das contribuições devidas pelos segurados ou pelo Município, suas autarquias e fundações, a Coordenação de Previdência - Previpalmas, incidirão juros, multas e atualizações sobre o valor originalmente devido, calculado sob o mesmo regime aplicável às hipóteses de não pagamento de tributos municipais, sem prejuízo das cominações legais.

O § 1º citado no § 6º do art. 17, como foi frisado na problemática deste projeto, informa que o servidor LIP poderá continuar contribuindo mesmo sem o recebimento de provimentos. Já o § 2º diz que:

Incumbe ao cessionário, na hipótese dos incisos II e III deste artigo, promover o recolhimento das contribuições previdenciárias devidas originariamente pelo cedente e o repasse desses valores ao Regime Próprio de Previdência Social de origem do servidor cedido.

Ou seja, é de total obrigação da entidade a qual o servidor está cedido, o recolhimento de suas contribuições e repasse ao Previpalmas, entidade gestora do RPPS. Dessa forma, vê-se que o Instituto deve aplicar a atualização monetária⁷ para estes dois tipos de servidores (LIP e cedidos), caso atrasem suas contribuições.

Com base nas referências aqui citadas, o Previpalmas deve por obrigação realizar o gerenciamento e controle das contribuições dos servidores LIP e cedidos, e gerenciar os servidores exonerados. Portanto, uma solução para garantir todo esse gerenciamento é o desenvolvimento do sistema aqui proposto: o SCPREVI.

1.6 ESTRUTURA DO TRABALHO

Este projeto está organizado em três capítulos. Após a introdução no capítulo um, faz-se uma visualização da especificação do sistema no capítulo dois, frisando as técnicas utilizadas para o estudo de caso, e também apresenta uma amostra de todos os diagramas, facilitando não só o entendimento do leitor, como também o desenvolvimento do sistema em si. Neste capítulo também estão dispostos tópicos dedicados a exploração das ferramentas utilizadas para o desenvolvimento do

⁷ Vide **Tabela 5**, TG09.

sistema, explicando claramente como foram implementados os principais processos. No capítulo três, encontram-se as conclusões e resultados do trabalho e, também projetos futuros.

2 ESPECIFICAÇÃO DO SISTEMA

2.1 VISÃO GERAL DO PRODUTO

O produto final deste trabalho tem como objetivo prover as necessidades do Previpalmas em relação à cobrança de servidores LIP e cedidos, principalmente. Como frisado anteriormente, este projeto também irá gerenciar o trâmite dos dados de servidores exonerados. Abaixo, alguns benefícios que o sistema fornecerá ao Instituto:

- Maior agilidade e, sobretudo produtividade no desenvolvimento do trabalho em relação à cobrança de servidores LIP e cedidos. Isto é garantido pelo SCPREVI a partir da geração de boletos de contribuições atrasadas de servidores e, no momento em que é gerado o título, também já será feito o cálculo de atualização monetária, o que torna o processo ágil;
- Usabilidade e praticidade beneficiando, sobretudo os próprios servidores LIP ou cedidos, através da disponibilização de seus boletos para *download*. Dessa forma, os servidores podem acessar de qualquer lugar o sistema, pois o mesmo apresenta tecnologia *web* e estará disponibilizado na *Internet*;
- Garantia de segurança em relação às transações de cada boleto gerado, pois o SCPREVI irá utilizar a cobrança registrada;
- Organização e segurança dos dados de servidores exonerados, através de uma funcionalidade que permite a listagem e exportação de dados de cada servidor exonerado cadastrado no sistema, para agilizar o processo executado nos sistemas SIPREVI e COMPREV.

Para a implementação desse sistema foi adotada uma técnica de desenvolvimento de sistemas que apresenta características evolutivas e iterativas, conhecida como Processo Unificado (PU) e normalmente chamado também de Processo Unificado Racional (RUP), do Inglês *Rational Unified Process*. Como afirmado, o PU é um processo iterativo, o que implica no desenvolvimento a partir de realimentação de informações em cada iteração.

O desenvolvimento iterativo e evolutivo – em contraposição ao ciclo de vida seqüencial ou em cascata – envolve a imediata programação e teste de um

sistema parcial em ciclos repetidos. Também considera normalmente que o desenvolvimento começa antes que os requisitos tenham sido definidos em detalhe; a realimentação é usada para esclarecer e aperfeiçoar as especificações em evolução. (LARMAN, 2007, p. 46).

Dessa forma, entende-se que em cada iteração é possível ter uma noção do andamento do projeto, sendo desenvolvida tanto a parte de documentação e modelagem – descrição de casos de uso, diagramas, requisitos parciais, dentre outros – como a codificação do sistema. Assim, para Larman, os requisitos evoluem ao longo das iterações, sendo eles refinados com o objetivo de, no final, responder com confiança: o que, quanto e quando. Pressman divide o PU em cinco fases (vide **Figura 1**) e às descreve da seguinte maneira:

Fase de concepção – envolve tanto a atividade de comunicação com o cliente como a de planejamento. Colaborando com os interessados, identificam-se as necessidades de negócio para o software;
Fase de elaboração – envolve atividades de comunicação e modelagem do modelo de processo genérico. A elaboração refina e expande os casos práticos preliminares, desenvolvidos como parte da fase de concepção e amplia a representação da arquitetura, incluindo cinco visões diferentes do *software*: modelo de caso prático, modelo de requisitos, modelo de projeto, modelo de implementação e modelo de emprego;
Fase de construção – tendo como entrada o modelo de arquitetura, a fase de construção desenvolve ou adquire componentes de *software*. Esses componentes farão com que cada caso prático (de uso) se torne operacional para os usuários finais;
Fase de transição – esta fase abarca os últimos estágios da atividade da construção genérica e a primeira parte da atividade de emprego genérico: entrega e realimentação (*feedback*). Entrega-se o *software* aos usuários finais para testes beta e o *feedback* dos usuários relata defeitos e mudanças necessárias;
Fase de produção – coincide com a atividade de emprego do processo genérico. Durante essa fase, monitora-se o uso contínuo do *software* disponibiliza-se suporte para o ambiente (infra-estrutura) operacional, realiza-se e avalia-se relatórios de defeitos e solicitações de mudanças. (PRESSMAN, 2011, p.72).

Dessa forma, vê-se que Pressman conceituou de forma clara todas as fases do PU, demonstrando claramente que durante todas as fases existe, de fato, a iteração e realimentação e, por consequência positiva, nunca será desenvolvido antes uma documentação “completa” para posterior codificação do sistema, como ocorre no modelo de processo em cascata. Portanto, durante o desenvolvimento do SCPREVI, em determinada iteração foram concluídos os requisitos funcionais e não funcionais parciais, os quais foram descritos e expostos no pré-projeto de qualificação.

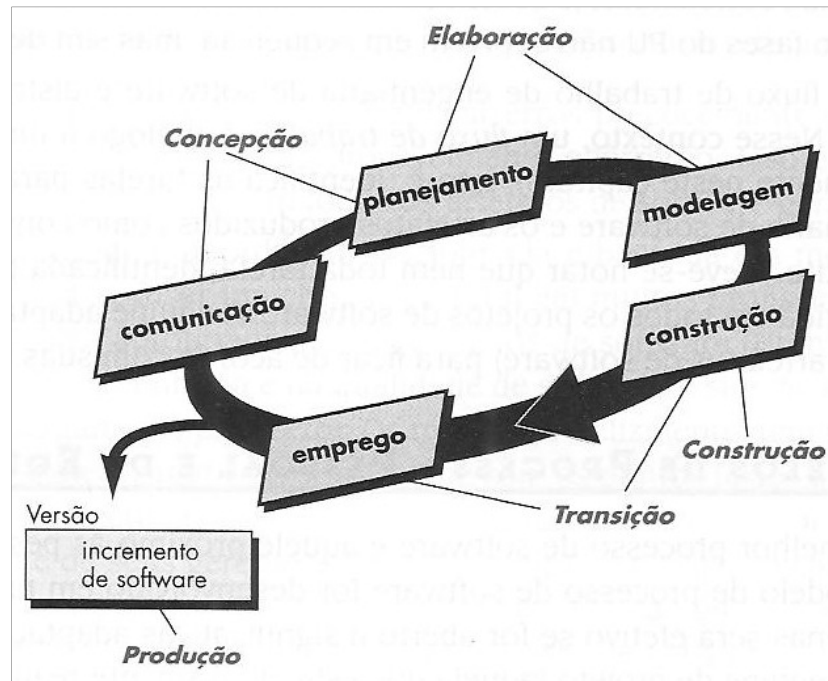


Figura 1 - Fases do Processo Unificado.
Fonte: Pressman, 2011, p. 73.

2.2 DOCUMENTO DE REQUISITOS

2.2.1 Requisitos Funcionais

Os requisitos funcionais de *software* expressam um modelo comportamental de todo o sistema, descrevendo cada funcionalidade do mesmo.

São as declarações de serviços que o sistema deve fornecer como o sistema deve reagir a entradas específicas e como o sistema deve se comportar em determinadas situações. Em alguns casos, os requisitos funcionais podem também estabelecer explicitamente o que o sistema não deve fazer. (SOMMERVILLE, 2007, p. 80).

Assim, com o auxílio da modelagem de casos de uso e dos diagramas UML, foi possível chegar aos requisitos finais do SCPREVI. Estes, por sua vez, fazem referência a determinados passos dos casos de uso. Entretanto, por questões de regras de projeto, chegou-se a conclusão de que não seria interessante postar todas as descrições de casos de uso no texto.

Assim, seguem os requisitos funcionais do SCPREVI logo abaixo:

Tabela 2: Requisitos Funcionais SCPREVI.

Requisito	Descrição	Caso de Uso
RF01	O sistema deverá permitir que o usuário com	SCPREVI-CDU-

	permissão <code>ROLE_ADMINISTRADOR</code> efetue o processo de autenticação.	MSERV , P01.
RF02	O sistema deverá permitir que o usuário com permissão <code>ROLE_ADMINISTRADOR</code> autenticado, cadastre, visualize, edite e exclua os dados de proprietário.	SCPREVI-CDU-MPROP , P01, A01, A02, A03.
RF03	O sistema deverá permitir que o usuário com permissão <code>ROLE_ADMINISTRADOR</code> autenticado, crie outros usuários com esta mesma permissão de proprietários cadastrados.	SCPREVI-CDU-MPROP , A04.
RF04	O sistema deverá permitir que o usuário com permissão <code>ROLE_ADMINISTRADOR</code> autenticado, visualize, edite e exclua os usuários de proprietários.	SCPREVI-CDU-MPROP , A05, A06, A07.
RF05	O sistema deverá permitir que o usuário com permissão <code>ROLE_ADMINISTRADOR</code> autenticado, consulte usuários específicos e proprietários específicos cadastrados no sistema.	SCPREVI-CDU-MPROP , A08, A09.
RF06	O sistema deverá permitir que o usuário com permissão <code>ROLE_ADMINISTRADOR</code> autenticado, cadastre, visualize, edite e exclua os dados de servidor.	SCPREVI-CDU-MSERV , P01, A01, A02, A03.
RF07	O sistema deverá permitir que o usuário com permissão <code>ROLE_ADMINISTRADOR</code> autenticado, crie usuários com permissão <code>ROLE_SERVIDOR</code> para servidores cadastrados.	SCPREVI-CDU-MSERV , A04.
RF08	O sistema deverá permitir que o usuário com permissão <code>ROLE_ADMINISTRADOR</code> autenticado, visualize, edite e exclua os usuários de servidores.	SCPREVI-CDU-MSERV , A05, A06, A07.
RF09	O sistema deverá permitir que o usuário com permissão <code>ROLE_ADMINISTRADOR</code> autenticado, efetue geração de boletos de cobrança de servidor.	SCPREVI-CDU-MCSERV , P01. (Vide Tabela 1).
RF10	O sistema deverá permitir que o usuário com permissão <code>ROLE_ADMINISTRADOR</code> autenticado, faça o <i>download</i> do boleto de cobrança gerado.	SCPREVI-CDU-MCSERV , P12. (Vide Tabela 1).
RF11	O sistema deverá permitir o registro de arquivo de boleto de cobrança no momento da geração do mesmo.	SCPREVI-CDU-MCSERV , P12. (Vide Tabela 1).
RF12	O sistema deverá permitir que o usuário com permissão <code>ROLE_ADMINISTRADOR</code> autenticado, consulte e visualize determinado boleto de servidor.	SCPREVI-CDU-MCSERV , A01, A02. (Vide Tabela 1).
RF13	O sistema deverá permitir que o usuário com permissão <code>ROLE_ADMINISTRADOR</code> autenticado, altere situação de boleto, exclua o boleto de servidor e retire o extrato de sacado do	SCPREVI-CDU-MCSERV , A03, A04, A05. (Vide Tabela 1).

	servidor.	
RF14	O sistema deverá permitir que o usuário com permissão <code>ROLE_ADMINISTRADOR</code> autenticado, gere, visualize e exclua o arquivo remessa de boleto de servidor.	SCPREVI-CDU-MCSERV , A06, A07, A08. (Vide Tabela 1).
RF15	O sistema deverá permitir que o usuário com permissão <code>ROLE_ADMINISTRADOR</code> autenticado, leia o arquivo retorno de boleto de servidor e informe se foi aceito ou não e, caso não aceito, informe o motivo.	SCPREVI-CDU-MCSERV , A09. (Vide Tabela 1).
RF16	O sistema deverá permitir que o usuário com permissão <code>ROLE_ADMINISTRADOR</code> autenticado, cadastre o arquivo retorno, caso aceito, no momento de sua leitura.	SCPREVI-CDU-MCSERV , A09.5. (Vide Tabela 1).
RF17	O sistema deverá permitir que o usuário com permissão <code>ROLE_ADMINISTRADOR</code> autenticado, cadastre, edite, visualize e exclua os dados de cedente.	SCPREVI-CDU-MCSERV , A10. (Vide Tabela 1).
RF18	O sistema deverá permitir que o usuário com permissão <code>ROLE_ADMINISTRADOR</code> autenticado, cadastre, edite, visualize e exclua os dados de conta bancária.	SCPREVI-CDU-MCSERV , A11. (Vide Tabela 1).
RF19	O sistema deverá permitir que o usuário com permissão <code>ROLE_ADMINISTRADOR</code> autenticado, cadastre, visualize, edite e exclua os dados de servidor exonerado.	SCPREVI-CDU-MSERVEXO , P01, A01, A02, A03.
RF20	O sistema deverá permitir que o usuário com permissão <code>ROLE_ADMINISTRADOR</code> autenticado, consulte servidores exonerados específicos e exporte a listagem em arquivos nos formatos: <i>“.PDF”</i> , <i>“.XLS”</i> , <i>“.XML”</i> e <i>“.CSV”</i> .	SCPREVI-CDU-MPROP , A04, A05.
RF21	O sistema deverá permitir que o usuário com permissão <code>ROLE_SERVIDOR</code> efetue o processo de autenticação no sistema.	SCPREVI-CDU-SERV , P01.
RF22	O sistema deverá permitir que o usuário com permissão <code>ROLE_SERVIDOR</code> autenticado, visualize seus dados e altere sua senha de acesso.	SCPREVI-CDU-SERV , A01, A02.
RF23	O sistema deverá permitir que o usuário com permissão <code>ROLE_SERVIDOR</code> autenticado, tire o extrato de todos os seus boletos e consulte-os por mês.	SCPREVI-CDU-SERV , A03.
RF24	O sistema deverá permitir que o usuário com permissão <code>ROLE_SERVIDOR</code> autenticado, exporte a listagem de seus boletos em arquivos nos formatos: <i>“.PDF”</i> , <i>“.XLS”</i> , <i>“.XML”</i> e <i>“.CSV”</i> .	SCPREVI-CDU-SERV , A04.

2.2.2 Requisitos Não Funcionais

Os requisitos não funcionais de *software* constituem e delimitam restrições aos requisitos funcionais do sistema. Basicamente, eles descrevem como as funcionalidades descritas nos requisitos funcionais devem ser executadas. Sommerville conceitua-os da seguinte forma:

São restrições sobre os serviços ou as funções oferecidas pelo sistema. Eles incluem restrições de *timing*, restrições sobre o processo de desenvolvimento e padrões. Os requisitos não funcionais aplicam-se frequentemente, ao sistema como um todo. Em geral eles não se aplicam às características ou serviços individuais de sistemas. (SOMMERVILLE, 2007, p. 80).

Dessa forma, para o SCPREVI, e, de acordo com os requisitos funcionais já descritos, foram desenvolvidos os requisitos não funcionais do sistema, segue abaixo:

Tabela 3: Requisitos Não Funcionais SCPREVI.

Requisito	Descrição
RNF01	O sistema deverá oferecer usabilidade com uma interface funcional e de simples entendimento do usuário.
RNF02	O sistema deverá oferecer confiabilidade, segurança dos dados transitados através de backups semanais do banco de dados utilizado e controle de acesso via autenticação.
RNF03	O sistema deverá oferecer alta performance com tempo mínimo de resposta de 2 segundos às operações propostas.
RNF04	O sistema deverá oferecer portabilidade, permitindo o funcionamento multiplataforma, podendo ser executado em vários sistemas operacionais e arquiteturas.

2.2.3 Regras de Negócio

O desenvolvimento deste projeto como já informado, está baseado nos artefatos do PU. Um desses artefatos chama-se **Regras de Negócio**, não necessariamente descritos somente no PU. Essas regras são baseadas em particularidades específicas do sistema, por isso são comumente chamadas de regras de domínio. Larman define as regras de negócio da seguinte maneira: “Regras de negócio descrevem tipicamente requisitos ou políticas que transcendem um projeto de *software* – elas são necessárias no domínio ou no negócio e muitas aplicações podem precisar respeitá-las”. Dessa forma, com base nessas e em outras informações, para o SCPREVI, estão designadas as seguintes regras de negócio de *software*:

Tabela 4: Regras de Negócio SCPREVI.

Regra	Descrição
RN01	Somente será permitida a criação de usuários para posterior acesso ao sistema, caso já exista uma instância de proprietário, servidor LIP ou servidor cedido cadastrada no sistema.
RN02	Os boletos de cobrança somente poderão ser gerados, caso já exista uma instância de cedente cadastrada e uma instância de conta bancária cadastrada e vinculada ao cedente.
RN03	Os boletos de cobrança somente poderão ser gerados, caso sejam preenchidos os campos obrigatórios, de acordo com o caso de uso SCPREVI-CDU-MCSERV , P11.
RN04	Somente será permitida a geração de arquivo remessa, caso já exista ao menos um boleto de cobrança de servidor registrado no sistema, pois o arquivo remessa buscará informações de um boleto que, por sua vez, está vinculado a um servidor.
RN05	Somente será efetuado o processo de registro de arquivo retorno, caso o banco tenha enviado um arquivo a aplicação e caso esse arquivo venha com o valor “A” no campo “Aceite”, o que significa que o arquivo remessa enviado ao banco estava nos padrões e não apresentou nenhuma irregularidade.
RN06	Somente será permitida a exclusão de título, caso este não esteja vinculado a nenhum arquivo remessa.
RN07	Somente será permitida a exclusão de servidor LIP ou cedido, caso este não esteja vinculado a nenhum título.
RN08	Cada título é vinculado a um único servidor, e cada servidor pode ter vários títulos vinculados.
RN09	Somente será permitida a geração de boletos no formato “.PDF”.
RN10	Cada título terá duas situações: “Pendente” e “Pago”.

2.2.4 Modelagem de Casos de Uso

De acordo com as regras do PU, dentro da disciplina **Requisitos** deve conter a modelagem de casos de uso, que é na verdade uma descrição de determinada funcionalidade do sistema, como Larman (2007, p. 90) afirma: “Casos de uso são documentos de texto e não diagramas. Portanto, a modelagem de casos de uso é essencialmente uma ação de redigir texto e não de desenhar diagramas”. Essa descrição é formada a partir de um conjunto de todos os passos que o usuário deve seguir até se chegar ao final da transação. Dessa forma, abaixo está descrita uma das principais modelagens de casos de uso do SCPREVI - Manter Cobrança de Servidor:

Tabela 1: Caso de Uso – Manter Cobrança de Servidor.

Caso de uso	SCPREVI-CDU-MCSERV - Manter Cobrança de Servidor.
Descrição	Este caso de uso tem como objetivo descrever como é feita

	a cobrança de um servidor (Servidor LIP ou Servidor Cedido) e como essa é mantida no sistema. As telas foram tiradas de uma instância de servidor LIP, porém servem de exemplo para servidor cedido. Este é um dos principais focos do sistema. Alguns dos processos aqui descritos serão: gerar boleto de servidor, gerar arquivo remessa de servidor e ler arquivo retorno. O processo de manter cobrança de servidor possibilitará o armazenamento de informações em seis classes: título, cedente, conta bancária, boleto, arquivo remessa e arquivo retorno.
Atores	Proprietário: responsável por manter a cobrança de servidor .
Pré-condição	- o O ator proprietário deve estar autenticado no sistema; - o Os seguintes dados de servidor, sacado, pessoa e endereço devem estar cadastrados: Nome, Sobrenome, CPF, <i>E-mail</i>, Logradouro, Número, Bairro, Cidade, Estado e CEP; - o Devem estar cadastrados um cedente e uma conta bancária;
Fluxo Principal	- o P01. O ator proprietário solicita o procedimento para manter a cobrança de servidor no sistema; - o P02. O sistema exibe a [Figura A.1], Anexo A, tela principal para o processo de autenticação no sistema; - o P03. [E01]. O ator proprietário informa <i>login</i> e senha e clica no botão “entrar”; - o P04. O sistema exibe a [Figura A.2], Anexo A, tela principal de usuário com perfil de proprietário (ROLE_ADMINISTRADOR) no sistema; - o P05. O ator proprietário entra na área de servidor clicando, no <i>menu</i> horizontal, na opção “Servidores LIP” ou “Servidores Cedidos”; - o P06. O sistema exibe a [Figura A.3], Anexo A, tela principal da área de servidores no sistema, que exibe uma listagem dos servidores já cadastrados e um <i>menu</i> vertical com outras opções; - o P07. O ator proprietário clica na opção “Atualização Monetária”, no <i>menu</i> vertical; - o P08. O sistema exibe a [Figura A.4], Anexo A, tela que exibe uma listagem dos servidores já cadastrados no sistema e uma ação disponível: “Atualização Monetária”; - o P09. O ator proprietário clica no ícone com uma imagem referente a um boleto; - o P10. O sistema exibe a [Figura A.5], Anexo A, que apresenta um formulário com os dados necessários para geração do boleto de cobrança para o servidor; - o P11. [E02, E03]. O ator proprietário preenche os

	campos referentes aos dados de título e boleto: *Nosso Número, *Dígito Nosso Número, *Número Documento, *Valor Original, Valor Desconto, Valor Dedução, *Valor Mora (%), Valor Acréscimo, *Data Documento, *Data Vencimento, Local Pagamento, *Mês, Instrução ao Sacado e Outras Instruções. Após o preenchimento dos dados, o ator proprietário clica no botão “Confirmar”; o P12. [E04]. O sistema registra o boleto de cobrança para o servidor deixando-o disponível para ser visualizado posteriormente pelo ator proprietário ou pelo próprio servidor. Neste momento o sistema oferece também a possibilidade de <i>download</i> do boleto, como o exemplo da [Figura A.6], Anexo A.
Fluxos Alternativos	- o A01. [P06]. Listar boletos de servidores: 1. O ator proprietário escolhe no <i>menu</i> vertical a opção “Listar Boletos” para consulta de todos os boletos dos servidores; 2. O sistema exibe a [Figura A.7], Anexo A, com uma listagem de todos os boletos dos servidores com as seguintes opções: a. Consulta: por servidor e por mês; b. Ações: “Visualizar” boleto no <i>browser</i>; “Alterar Situação” e “Excluir”. - o A02. [A01.2]. Visualizar boleto de servidor no <i>browser</i>: 1. O ator proprietário informa o nome do servidor ou o mês de referência da cobrança para consulta; 2. O sistema consulta automaticamente um ou mais boletos cadastrados que contenham os dados informados e os exibe na tela apresentando as ações disponíveis no passo A01.2b; 3. O ator proprietário escolhe a opção “Visualizar”; 4. O sistema exibe a [Figura A.8], Anexo A, com a visualização do boleto selecionado no <i>browser</i> do cliente, deixando também disponível para <i>download</i>; - o A03. [A02.2]. Alterar situação de título de servidor: 1. O ator proprietário escolhe a opção “Alterar Situação”; 2. O sistema executa uma atualização nos dados do título, altera a situação do mesmo e exibe o resultado no campo “Situação” da listagem. - o A04. [A02.2]. Excluir título de servidor: 1. O ator proprietário escolhe a opção “Excluir”; 2. [E05]. O sistema exibe a [Figura A.9], Anexo A,

	com uma janela solicitando a confirmação de exclusão. A janela apresenta as ações disponíveis: “Confirmar” e “Cancelar”; 3. [E11]. O ator proprietário confirma a exclusão do título escolhendo a opção “Confirmar”; 4. O sistema executa a operação excluindo o título selecionado; 5. O sistema exibe uma mensagem informando que a operação foi realizada com sucesso. o A05. [P06]. Extrato de sacado: 1. O ator proprietário escolhe no <i>menu</i> vertical a opção “Extrato de Sacado” para consulta de títulos por servidor; 2. O sistema exibe a [Figura A.10], Anexo A, apresentando uma listagem de todos os servidores cadastrados e uma opção para listar todos os boletos do servidor; 3. O ator proprietário escolhe a opção “Extrato de Sacado”; 4. O sistema exibe a [Figura A.11], Anexo A, com a listagem de todos os títulos do servidor selecionado e com as opções de exportação: a. Arquivo no formato “XLS” do <i>Microsoft Excel</i>; b. Arquivo no formato “PDF”; c. Arquivo no formato “CSV” do <i>Microsoft Excel</i>; d. Arquivo no formato “XML”. o A06. [P06]. Gerar arquivo remessa: 1. O ator proprietário escolhe no <i>menu</i> vertical a opção “Gerar Arquivo Remessa”; 2. O sistema exibe a [Figura A.12], Anexo A, com a listagem de todos os títulos dos servidores com as opções de consulta por sacado e por mês e com as ações “Visualizar” e “Gerar Arquivo Remessa”; 3. O ator proprietário escolhe a opção “Gerar Arquivo Remessa”; 4. O sistema executa a operação e gera um arquivo remessa para o título selecionado; 5. O sistema exibe a [Figura A.13], Anexo A, com uma listagem de todos os arquivos remessa já cadastrados e com as opções de visualização e exclusão dos mesmos. O sistema também exibe uma mensagem informando ao usuário que a operação foi realizada com sucesso; o A07. [A06.5]. Visualizar arquivo remessa: 1. O ator proprietário escolhe a opção “Visualizar” e o sistema exibe a [Figura A.14], Anexo A, abrindo
--	--

	o arquivo remessa selecionado no <i>browser</i> do cliente; o A08. [A06.5]. Excluir arquivo remessa: 1. [E06]. O ator proprietário escolhe a opção “Excluir” e o sistema exibe a [Figura A.15], Anexo A, com uma janela solicitando a confirmação de exclusão e informando ao usuário que, caso o processo seja confirmado, os arquivos retorno ligados a este arquivo remessa também serão excluídos. A janela apresenta as ações disponíveis: “Confirmar” e “Cancelar”; 2. O ator proprietário confirma a exclusão do arquivo remessa escolhendo a opção “Confirmar”; 3. O sistema executa a operação excluindo o arquivo remessa selecionado. Caso existam arquivos retorno ligados ao arquivo remessa; 4. O sistema exibe uma mensagem informando que a operação foi realizada com sucesso. o A09. [P06]. Ler arquivo retorno: 1. O ator proprietário escolhe no <i>menu</i> vertical a opção “Arquivo Retorno”; 2. O sistema exibe a [Figura A.16], Anexo A, com uma listagem de todos os arquivos remessa cadastrados e com as opções de consulta por sacado e por mês. 3. O ator proprietário informa o nome do sacado ou o mês de referência do arquivo remessa; 4. O ator proprietário escolhe a opção “Ler Arquivo Retorno”; 5. [E07, E08]. O sistema verifica a existência de um arquivo retorno enviado pelo banco para aquele arquivo remessa; 6. O sistema cadastra o arquivo retorno e exibe uma mensagem informando que o arquivo foi aceito. o A10. [P06]. Cadastrar novo cedente: 1. O ator proprietário escolhe no <i>menu</i> horizontal a opção “Cedentes”; 2. O sistema exibe a [Figura A.17], Anexo A, com uma listagem dos cedentes já cadastrados e com um <i>menu</i> vertical com outras opções; 3. O ator proprietário escolhe no <i>menu</i> vertical a opção “Cadastrar Novo Cedente”; 4. O sistema exibe a [Figura A.18], Anexo A, com um formulário de cadastro com os campos: *Nome Cedente e *CNPJ;
--	---

	5. O ator proprietário preenche os dados e clica no botão “Confirmar”; 6. [E09]. O sistema faz a validação das informações; 7. O sistema cadastra os dados do novo cedente e exibe uma mensagem informando que a operação foi realizada com sucesso. o A11. [A10.2]. Cadastrar nova conta bancária: 1. O ator proprietário escolhe no <i>menu</i> vertical a opção “Cadastrar Nova Conta”; 2. O sistema exibe a [Figura A.19], Anexo A, com um formulário de cadastro com os campos: *Cedente, *Número Conta, *Dígito Número Conta, *Número Agência, *Dígito Número Agência e *Banco; 3. O ator proprietário preenche os dados e clica no botão “Confirmar”; 4. [E10]. O sistema faz a validação das informações; 5. O sistema cadastra os dados da nova conta bancária e exibe uma mensagem informando que a operação foi realizada com sucesso.
Exceções	o E01. [P03]. Nome de usuário ou senha incorreto: • O sistema exibe uma mensagem informando que o usuário ou senha está incorreto ou, no caso de ausência de informação, o sistema informa que os campos são obrigatórios e continua no passo P03. o E02. [P11]. Ainda não existe um cedente ou conta bancária cadastrados no sistema: • O ator proprietário deve avançar ao passo A10 e ao passo A11 para proceder com o cadastro de um cedente e uma conta bancária respectivamente. o E03. [P11]. Erro de validação: campos obrigatórios não preenchidos: • O sistema exibe as mensagens informando que os campos marcados com asterisco em P11 são obrigatórios e retorna ao passo P10, como exemplo na [Figura A.20], Anexo A. o E04. [P12]. O ator proprietário escolhe a opção “Cancelar”: • O sistema fecha a janela de download do boleto e não salva o boleto na máquina do cliente e retorna ao passo P11. Entretanto, o boleto é registrado no servidor da aplicação, podendo ser aberto posteriormente no passo A02. o E05. [A04.2]. O ator proprietário escolhe a opção

	“Cancelar”: • O sistema fecha a janela de confirmação e retorna ao passo A04. o E06. [A08.1]. O ator proprietário escolhe a opção “Cancelar”: • O sistema fecha a janela de confirmação e retorna ao passo A08.1. o E07. [A09.5]. Arquivo retorno não existe: • Caso o banco ainda não tenha enviado o arquivo retorno para o título selecionado, o sistema exibe uma mensagem informando: “Ainda não existe um retorno do banco para este título”, conforme [Figura A.22], Anexo A, e retorna ao passo A09. o E08. [A09.5]. Arquivo retorno rejeitado pelo banco: • Caso o banco tenha rejeitado o título o sistema exibe uma mensagem informando que o título foi rejeitado e por qual motivo, conforme [Figura A.23], Anexo A. o E09. [A10.6]. Erro de validação: CNPJ inválido ou campos obrigatórios não preenchidos: • O sistema exibe as mensagens informando que o CNPJ é inválido ou que os campos marcados com asterisco em A10.4 são obrigatórios e retorna ao passo A10.4. o E10. [A11.4]. Erro de validação: campos obrigatórios não preenchidos: • O sistema exibe as mensagens informando que os campos marcados com asterisco em A11.2 são obrigatórios e retorna ao passo A11.2. o E11. [A04.3]. Existe um arquivo remessa vinculado ao boleto: • O sistema exibe uma mensagem informando que não é possível excluir um boleto que contenha um arquivo remessa vinculado e não executa a operação, conforme [Figura A.21], Anexo A.
Pós-condição	o Foi feita a atualização monetária de servidor, criando uma instância de título, boleto, arquivo remessa e arquivo retorno no sistema; o Essa instância foi atualizada ou excluída no sistema; o Uma instância de cedente e conta bancária foi criada, atualizada ou excluída do sistema; o Um boleto de cobrança foi gerado com a extensão “.pdf”; o Um arquivo remessa foi gerado com a extensão “.txt”.

2.2.5 Linguagem UML

A UML (*Unified Modeling Language*), que em português significa Linguagem de Modelagem Unificada, é uma linguagem que trata principalmente, da documentação da aplicação, o que é de total importância para um desenvolvimento de qualidade. Segundo Larman (2007, p. 32), a UML é apenas um padrão ou notação de diagramação, ou seja, é utilizada principalmente para a modelagem de dados de um sistema através de mecanismos gráficos, como o Diagrama de Classes de Domínio, ou também como o Diagrama de Sequência, por exemplo. Dessa forma, é uma das fases no desenvolvimento de *software* crucial para a criação e evolução de um Projeto Orientado a Objetos.

Em outras palavras, assim como os arquitetos criam plantas e projetos para serem usados por uma empresa de construção, os arquitetos de *software* criam diagramas UML para ajudar os desenvolvedores de *software* a construir o *software*. Se você entender o vocabulário da UML (os elementos visuais do diagrama e seus significados), pode facilmente entender e especificar um sistema e explicar o projeto daquele sistema para outros interessados. (PRESSMAN, 2011, p. 727).

A linguagem UML dispõe de vários tipos de diagramas para a modelagem de elementos de um sistema computacional. Dentre os mais importantes, e os que foram implementados para a documentação do SCPREVI e, conseqüentemente, auxiliaram no desenvolvimento, podemos citar:

- Diagrama de Casos de Uso;
- Diagrama de Classes de Domínio;
- Diagrama de Atividades;
- Diagrama de Sequência;

2.2.5.1 Diagrama de Casos de Uso

O diagrama de casos de uso é de total importância para o entendimento inicial do *software*. Entretanto, para se chegar ao diagrama em si, é necessário antes, desenvolver a descrição do caso de uso. Larman faz questão de frisar esta afirmação, dizendo:

Diagramas de casos de uso e relacionamentos entre casos de uso são secundários no trabalho de definição dos casos de uso. Casos de uso são documentos de texto. Fazer o trabalho de definição de casos de uso significa escrever texto. (LARMAN, 2007, p. 115).

Portanto, para se chegar ao produto final de um dos diagramas de casos de uso utilizados para o desenvolvimento do SCPREVI, foi escrita antes sua descrição em detalhe, vide **Tabela 1**. O diagrama pode ser observado na **Figura B.1**, Anexo B, que descreve as principais funções dos atores do sistema.

2.2.5.2 Diagrama de Atividades

No PU, existe uma disciplina conhecida como **Modelagens de Negócio**, que apresenta como um de seus artefatos, o **Modelo de Objetos de Negócios**. Neste artefato está descrito o diagrama de atividades da UML. O diagrama de atividades complementa o diagrama de casos de uso, fornecendo uma apresentação gráfica, dinâmica e sequencial do fluxo de interação em determinado cenário do sistema.

Um diagrama de atividades da UML oferece uma notação rica para mostrar uma sequência de atividades, inclusive atividades paralelas. Ele pode ser aplicado em qualquer perspectiva ou propósito, mas é mais popular para visualizar fluxos de trabalho e processos de negócios, além de casos de uso. (LARMAN, 2007, p. 484).

Dessa forma, observa-se na **Figura B.2**, Anexo B, o diagrama de atividades intitulado que complementa o caso de uso **SCPREVI-CDU-MCSERV**, vide **Tabela 1**. Em determinados pontos do diagrama, são citados alguns passos do caso de uso descrito na **Tabela 1**, e também de um caso de uso intitulado **SCPREVI-CDU-MSERV**, o qual não está presente neste texto por questões óbvias. Isto justifica a afirmação de que este diagrama complementa o caso de uso.

Nota-se neste diagrama, a divisão por papéis, ou seja, a cada ator é designada determinada tarefa. No caso, o ator *Proprietário* e o ator *Caixa Econômica*. Assim, também pode ser chamado de Diagrama de Raias. Entretanto, é importante frisar que este diagrama apenas ilustra o funcionamento geral de uma cobrança registrada. Ou seja, a parte de comunicação com o banco não foi implementada no sistema, pois será uma fase futura deste projeto.

2.2.5.3 Diagrama de Classes de Domínio

Na modelagem orientada a objetos proposta no PU, está descrito o diagrama de classes de domínio que a UML fornece para, segundo Pressman (2011, p. 727), “modelar classes incluindo seus atributos, operações, relações e associações com

outras classes". O diagrama de classes apresenta uma visão estática e até estrutural dos objetos do sistema a ser desenvolvido.

Dessa forma, entende-se que o diagrama de classes auxilia, sobretudo no desenvolvimento da camada de acesso aos dados do sistema. Ou seja, como o próprio nome do diagrama sugere, cada componente do diagrama equivale não somente a uma classe do sistema, como também a uma tabela do banco de dados do sistema. Assim, para o SCPREVI, foi desenvolvido no produto final, o diagrama de classes que pode ser observado na **Figura B.3**, Anexo B.

2.2.5.4 Diagrama de Sequência

O diagrama de sequência é mais uma funcionalidade oferecida pela UML que permite ao analista ou arquiteto, desenvolver um modelo comportamental, indicando determinadas funções de eventos. Dessa forma, vê-se que este diagrama, assim como o diagrama de atividades, complementa a descrição de casos de uso. Entretanto, o diagrama de atividades apresenta uma interação de determinado cenário do sistema. Já o diagrama de sequência, mostra mais profundamente como são realizados determinados eventos e processos no sistema. Larman explica esta interligação da seguinte forma:

Os casos de uso descrevem como os atores externos interagem com o sistema de *software* que estamos interessados em criar. Durante essa interação, um ator gera **eventos de sistema** para um sistema, geralmente solicitando alguma **operação de sistema** para tratar o evento. (LARMAN, 2007, p. 197).

Ou seja, o que está descrito no caso de uso é tratado de forma concreta e explícita no diagrama de sequência. Dessa forma, para o SCPREVI foram desenvolvidos durante as iterações, os diagramas de sequência descritos na **Figura B.4** e **Figura B.5**, Anexo B. Assim, é possível notar a partir das figuras apresentadas, que um diagrama complementa o outro. Ou seja, o diagrama descrito na **Figura B.5** inicia a partir do ponto final do diagrama descrito na **Figura B.4**. Isto é feito para garantir flexibilidade e produtividade ao desenvolvimento do sistema, tornando mais fácil o entendimento por parte dos desenvolvedores.

2.2.6 Glossário

O glossário, também descrito no PU, é uma forma de auxiliar no desenvolvimento do sistema não só para os desenvolvedores, como também para os analistas. Isto porque o glossário deve ser acessado a todo momento durante as iterações do PU, portanto desde suas primeiras iterações já deve estar sendo desenvolvido.

Em sua forma mais simples, o glossário define termos importantes. Também abrange o conceito de **dicionário de dados**, que registra requisitos relativos aos dados, tais como: regras de validação, valores aceitáveis, etc. O glossário pode detalhar qualquer elemento: um atributo de um objeto, um parâmetro de chamada de uma operação, um *layout* de relatório, etc. (LARMAN, 2007, p. 84).

Portanto, segue abaixo o glossário utilizado no desenvolvimento do SCPREVI, que descreve alguns dos termos mais importantes para entendimento:

Tabela 5: Glossário SCPREVI.

Código	Termo	Descrição e Informações	Formato
TG01	Cobrança Registrada	Neste tipo de cobrança, uma comunicação com o banco do cedente é necessária. Esse tipo de cobrança tem como características o registro de informações dos boletos dos clientes no banco do cedente. Em contraposição, o tipo de cobrança por boletos não registrados, não apresenta a necessidade de comunicação com o banco e registro de informações dos boletos de clientes no banco. Logo, o banco só terá informações do boleto após o pagamento pelo sacado, seja este efetuado após a data de vencimento ou não. Dessa forma, entende-se que a cobrança com boletos registrados garante a segurança da transação	Bloqueto de cobrança no padrão exigido pela Febraban.

		para o cedente, pois o banco terá todas as informações do título, antes do pagamento do mesmo por parte do sacado. Assim, é possível inclusive, o protesto, caso o sacado atrase o pagamento.	
TG02	Cedente	O cedente, também denominado cliente, é a entidade a qual requisita comunicação com o banco a fim de realizar o registro da transação. No âmbito deste projeto o banco do cedente é a Caixa Econômica Federal.	
TG03	Conta Bancária	Conta criada no banco do cedente, necessária para o processo de cobrança. É nesta conta, onde serão depositados os valores referentes a pagamentos dos títulos por parte dos sacados.	
TG04	Servidor LIP	Servidor devidamente efetivo no município – no âmbito deste projeto, servidor efetivo do município de Palmas - TO - que decide na forma da lei, entrar de Licença para tratar de Interesses Particulares.	
TG05	Servidor Cedido	Servidor devidamente efetivo no município – no âmbito deste projeto, servidor efetivo do município de Palmas - TO – que durante exercício de cargo, é invocado informalmente a exercer profissão em outro órgão ou entidade da administração direta ou indireta da União, dos Estados, do Distrito Federal e de Municípios.	
TG06	Proprietário	Principal usuário do	

		SCPRESVI. Este termo é utilizado, sobretudo para especificação de documentação do projeto. Dessa forma, o proprietário executa os principais e mais importantes procedimentos do sistema.	
TG07	Arquivo Remessa	Arquivo gerado pelo SCPRESVI com o objetivo de comunicar-se com a aplicação do banco Caixa Econômica Federal, a partir de dados do sacado e do título em questão. É através deste arquivo que o banco obtém as informações necessárias para registro do título em seu banco de dados.	Padrão CNAB-240-SIGCB.
TG08	Arquivo Retorno	Arquivo gerado pela aplicação do banco Caixa Econômica Federal, a partir dos dados lidos no arquivo remessa enviado pelo cliente. O arquivo retorno tem o objetivo de informar a aplicação do cliente se o título foi aceito, e caso não aceito, quais os motivos de rejeição.	Padrão CNAB-240-SIGCB.
TG09	Atualização Monetária	Processo pelo qual se aplicam multas como moras, e juros em determinada dívida, que por algum motivo não foi paga em seu valor original até a data de vencimento. No âmbito deste projeto, será aplicada de acordo com o vencimento de contribuições de servidores ao PREVIPALMAS, que será conferida sempre aos dias quinze (15) de cada mês subsequente ao da competência, de acordo	

		com a lei nº 1.414/2005, art. 86. Nesse caso, somente são gerados os boletos de cobrança, se os servidores em questão atrasem suas contribuições, aplicando-se então o disposto nessa descrição de atualização monetária: multa de 2% abatidos em cima do valor original da dívida; juros de 0.5% am (ao mês).	
TG10	Título/Boleto/Bloqueto	No âmbito deste projeto são equivalentes. Culminam em um documento utilizado para fazer cobrança de dívidas. O pagamento desta dívida a partir deste tipo de arquivo pode ser efetuado através de agências bancárias, casas lotéricas e outros locais devidamente autorizados.	Bloqueto de cobrança no padrão exigido pela FEBRABAN.
TG11	Salário de Contribuição Previdenciária	Valor referente a contribuições de servidores ao RPPS ou ao RGPS, estando o servidor devidamente efetivo a um órgão ou entidade da administração direta ou indireta da União, dos Estados, do Distrito Federal e de Municípios, com o principal objetivo de gozar de benefícios oferecidos pelas entidades ligadas aos regimes.	
TG12	Sacado	Cliente da empresa responsável pela cobrança. No caso, no SCPREVI, um sacado é um servidor devedor de uma contribuição atrasada, servidor LIP ou cedido.	

3 APORTE TEÓRICO

Este tópico apresenta como principal objetivo demonstrar em detalhe quais ferramentas, linguagens de programação, *frameworks* de desenvolvimento, padrões de projeto e demais técnicas foram utilizadas para a codificação e programação do SCPREVI.

3.1 Base de Dados

Segundo Deitel (2005, p. 895), um banco de dados é uma “coleção organizada de dados” e existem várias formas para acessar e manipular essas informações. Deitel afirma ainda que uma dessas técnicas chama-se Sistema de Gerenciamento de Bancos de Dados (SGBD) ⁸, que “fornece mecanismos para armazenar, organizar, recuperar, e modificar dados para muitos usuários”.

Para o desenvolvimento do SCPREVI, adotou-se um SGBD relacional (RDBMS) ⁹, denominado *MySQL*, primeiramente por ser um SGBD de código aberto (*Open Source*), disponível tanto para *Windows* como para *Linux* e por ser indicado para projetos não muito longos.

O *MySQL* é o banco de dados de código-fonte aberto mais popular do mundo, tendo mais de 70 milhões de instalações no mundo todo. O fato de ser de código-fonte aberto não significa que ele seja gratuito: isso dependerá de como você utilizará o banco de dados. Para que o *MySQL* possa ser utilizado gratuitamente, o aplicativo que o utiliza tem que seguir a licença GPL ¹⁰. (LUCKOW e MELO, 2010, p. 63).

Qualquer banco de dados que disponha de uma base de dados relacional utiliza uma linguagem para manipulação de suas informações. Esta linguagem chama-se SQL (*Structured Query Language*) ou Linguagem de Consulta Estruturada. Entretanto, neste projeto adotou-se um *framework* para tratamento da camada de acesso aos dados da aplicação, o *Hibernate*. E este *framework* já apresenta sua própria linguagem de manipulação e consulta de dados, baseada na SQL, a HQL (*Hibernate Query Language*).

A diferença em relação à SQL, é que a HQL é totalmente orientada a objetos, com o intuito de desvincular o desenvolvedor da programação de bancos de dados, dependendo de sua aplicação aumentando assim, a produtividade no

⁸ Do Inglês, *Database Management System (DBMS)*.

⁹ Do Inglês, *Relational Database Management Systems*.

¹⁰ Do Inglês, *General Public License* ou Licença Pública Geral.

desenvolvimento. Isto não quer dizer que a SQL não será utilizada no projeto, pois a HQL utiliza “por traz” de toda sua programação orientada a objetos, comandos em SQL puro, que inclusive podem ser avistados durante a execução da aplicação, dependendo da ferramenta de desenvolvimento utilizada.

3.1.1 MySQL Workbench

O *MySQL Workbench* é uma ferramenta de tecnologia CASE¹¹ (Análise e Engenharia de Sistemas Apoiada e Assistida por Computador) desenvolvida pela MySQL, multiplataforma, e dispõe de funcionalidades de design visual de um banco de dados. A ferramenta *Workbench* foi utilizada neste projeto para a interação com o SGBD, sobretudo no que diz respeito ao desenvolvimento do MER (Modelo de Entidades e Relacionamentos). Posteriormente através do MER, foi possível gerar o *script* do banco e rodá-lo através da própria ferramenta, no SGBD fisicamente, já com todas as amarrações e dependências das tabelas e futuramente, utilizar outra ferramenta para com suas funcionalidades gerar as classes de domínio na aplicação de acordo com as tabelas do banco (este procedimento será descrito em tópicos posteriores).

Isto garante de certa forma a segurança do sistema, pois a integridade do banco de dados não será afetada, uma vez que a aplicação seguirá as regras já executadas. Dessa forma, para auxiliar no desenvolvimento do SCPREVI, implementou-se o DER (Diagrama Entidade Relacionamento) do projeto através dessa ferramenta e configurou-se todas suas particularidades, podendo ser observado na **Figura 7**, logo abaixo:

¹¹ Do Inglês, *Computer Assisted Aided Software Engineering*.

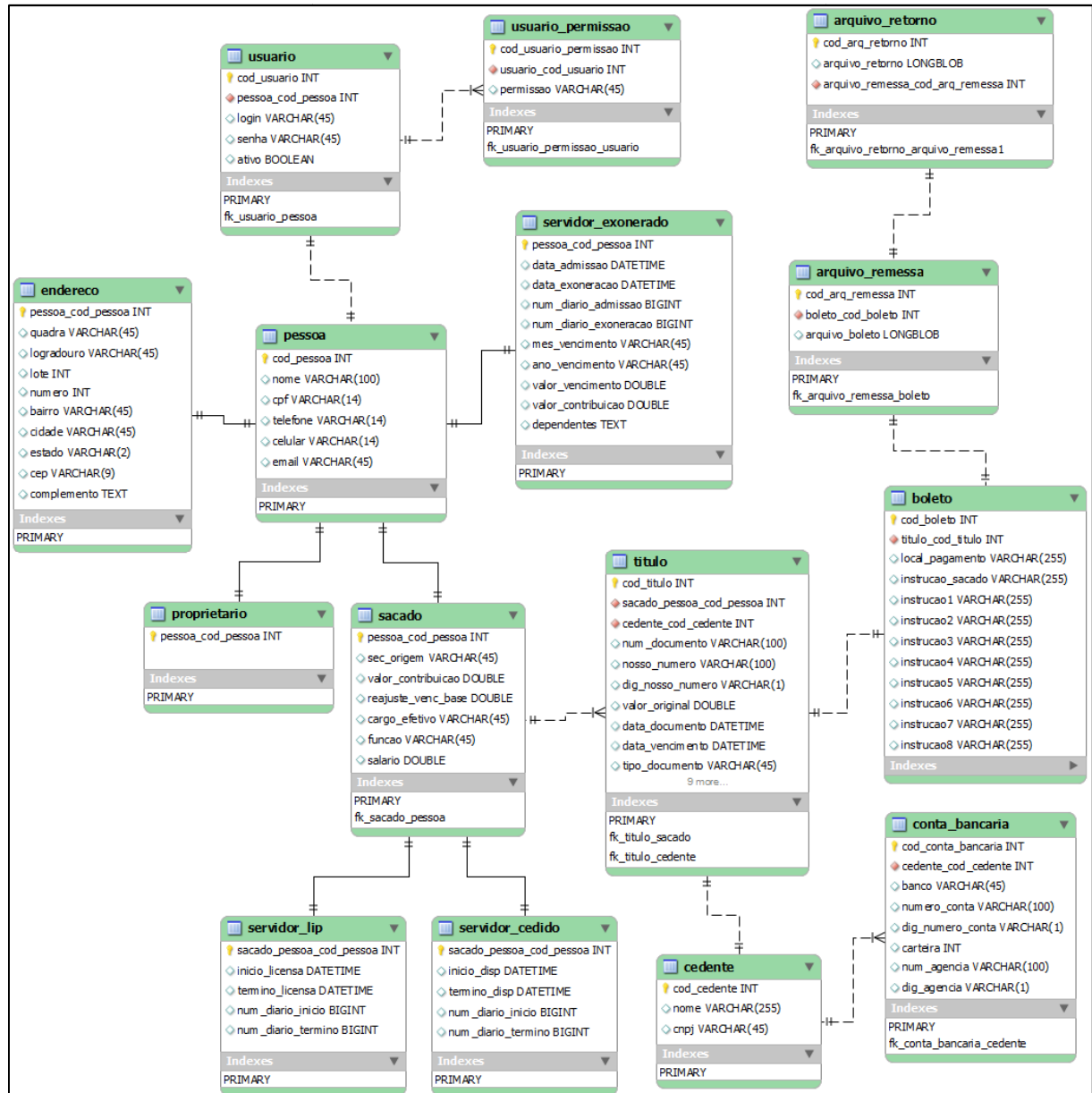


Figura 2 - Diagrama Entidade Relacionamento – SCPREVI.

3.2 Linguagem Java

Java é uma linguagem de programação criada pela empresa **Sun Microsystems** com o objetivo principal de promover soluções para a plataforma *web* e devido suas características e imensa quantidade de funcionalidades oferecidas, se tornou uma ferramenta padrão para o desenvolvimento de soluções corporativas. É uma linguagem fortemente orientada a objetos, que por apresentar segurança, versatilidade, portabilidade e eficiência total aos sistemas, garante maior performance aos mesmos e este é um dos motivos pelo qual ela foi escolhida para o desenvolvimento do projeto aqui ilustrado.

A plataforma de programação *Java* é formada de uma linguagem de mesmo nome e de uma extensa biblioteca de classes 100% orientada a objetos, independente de plataforma, segura e de ótima performance. (JANDL, 2008, p. 4).

A plataforma *Java* apresenta atualmente três subdivisões:

- *Java ME (Java Micro Edition)* – destinada a dispositivos móveis;
- *Java SE (Java Standard Edition)* – dispõe de todas as funcionalidades padrão do *Java*;
- *Java EE (Java Enterprise Edition)* – destinada a aplicações corporativas.

A **Figura 8** logo abaixo ilustra de forma clara o funcionamento do ambiente *Java*:

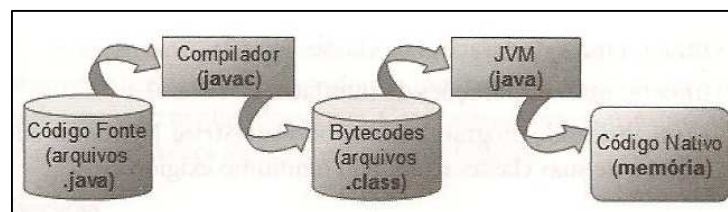


Figura 3 - Ambiente *Java*.
Fonte: Jandl, 2008, p. 5.

Ou seja, os arquivos com extensão “*.java*”, códigos criados por programadores, passam por um processo conhecido como **compilação**. Isto é possível através de uma ferramenta disponível internamente no *Java* denominado compilador *javac*. Dessa forma, o compilador transforma o código em *bytecodes*, que são armazenados em arquivos com a extensão “*.class*” legíveis por outra ferramenta indispensável no ambiente *Java*, a JVM (*Java Virtual Machine* - Máquina Virtual *Java*), que é responsável por interpretar os *bytecodes* gerados e integrá-los ao sistema operacional como código nativo.

A API¹² *Java* dispõe de duas formas para configuração:

- *JRE (Java Runtime Environment)* – apresenta um ambiente mínimo para funcionamento de determinadas aplicações, sobretudo com tecnologia *web*, como o acesso ao site do Banco do Brasil, por exemplo. Basicamente é o *Java* para clientes;

¹² Do Inglês *Application Programming Interface* ou Interface de Programação de Aplicativos.

- *JDK (JavaSE Development Kit)* – ferramenta completa destinada e utilizada sobretudo por desenvolvedores. É o ambiente de desenvolvimento *Java* considerado padrão e inclui um conjunto útil de ferramentas de desenvolvimento, além do próprio JRE.

A versão do *Java* utilizada para o desenvolvimento do SCPREVI foi a **1.7.0_03**. Esta é a versão atual do JDK oferecido pela *Sun*, que é atualmente parte inclusa à empresa *Oracle*.

3.2.1 Framework de Persistência *Hibernate*

Praticamente qualquer solução que disponha de um sistema de informação é composta por um ORM (*Object Relational Mapping*, ou Mapeamento Objeto-Relacional), ou seja, automaticamente deve persistir e consistir alguma informação em um banco de dados relacional. Dessa forma, para realizar este processo, existem soluções desenvolvidas sob a plataforma *Java* que nasceram com a idéia de auxiliar o desenvolvedor. Uma dessas soluções chama-se *Hibernate*, um *framework* que possibilita ao programador: portabilidade, segurança e alta produtividade. Assim, por se tratar de uma ferramenta ORM, deve seguir e, sobretudo, implementar uma especificação.

Dessa forma, o *Hibernate* implementa a especificação JPA (*Java Persistence API*), que define um padrão de persistência para permitir que desenvolvedores *Java* gerenciem dados que são armazenados de forma relacional, e está inclusa no documento *JSR: 220 Enterprise JavaBeans 3.0* – especificação da arquitetura EJB (*Enterprise JavaBeans*) para desenvolvimento de aplicações baseadas em componentes.

O *Hibernate* é apenas uma das soluções ORM encontradas hoje no mercado para a linguagem *Java*, embora seja a mais utilizada. Existem outras, como o *TopLink*, da *Oracle*, e *OpenJPA*, da *Apache*. Independentemente disso e seja qual for a solução que venha a ser adotada pelo desenvolvedor em seus projetos, todas tendem a seguir a especificação JPA, que integra uma especificação maior, a EJB 3.0, responsável por padronizar o modelo de programação EJB. (LUCKOW e MELO, 2010, p. 122).



Figura 4 - Arquitetura do *Hibernate*.

Fonte: Luckow e Melo, 2010, p. 123.

Acima, observa-se uma imagem que ilustra de forma clara a arquitetura do *Hibernate*. A versão do *Hibernate* utilizada no SCPREVI foi a **3.2.5**. Logo, desenvolveu-se a partir dos benefícios oferecidos por essa ferramenta a camada de persistência de dados do projeto SCPREVI, tornando o desenvolvimento mais produtivo e garantindo segurança à aplicação. Aliada a isto, a principal configuração do *Hibernate* para que se torne possível o seu funcionamento, é a conexão com o banco de dados. Para isto, é necessário realizar a parametrização de um arquivo denominado "*hibernate.cfg.xml*". É a partir deste arquivo que o *framework* realiza os procedimentos configurados no mapeamento. Dessa forma, segue abaixo a configuração utilizada para acesso ao banco de dados do SCPREVI:

```

3 </hibernate-configuration>
4 <session-factory>
5     <!-- Dialete do hibernate para o MySQL -->
6     <property name="hibernate.dialect">org.hibernate.dialect.MySQL5InnoDBDialect</property>
7     <!-- Aponta para o dataSource JNDI -->
8     <property name="hibernate.connection.datasource">java:/comp/env/jdbc/SCPREVI_Resource</property>
9     <property name="hibernate.current_session_context_class">thread</property>
10    <!-- debug -->
11    <property name="hibernate.show_sql">true</property>
12    <property name="hibernate.format_sql">true</property>
13    <property name="hibernate.generate_statistics">true</property>
14    <property name="hibernate.use_sql_comments">true</property>
15    <property name="hibernate.hbm2ddl.auto">update</property>
16    <!-- Classes de domínio -->
17    <mapping class="scprevi.model.dominio.ArquivoRemessa_" />
18    <mapping class="scprevi.model.dominio.ArquivoRetorno_" />
19    <mapping class="scprevi.model.dominio.Boleto_" />
20    <mapping class="scprevi.model.dominio.Cedente_" />
21    <mapping class="scprevi.model.dominio.ContaBancaria_" />
22    <mapping class="scprevi.model.dominio.Endereco_" />
23    <mapping class="scprevi.model.dominio.Pessoa_" />
24    <mapping class="scprevi.model.dominio.Proprietario_" />
25    <mapping class="scprevi.model.dominio.Sacado_" />
26    <mapping class="scprevi.model.dominio.ServidorCedido_" />
27    <mapping class="scprevi.model.dominio.ServidorExonerado_" />
28    <mapping class="scprevi.model.dominio.ServidorLip_" />
29    <mapping class="scprevi.model.dominio.Titulo_" />
30    <mapping class="scprevi.model.dominio.Usuario_" />
31    <mapping class="scprevi.model.dominio.UsuarioPermissao_" />
32 </session-factory>
33 </hibernate-configuration>

```

Figura 5 - Arquivo de configuração do *Hibernate* do SCPREVI.

Ao observar a imagem, percebe-se que não existem as informações do banco de dados como, por exemplo, nome de usuário e senha. Entretanto, observa-se também, na linha oito a propriedade “*connection.datasource*” do *hibernate*, com o valor “*java:/comp/env/jdbc/SCPREVI_Resource*”. Isto significa que está sendo utilizada uma conexão por meio de um *Data Source*¹³ JNDI (*Java Naming and Directory Interface* ou Espaço de Nomes e Objetos Java), apontando para o *Data Source* intitulado “*SCPREVI_Resource*”. Este, por sua vez, está configurado em outro arquivo. Este arquivo chama-se “*context.xml*” e segue na imagem abaixo:

¹³ “A configuração da conexão por meio de um *Data Source* permite, além da conexão, configurar um *pool* de conexão. Este manterá várias conexões abertas simultaneamente, tornando mais rápido o acesso ao banco de dados para aqueles que precisarem de uma conexão”. (LUCKOW e MELO, 2010, p. 236).

```

1  <?xml version="1.0" encoding="iso-8859-1"?>
2  <Context path="/SCPREVI_1.3" reloadable="true">
3    <Resource
4      name="jdbc/SCPREVI_Resource"
5      auth="Container"
6      driverClassName="com.mysql.jdbc.Driver"
7      logAbandoned="true" maxActive="20"
8      maxIdle="10" maxWait="15000"
9      password="123456"
10     removeAbandoned="true"
11     removeAbandonedTimeout="30"
12     type="javax.sql.DataSource"
13     url="jdbc:mysql://localhost:3306/scprevi"
14     username="root"
15     validationQuery="select current_date" />
16  </Context>

```

Figura 6 - Arquivo de configuração do *Data Source* de conexão do SCPREVI.

Neste arquivo é feita toda a parametrização para que seja possível a conexão com o banco de dados. Perceba que o atributo “*name*” na linha quatro, apresenta o mesmo valor citado no arquivo anterior, fazendo dessa forma, a interligação. Dentre os atributos mais importantes deste arquivo estão os referentes à configuração de *pool* de conexão:

- *maxActive* (linha sete) – permite configurar o número máximo de conexões em uso;
- *maxIdle* (linha oito) – permite configurar o número máximo de conexões aguardando para serem utilizadas;
- *maxWait* (linha oito) – permite configurar o tempo máximo de espera por uma conexão disponível.

Portanto, esta configuração facilita, sobretudo, a integração com outros *frameworks* Java que também necessitam executar conexão com o mesmo banco de dados. Assim, é importante não esquecer-se de avisar ao servidor *web* – neste caso utilizou-se o *Apache Tomcat 7.0.22*, que será explanado em tópicos posteriores – de que a aplicação atual está utilizando essa conexão via *Data Source JNDI*. Isso pode ser configurado em um arquivo chamado “*web.xml*”, que se encontra dentro do diretório “WEB-INF” da aplicação. Logo, para o SCPREVI foi configurado este parâmetro no “*web.xml*” da seguinte forma:

```

24      <!-- Data Source JNDI -->
25      <!-- Avisa o servidor web sobre existência do Data Source -->
26      <resource-ref>
27          <description>DataSource SCPREVI_Resource</description>
28          <res-ref-name>jdbc/SCPREVI_Resource</res-ref-name>
29          <res-type>javax.sql.DataSource</res-type>
30          <res-auth>Container</res-auth>
31      </resource-ref>

```

Figura 7 - Configuração do Data Source no arquivo “web.xml” do SCPREVI.

3.2.2 Padrão de Projeto DAO

Para organizar o código fonte do projeto e consequentemente, garantir produtividade adotou-se para o desenvolvimento do SCPREVI o padrão de Projeto ou *Design Pattern* DAO (*Data Access Object*) que em tradução livre significa Acesso aos Dados por Objetos. Ou seja, como a própria descrição informa, o DAO realiza as transações necessárias através do paradigma de orientação a objetos. Segundo Luckow e Melo (2010, p. 177), “essa abordagem de separar a responsabilidade do acesso aos dados em classes específicas é orientado pelo padrão de projeto DAO, que fornece uma maneira transparente para as classes de regra de negócio acessarem as fontes de dados do sistema”.

Dessa forma, a organização do projeto fica em alta não só na fase de desenvolvimento, mas, principalmente, na manutenção do sistema, pois cada classe apresenta uma responsabilidade, tornando-se assim, uma regra na utilização desse padrão de projeto. Assim, para tornar o desenvolvimento ainda mais ajustado, adotou-se também uma técnica do *Java* conhecida como classes genéricas (*generics*).

As classes genéricas permitem prover funcionalidades comuns que podem ser empregadas com diferentes tipos de dados, descrevendo conceitos de modo independente dos tipos envolvidos (tal como as idéias de lista ou pilha se aplicam a diferentes tipos de dados). As classes genéricas podem ser utilizadas através da herança para construção de novas classes, genéricas ou não. (JANDL, 2008, p. 59).

Portanto, desenvolveu-se então a partir dessas duas filosofias, uma classe denominada “*HibernateDAO<T>*” que implementa uma *interface* com nome “*InterfaceHibernateDAO<T>*”. Ambas genéricas, ou seja, os seus métodos podem servir para qualquer entidade do sistema, bastando informá-la no momento em que instanciá-la. Uma *interface* é de total importância para uma boa codificação, pois, segundo Jandl (2008, p. 48), ela atua como um “mecanismo que define um conjunto de operações, sem necessidade de implementação, estabelecendo um modelo de

comportamento para outras classes”. Assim aplica-se perfeitamente à idéia do projeto. Segue abaixo as duas classes aqui citadas que foram desenvolvidas para o SCPREVI:

```

1  package scprevi.model.util;
2  import java.util.List;
3
4  public interface InterfaceHibernateDAO<T> {
5
6      public void salvar(T entidade);
7      public void atualizar(T entidade);
8      public void excluir(T entidade);
9      public T carregar(Integer codigo);
10     public List<T> listar();
11
12 }

```

Figura 8 - Classe “InterfaceHibernateDAO.java” do SCPREVI.

```

5  public class HibernateDAO<T> implements InterfaceHibernateDAO<T> {
6
7      private Session sessao;
8      private Class<T> classe;
9
10     // construtor passando a classe como parâmetro
11     public HibernateDAO(Class<T> classe) {
12         this.classe = classe;
13     }
14     // seta uma sessão
15     public void setSession(Session session){
16         this.sessao = session;
17     }
18     // métodos CRUD
19     @Override
20     public void salvar(T entidade) {
21         this.sessao.save(entidade);
22     }
23     @Override
24     public void atualizar(T entidade) {
25         this.sessao.update(entidade);
26     }
27     @Override
28     public void excluir(T entidade) {
29         this.sessao.delete(entidade);
30     }
31     @Override
32     public T carregar(Integer codigo) {
33         return (T) this.sessao.get(classe, codigo);
34     }
35     @Override
36     public List<T> listar() {
37         return this.sessao.createCriteria(classe).list();
38     }
39 }

```

Figura 9 - Classe “HibernateDAO.java” do SCPREVI.

Arelada a isto, há uma funcionalidade no Java que acompanha todo o processamento durante todas as transações na camada de acesso aos dados do sistema, conhecida como POJO, do Inglês *Plain and Old Java Object*, ou, em tradução livre “Bom e Velho Objeto Java”. Trata-se de uma classe que sempre será utilizada no mapeamento objeto/relacional com o banco de dados, através do *Hibernate*. Segundo Luckow e Melo (2010, p. 180), “essa classe deve seguir o padrão *JavaBean*¹⁴ que determina que a classe deve ter propriedades privadas (*private*) e os métodos “*get/set*” correspondentes. Não é uma boa prática colocar qualquer outro tipo de método dentro dessa classe”.

Seguindo esta linha de raciocínio, é necessário que exista uma maneira de conseguir uma fábrica de sessão do *Hibernate* conhecida também como *SessionFactory*. Essa funcionalidade permite ao *Hibernate* proceder com a leitura de todas as parametrizações configuradas no arquivo “*hibernate.cfg.xml*” citado anteriormente, vide **Figura 10**. Ou seja, assim como está descrito na documentação do *Hibernate*, essa funcionalidade é garantida através da classe “*HibernateUtil*”, que “toma conta do *startup* e faz o acesso a uma *SessionFactory* de maneira conveniente”. Assim, segue abaixo uma ilustração da classe “*HibernateUtil*” desenvolvida para o SCPREVI:

```

6   public class HibernateUtil {
7
8       private static final SessionFactory sessionFactory = buildSessionFactory();
9       private static SessionFactory buildSessionFactory() {
10          try {
11              AnnotationConfiguration cfg = new AnnotationConfiguration();
12              cfg.configure("scprevi/bd/config/hibernate.cfg.xml");
13              return cfg.buildSessionFactory();
14          } catch (Throwable ex) {
15              System.out.println("Criacao inicial do objeto "
16                  + "SessionFactory falhou! " + ex.getMessage());
17              throw new ExceptionInInitializerError(ex);
18          }
19      }
20
21      public static SessionFactory getSessionFactory() {
22          return sessionFactory;
23      }
24  }

```

Figura 10 - Classe “*HibernateUtil.java*” do SCPREVI.

Juntamente a isso criou-se também, com o objetivo de construir as classes DAO’s utilizadas na aplicação, uma classe denominada “*DAOFactory.java*”. Esta classe representa o único ponto em toda a aplicação onde a classe DAO genérica e

¹⁴ Também conhecido como *Entity Bean*, faz parte da arquitetura do EJB (*Enterprise Java Beans*).

outras específicas são instanciadas. É através dessa classe que se obtém a instância da classe “*HibernateDAO<T>*” independente de entidade e atribui-se a ela, a sessão carregada a partir da classe “*HibernateUtil*”. Dessa forma segue abaixo uma imagem que demonstra somente alguns métodos da classe “*DAOFactory*” utilizada no SCPREVI, pois ela deve implementar todos os métodos referentes a todas as entidades do sistema que necessitam comunicar-se com o banco de dados:

```

5 public class DAOFactory {
6
7     public static InterfaceHibernateDAO<Boleto_> criarBoletoDAO() {
8         HibernateDAO<Boleto_> dao = new HibernateDAO<Boleto_>(Boleto_.class);
9         dao.setSession(HibernateUtil.getSessionFactory().getCurrentSession());
10        return dao;
11    }
12
13    public static InterfaceHibernateDAO<Endereco_> criarEnderecoDAO() {
14        HibernateDAO<Endereco_> dao = new HibernateDAO<Endereco_>(Endereco_.class);
15        dao.setSession(HibernateUtil.getSessionFactory().getCurrentSession());
16        return dao;
17    }
18
19    public static InterfaceHibernateDAO<Pessoa_> criarPessoaDAO() {
20        HibernateDAO<Pessoa_> dao = new HibernateDAO<Pessoa_>(Pessoa_.class);
21        dao.setSession(HibernateUtil.getSessionFactory().getCurrentSession());
22        return dao;
23    }

```

Figura 11 – Parte da classe “*DAOFactory.java*” do SCPREVI.

Portanto, entende-se que o padrão de projeto DAO possibilita ao desenvolvedor organizar-se conforme suas necessidades. Dessa forma, segue abaixo uma imagem que resume de forma clara a separação entre camadas utilizada neste projeto:

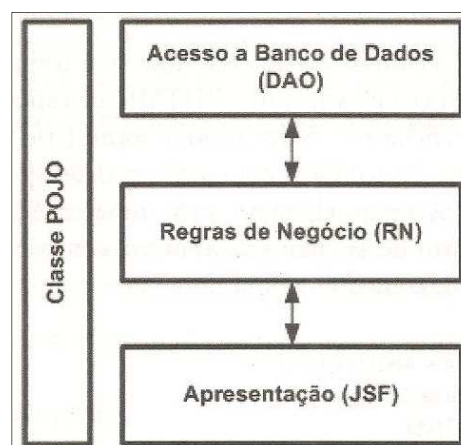


Figura 12 - Representação da arquitetura entre camadas.

Fonte: Luckow e Melo, 2010, p. 180.

Observa-se assim, que nesta ilustração aparecem não somente as classes POJO que estão presentes em todo o processo, como também a camada de regras

de negócio (RN) e a camada de apresentação, esta última que será descrita em tópicos posteriores. A camada de RN é responsável pela “tomada de decisão”, acessando diretamente a camada de acesso ao banco de dados, representada aqui pelas classes DAO. Assim, foram criadas classes para cada entidade que apresenta necessidade de comunicar-se com a base de dados, seguindo um padrão de nomenclatura como a classe da imagem abaixo, por exemplo, “PessoaRN.java”.

```
7   public class PessoaRN {
8
9       private InterfaceHibernateDAO<Pessoa_> pessoaDAO;
10
11   public PessoaRN() {
12       this.pessoaDAO = DAOFactory.criarPessoaDAO();
13   }
14
15   public void salvar(Pessoa_ pessoa) {
16       this.pessoaDAO.salvar(pessoa);
17   }
18   public List<Pessoa_> listar() {
19       return this.pessoaDAO.listar();
20   }
21   public void atualizar(Pessoa_ pessoa) {
22       this.pessoaDAO.atualizar(pessoa);
23   }
24   public void excluir(Pessoa_ pessoa) {
25       this.pessoaDAO.excluir(pessoa);
26   }
27   public Pessoa_ carregar(Integer codigo) {
28       return this.pessoaDAO.carregar(codigo);
29   }
30 }
```

Figura 13 - Classe “PessoaRN.java” do SCPREVI.

A tomada de decisão citada trata-se de uma instrução à classe DAO para execução de determinada transação no banco de dados. Assim, jamais será possível a camada de apresentação acessar diretamente a camada de acesso ao banco de dados. Logo, com essa configuração, o procedimento de acesso ao banco de dados da aplicação segue a seguinte ordem, considerando que ainda não foi citada a descrição completa de camadas:

- É solicitado um acesso ao banco de dados iniciando-se a partir da regra de negócio;
- Classe “EntidadeRN.java” cria uma instância de “DAOFactory.java” e invoca o método correspondente;

- Classe “*DAOFactory.java*” cria uma instância de “*HibernateDAO<Entidade>*” e atribui a ela uma sessão a partir do “*HibernateUtil.java*”.

A partir desse momento o objeto “*dao*”, que contém todos os métodos da classe “*HibernateDAO<T>*”, já está disponível para executar posteriormente as transações desejadas no banco de dados.

3.2.3 Padrão de Projeto MVC

O MVC (*Model, View and Controller*, ou em tradução livre, “Modelo, Visão e Controle”) é um padrão de projeto bastante utilizado pelo fato de ter como uma de suas características, assim como o DAO, a separação de responsabilidades. Entretanto o padrão de projeto DAO atua na camada de acesso ao banco de dados do sistema e o padrão MVC atua na camada de apresentação. Segundo Luckow e Melo (2010, p. 180), “o padrão MVC determina que um sistema seja separado em três camadas, chamadas de *Model*, *View* e *Controller*, mas o principal objetivo do MVC não é definir como separar em camadas, mas sim definir como as camadas devem interagir”. Dessa forma, segue abaixo uma breve descrição de cada uma dessas camadas:

- **Model** - é a camada responsável pelo acesso aos dados da aplicação, ou seja, nela estão descritas as classes que executam diretamente as transações com o banco de dados do sistema. Compete a esta camada, toda a parte de conexão com o banco de dados (no âmbito deste projeto, acesso a dados utilizando DAO) e mapeamento ORM;
- **View** - esta camada tem como objetivo representar toda a interface da aplicação. Dessa forma, depois de feito o acesso a dados na camada *Model*, a *View* fica responsável pela exibição dos dados ao usuário;
- **Controller** - esta camada tem a função de unir as duas camadas anteriores (*Model* e *View*), de forma a buscar as informações da *Model* para gerar a *View* e, da mesma forma, receber as informações da *View* enviando-as para a *Model*.

Segue abaixo uma imagem representando a arquitetura do padrão de projeto MVC:

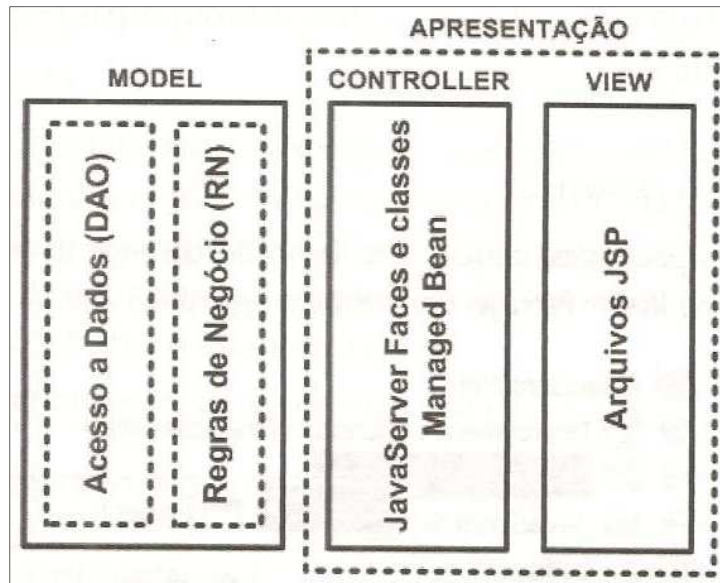


Figura 14 - Arquitetura do padrão de projeto MVC.

Fonte: Luckow e Melo, 2010, p. 181.

Dessa forma, observa-se que o padrão MVC está diretamente ligado ao padrão DAO, pois suas camadas e objetivos conversam entre si. Assim, é ideal a junção dessas duas técnicas para garantir produtividade ao desenvolvimento.

3.2.4 Técnica Open Session In View

Observando-se a **Figura 14**, nota-se que uma das exigências mais importantes do *Hibernate* não foi configurada: a abertura de uma transação a partir da sessão e posteriormente confirmação da mesma. Isto, para que fosse possível a utilização de uma técnica bastante funcional e produtiva desenvolvida pela comunidade do *Hibernate*, a *Open Session In View*, traduzido livremente para Sessão Aberta na Visão. Essa técnica foi utilizada neste projeto devido ao fato de, a partir de sua utilização, não existir a necessidade de criar uma nova conexão ao banco de dados para cada operação, pois dessa forma, os recursos do banco seriam consumidos de forma exagerada, deixando lento o processamento da página.

Isso é garantido com esta técnica, pois, toda requisição que chega ao servidor é interceptada por um filtro que somente fecha a sessão do *Hibernate* aberta anteriormente, quando a operação é concluída. Para isso, utilizou-se os *Filters* do *JavaEE*. Assim, segue abaixo o filtro desenvolvido para o SCPREVI, intitulado “*ConexãoHibernateFilter.java*”:

```

6  public class ConexaoHibernateFilter implements Filter {
7      private SessionFactory sf;
8      @Override
9      public void init(FilterConfig config) throws ServletException {
10         this.sf = HibernateUtil.getSessionFactory();
11     }
12     @Override
13     public void destroy() {
14     }
15     @Override
16     public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse,
17         FilterChain chain) throws ServletException {
18         try {
19             this.sf.getCurrentSession().beginTransaction();
20             chain.doFilter(servletRequest, servletResponse);
21             this.sf.getCurrentSession().getTransaction().commit();
22             this.sf.getCurrentSession().close();
23         } catch (Throwable ex) {
24             try {
25                 if (this.sf.getCurrentSession().getTransaction().isActive()) {
26                     this.sf.getCurrentSession().getTransaction().rollback();
27                 }
28             } catch (Throwable t) {
29                 System.out.println("ERRO: " + t.getMessage());
30             }
31             throw new ServletException(ex);
32         }
33     }
34 }

```

Figura 15 - Filtro “ConexaoHibernateFilter.java” do SCPREVI.

Observa-se que a classe implementa a *interface Filter*, recurso da especificação *Servlet*, encontrado no *JavaEE*. Esta *interface* exige a implementação de três métodos: *init*, *doFilter* e *destroy*. O método *init* somente é executado quando a aplicação é colocada em execução no servidor e o método *destroy* é executado quando o servidor é desativado. Já o método *doFilter* é a parte principal desta classe, e, portanto, segue logo abaixo uma descrição das principais funcionalidades do método:

- **Linha 19:** este procedimento é executado antes de qualquer processamento do sistema, pois logo após é executado o processo mais importante do método. Por isso, foi iniciada uma transação do *hibernate* neste ponto;
- **Linha 20:** ponto mais importante do método, pois é neste momento que o processo segue em frente para execução normal. Caso esta linha não fosse executada, a iniciação da transação e o resto do procedimento não seriam alcançados;
- **Linha 21:** é neste momento onde a requisição finalmente é executada, concluindo a transação e, posteriormente, fechando a sessão.

Vale ressaltar que esta técnica foi desenvolvida, sobretudo, para servir de solução a um dos principais erros que os programadores iniciantes do *framework* enfrentam, uma exceção intitulada *LazyInitializationException*. Esta exceção ocorre principalmente quando o mapeamento de determinadas entidades da aplicação é feito através do relacionamento *Lazy*, que informa ao *Hibernate* para executar os *selects* somente quando forem necessários. Com a entidade mapeada dessa forma, haverá sempre um momento em que a sessão que carregou o objeto já estará fechada, fazendo com que o *Hibernate* lance a exceção aqui citada. Entretanto, com a configuração da técnica *Open Session In View*, este problema não acontecerá, pois existirá uma sessão aberta até que seja concluída a transação. Por essas razões, a camada de acesso ao banco de dados do SCPREVI foi configurada de acordo com a técnica aqui ilustrada.

3.2.5 Framework de desenvolvimento JavaServer Faces

JavaServer Faces (JSF) é simplesmente uma das plataformas mais utilizadas atualmente no que diz respeito a desenvolvimento de aplicativos *web* com alta produtividade. Quando se trabalha com desenvolvimento *web* com a linguagem *Java*, é crucial a utilização de tal tecnologia, mesmo que outras tecnologias do ambiente *Java Web* ainda sejam muito utilizadas, como por exemplo, JSP (*Java Server Pages*) e JSTL (*Java Standard Tag Library*). Mas é importante seguir novos conceitos e a plataforma mais adequada para determinada aplicação, sobretudo quando esta tecnologia aumenta a produtividade dos desenvolvedores. Dessa forma, é de grande importância demonstrar de forma clara o que é de fato o *JavaServer Faces*.

JavaServerFaces (JSF) é a especificação para um *framework* de componentes para desenvolvimento *web* em *Java*. Essa especificação foi definida por meio do *Java Community Process* (JCP), que é a entidade que tem como objetivo especificar a evolução da linguagem *Java* de acordo com o interesse do mercado e não apenas da *Sun*, criadora da linguagem *Java*. (LUCKOW e MELO, 2010, p.73).

O JSF trabalha com a seguinte padronização:

- **Desenvolvimento de uma classe denominada *Backing Bean***, uma classe *Bean* padrão. Essa classe segue a idéia de implementação de métodos privados e os seus “*gets/sets*” da mesma maneira que as classes POJO. Essa classe também pode ser chamada de

ManagedBean e encontra-se na camada de apresentação da aplicação ou, no caso do MVC, na camada visão;

- **Mapeamento da classe Bean.** Após o seu desenvolvimento, é necessário realizar o mapeamento, que a partir da versão **2.x** do JSF (utilizada neste projeto), pode ser feita apenas com anotações nas classes, em vez de mapeamento em arquivos XML. Nesta anotação, informa-se o escopo¹⁵ da classe, que pode ser *application*, *session*, *request*, *none*, *view* e *custom*. Dentre estes, para este projeto adotou-se o escopo *session*, pois ele cria uma instância para cada usuário e também, pelo fato de ter sido executada a configuração da técnica *Open Session In View*;
- **Criação da página JSF** através de seus componentes básicos ou através de algum *framework* baseado no JSF. Vale lembrar que o JSF utiliza páginas XHTML¹⁶, onde são interpretados os componentes JSF e transformados em “tags” interpretadas pela HTML¹⁷.

O JSF apresenta como um de seus pontos fortes, um **ciclo de vida** ou, como os desenvolvedores JSF conhecem, **Life Cicle** que, por sua vez, foi muito bem desenvolvido. Isso se justifica de maneira positiva, pois, com o ciclo de vida, o processamento interno do *framework* sempre será garantido, mesmo que o ciclo permita que, em determinados momentos, o desenvolvedor avance para outro ponto do ciclo, o que não impede o seu funcionamento. Dessa forma, segue abaixo uma ilustração do ciclo de vida do JSF:

¹⁵ Escopo é o tempo de vida da instância da classe *Backing Bean* durante execução da aplicação no servidor web.

¹⁶ *Extensible Hypertext Markup Language*, traduzido para o português como “Linguagem de Marcação de Hipertexto Extensível”. Trata-se da linguagem HTML com base na XML.

¹⁷ *Hypertext Markup Language*, traduzido para o português como “Linguagem de Marcação de Hipertexto”. Linguagem utilizada para produzir páginas web estáticas.

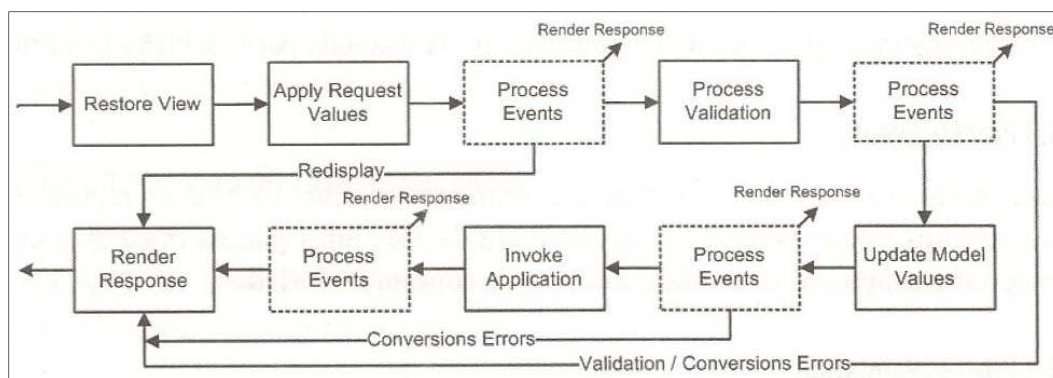


Figura 16 - Ciclo de vida do JSF.
Fonte: Luckow e Melo, 2010, p. 101.

De acordo com a documentação do JSF e observando a **Figura 21**, entende-se que foi designada para cada fase uma tarefa diferente. Na fase *Restore View* o JSF busca a URL¹⁸ solicitada após a requisição chegar ao servidor. Na próxima fase, a *Apply Request Values*, o JSF preenche a árvore de componentes requisitados em memória e aplica as conversões de valores necessárias. Na próxima fase, *Process Events*, o JSF executa cada evento registrado nos componentes, para que possa ser feita, posteriormente, a validação na fase seguinte, *Process Validation*. Logo, será possível passar os valores registrados e validados para a classe *Bean*, na fase *Update Model Values*. Na fase seguinte, *Invoke Application*, após a validação, conversão e atribuição de valores às respectivas propriedades da classe *Bean*, é o momento de invocar o método designado. E ao fim, chega-se à fase *Render Response*, que é a resposta montada e devolvida ao navegador.

3.2.6 Biblioteca de componentes *PrimeFaces*

Para a camada de visualização dos dados do SCPREVI, adotou-se a biblioteca de componentes JSF *PrimeFaces*. O *PrimeFaces* é uma biblioteca que dispõe de mais de 90 componentes para *JavaServer Faces* (v2.1)¹⁹. Dessa forma é de vital produtividade utilizar o *PrimeFaces*, sobretudo porque ele contém de forma implícita a maioria das funcionalidades *Ajax*²⁰ e *JavaScript*²¹ (em forma de

¹⁸ Do Inglês *Uniform Resource Location*. É o endereço para se chegar há uma página de destino na web.

¹⁹ Versão 2.1 do JSF, que foi utilizada neste projeto.

²⁰ "AJAX é a sigla em inglês para *Asynchronous JavaScript and XML* e trata-se de uma técnica de carregamento de conteúdos em uma página web com o uso de *JavaScript* e *XML*". (SILVA, 2009, p. 23).

componentes), tornando o desenvolvimento em um processo ágil. O *PrimeFaces* possui todas as bibliotecas padrão do JSF, mas com uma interface diferente e única. A versão do *PrimeFaces* utilizada para o desenvolvimento do SCPREVI foi a **3.0**.

A biblioteca *PrimeFaces* permite o desenvolvedor escolher temas prontos para utilizar em sua aplicação e, não menos importante, permite alteração do tema via interface gráfica. Isso é possível através de uma UI²² do *jQuery*²³. Assim, o desenvolvedor pode escolher os valores de cada componente de acordo com suas necessidades e logo após, fazer o *download* do tema e importar nas páginas da aplicação.

Um dos componentes mais utilizados do *PrimeFaces* é o componente `<p:dataTable>`. De acordo com Luckow e Melo (2010, p. 625), este componente “corresponde ao `<h:dataTable>` dos componentes do JSF HTML, mas com vários recursos adicionais, como paginação, ordenação, rolagem, seleção, carregamento tardio, filtro e redimensionamento de colunas, dentre outros”. A configuração deste componente não é complicada. Segue abaixo uma ilustração demonstrando a utilização desse componente no SCPREVI, frisando somente a parte de parametrização dos atributos na página.

```
<p:dataTable paginator="true" rows="5" var="servlip"
  paginatorTemplate="{CurrentPageReport}
    {FirstPageLink} {PreviousPageLink} {PageLinks}
    {NextPageLink} {LastPageLink} {RowsPerPageDropdown}"
  rowsPerPageTemplate="5,10,15"
  emptyMessage="Nenhum registro encontrado."
  value="#{servidorLipBean.lista}"
  style="margin-left: 206px;" >
```

Figura 17 - Demonstração de configuração do componente "`<p:dataTable>`".

Os atributos são auto-explicativos, e o resultado desta configuração pode ser observado na **Figura A.3**, Anexo A.

²¹ “JavaScript foi criada pela NetScape em parceria com a Sun, com a finalidade de fornecer um meio de adicionar interatividade a uma página web”. (SILVA, 2010, p. 23).

²² Do Inglês *User Interface* ou “Interface com Usuário”.

²³ *Framework* da linguagem de script JavaScript, utilizado para desenvolver eventos com mais facilidade no lado cliente.

3.2.7 Utilizando *Templates* com *Facelets* e Linguagem CSS

Segundo Luckow e Melo (2010, p. 294), o *Facelets* “é um *framework* de *templates* que ajuda a diminuir a quantidade de código-fonte das páginas padronizando e centralizando as definições visuais e estruturais das telas do sistema”. Dessa forma, torna-se de fácil adequação o desenvolvimento e manutenção da aplicação, pois o *Facelets* possibilita a criação das chamadas **páginas-modelos**, que servem de visual e base para as demais páginas do sistema. Por exemplo, para o SCPREVI, foram criadas as seguintes *templates*:

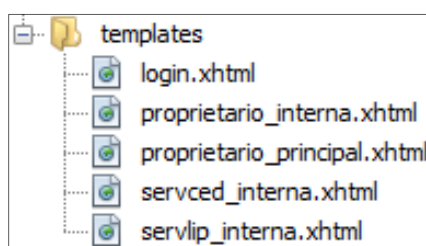


Figura 18 - *Templates Facelets* do SCPREVI.

Logo, para determinadas páginas, utiliza-se determinada *template*. Por exemplo, a *template* “*proprietario_interna.xhtml*” é destinada às páginas de Servidor LIP para usuário com permissão de proprietário²⁴, que executam algum procedimento interno. Dessa forma, nas páginas da aplicação é possível importá-las da seguinte maneira:

```

3 <ui:composition xmlns:ui="http://java.sun.com/jsf/facelets"
4   template="/resources/templates/proprietario_principal.xhtml"
5   xmlns:p="http://primefaces.org/ui"
6   xmlns:h="http://java.sun.com/jsf/html"
7   xmlns:f="http://java.sun.com/jsf/core">

```

Figura 19 - Invocação de *template* SCPREV.

Ou seja, através da tag “*<ui:composition>*” com atributo “*template*”, é possível passar o caminho da página de *template* desejada para esta página (linha quatro). Dessa forma, a página assume todos os componentes existentes da *template* e o desenvolvedor adiciona apenas os componentes específicos a cada página. Isso facilita muito a vida do desenvolvedor, aumentando a produtividade, sobretudo em relação à manutenção. Quando por exemplo for necessária a alteração de alguma funcionalidade no *layout* somente será necessário alteração nas *templates*. Um

²⁴ A permissão de proprietário está descrita neste projeto como “ROLE_ADMINISTRADOR”, vide Figura 30, linha nove.

exemplo está na **Figura 25** logo abaixo, extraída do código-fonte da *template* “*proprietario_principal.xhtml*”, onde está implementado o rodapé de todas as páginas:

```

31 <p:panel>
32   <div id="rodape">
33     Instituto de Previdência Social do Município de Palmas <br/>
34     <strong>PREVIPALMAS</strong> &copy; 2011
35     Telefone: (63) 3218-5050 - Fax: (63) 3218-5042 <br/>
36   </div>
37 </p:panel>

```

Figura 20 - Exemplo de implementação de *template* SCPREVI.

Aliada a isto, encontra-se a linguagem CSS (*Cascading Style Sheets*), em português, “Folha de Estilo em Cascata”, uma linguagem utilizada no ambiente *web*, que define estilos e posicionamentos de determinada informação. A CSS permite organizar de forma padronizada e de acordo com os padrões *web* (*Web Standards*), a informação estruturada e marcada pela HTML e, no caso deste projeto, pela XHTML. Entretanto, pelo fato da utilização do *PrimeFaces* e seus temas neste projeto, a CSS foi utilizada basicamente para posicionamento dos componentes, não somente os componentes das *templates*, como também, os componentes das demais páginas XHTML utilizadas. Dessa forma, segue logo abaixo a configuração no arquivo CSS para o “*div*” da *template* “*proprietario_principal.xhtml*”, ilustrada na **Figura 25**.

```

56 #rodape {
57   margin: 0px 0px 0px 0px;
58   text-align: center;
59   font-family: "Trebuchet MS", Arial, Helvetica, sans-serif;
60 }

```

Figura 21 - Configuração do “*div*” rodapé da *template* “*proprietario_principal.xhtml*” do SCPREVI.

3.2.8 Framework de Segurança Spring Security

Spring Security é um projeto incluído no *Spring Framework*. Segundo Luckow e Melo (2010, p. 232), “o *Spring Security* é um dos projetos *Spring* mais maduros. Existente desde 2003 é utilizado em milhares de projetos pelo mundo, incluindo agências governamentais e militares”. Dessa forma, entende-se que toda aplicação que será disponibilizada através da *Internet*, deve obrigatoriamente apresentar um padrão de segurança de informações. No SCPREVI, o *Spring Security* foi utilizado basicamente para a parametrização de controle de acesso às páginas do sistema.

Assim, é válido frisar o funcionamento do controle de *login* do *Spring Security*, explanado claramente no diagrama de atividades abaixo:

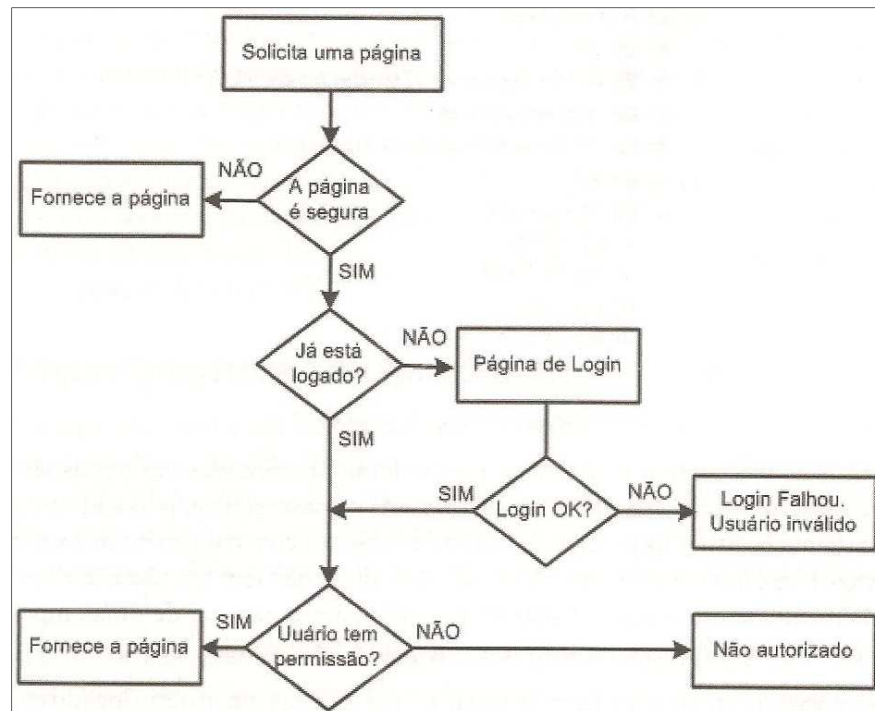


Figura 22 - Funcionamento do *Spring Security*.

Fonte: Luckow e Melo, 2010, p. 234.

Portanto, percebe-se que o usuário não conseguirá acessar o conteúdo de uma página se não estiver devidamente autenticado no sistema. Para realizar essa autenticação, o *Spring Security* realiza uma consulta no banco de dados para conferir se o usuário está cadastrado e se seus dados são verdadeiros. Para isso, o *Spring Security* necessita obviamente de conectar-se ao banco. Dessa forma, justifica-se a configuração implementada neste projeto de conexão via *Data Source* (vide **Figura 11**), pois assim, foi possível compartilhar uma mesma conexão para dois *Frameworks* acessarem o banco, neste caso, o *Spring* e o *Hibernate*. Logo, segue abaixo uma imagem ilustrando tal configuração:

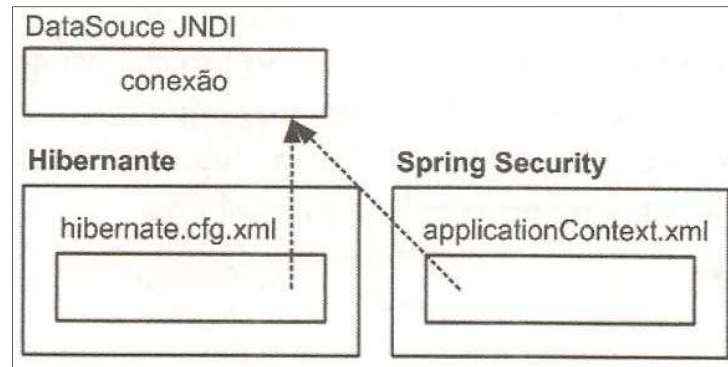


Figura 23 - Compartilhamento de conexão via *Data Source* JNDI.
Fonte: Luckow e Melo, 2010, p. 236.

Assim, como ilustra a **Figura 28**, é necessário ainda, para concluir a configuração do *Spring Security*, parametrizar não somente o arquivo “*applicationContext.xml*” como também o arquivo “*applicationContext-security.xml*”, ambos localizados no diretório “WEB-INF” da aplicação. Logo, segue abaixo uma ilustração dos dois arquivos utilizados para a configuração de “login” do SCPREVI:

```

6 <bean id="scpreviDataSource"
7     class="org.springframework.jndi.JndiObjectFactoryBean">
8     <property name="jndiName">
9         <value>java:comp/env/jdbc/SCPREVI_Resource</value>
10    </property>
11 </bean>

```

Figura 24 - Arquivo “*applicationContext.xml*” do SCPREVI.

```

8 <http>
9     <intercept-url pattern="/proprietario/**" access="ROLE_ADMINISTRADOR" />
10    <intercept-url pattern="/cliente/ServCed/**" access="ROLE_SERVCED" />
11    <intercept-url pattern="/cliente/ServLip/**" access="ROLE_SERVLIP" />
12
13    <form-login login-page="/index.jsf"
14        always-use-default-target="true" default-target-url="/login/principal.jsf"
15        authentication-failure-url="/index.jsf?login_error=1" />
16    <logout />
17 </http>
18
19 <authentication-manager>
20     <authentication-provider>
21         <jdbc-user-service data-source-ref="scpreviDataSource"
22             users-by-username-query="
23                 SELECT login, senha, ativo
24                 FROM usuario
25                 WHERE login = ?"
26             authorities-by-username-query="
27                 SELECT u.login, p.permissao
28                 FROM usuario u, usuario_permissao p
29                 WHERE p.fk_permissao_usuario = u.cod_usuario
30                 AND login = ?" />
31     </authentication-provider>
32 </authentication-manager>

```

Figura 25 - Arquivo “*applicationContext-security.xml*” do SCPREVI.

De acordo com as imagens, percebe-se que na **Figura 29**, é feita a configuração de acesso ao banco de dados, ou melhor, ao *Data Source*, da mesma forma que foi apontado na configuração do *Hibernate*, vide **Figura 10**. Na **Figura 30**, é de fato configurada a segurança do *Spring Security*. Assim, nas linhas de nove a onze, são efetuadas as permissões dos usuários e quais páginas esses usuários têm acesso. Dentro da tag “<form-login>” parametrizou-se qual o caminho da página de *login*, qual o caminho para a página após o *login* bem sucedido, qual o caminho para a página caso o *login* não tenha sido executado com sucesso e *logout*.

Na tag seguinte “<authentication-manager>” são realizadas as consultas ao banco de dados para conferir se o usuário em questão está cadastrado devidamente com alguma permissão. Assim, o *Spring Security* garante a segurança da aplicação com bastante facilidade. Um exemplo da tela de *login* utilizada no SCPREVI pode ser observado na **Figura A.1**, Anexo A. Não menos importante, é necessário configurar o arquivo “web.xml” da aplicação, adicionando as parametrizações do *Spring Security*. Segue abaixo como foi feita essa configuração no “web.xml” do SCPREVI:

```

42 <!-- Spring Security -->
43 <!-- Informa arquivos de config. disponíveis -->
44 <context-param>
45   <param-name>contextConfigLocation</param-name>
46   <param-value>
47     /WEB-INF/applicationContext.xml
48     /WEB-INF/applicationContext-security.xml
49   </param-value>
50 </context-param>
51 <!-- Permite Spring Security interceptar as requisições -->
52 <filter>
53   <filter-name>springSecurityFilterChain</filter-name>
54   <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
55 </filter>
56 <filter-mapping>
57   <filter-name>springSecurityFilterChain</filter-name>
58   <url-pattern>/*</url-pattern>
59 </filter-mapping>
60 <!-- Permite Spring Security carregar arquivos de config. -->
61 <listener>
62   <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
63 </listener>

```

Figura 26 - Configuração do *Spring Security* no arquivo “web.xml” do SCPREVI.

Aqui cabe uma observação interessante. O *Spring Security* possibilita a restrição de determinados conteúdos de uma página de acordo com a permissão do usuário. Segue abaixo como foi feita esta configuração na tela principal do sistema exibida para todos os usuários após autenticação:

```

10 <ui:define name="menu_horizontal">
11     <!-- Menu Cliente (ServLip) -->
12     <sec:ifAnyGranted roles="ROLE_SERVLIP">
13         <ui:include src="/cliente/ServLip/menu_horizontal.xhtml" />
14     </sec:ifAnyGranted>
15
16     <!-- Menu Cliente (ServCed) -->
17     <sec:ifAnyGranted roles="ROLE_SERVCED">
18         <ui:include src="/cliente/ServCed/menu_horizontal.xhtml" />
19     </sec:ifAnyGranted>
20
21     <!-- Menu Proprietário -->
22     <sec:ifAnyGranted roles="ROLE_ADMINISTRADOR">
23         <ui:include src="/proprietario/restrito/menu_horizontal.xhtml" />
24     </sec:ifAnyGranted>
25 </ui:define>

```

Figura 27 - Restrição de conteúdo por usuário no arquivo “principal.xhtml” do SCPREVI.

Assim, vê-se que o *menu* horizontal é único para cada usuário autenticado. Ou seja, não foi necessário criar uma página para cada usuário. Percebe-se também, que cada *menu* está sendo importado através da tag “<ui:include>” do *PrimeFaces*. Essa configuração também ajuda bastante o desenvolvedor, sobretudo na manutenção do sistema, pois caso tenha que mudar ou adicionar alguma propriedade nos *menus*, o desenvolvedor tem necessidade de alterar apenas os *menus* e não todas as páginas do sistema. Portanto, o *Spring Security* foi utilizado no SCPREVI, tornando o projeto mais seguro e eficiente.

3.2.9 Projeto JRimum - Bopepo

O projeto *JRimum* é um projeto de código-fonte aberto (*Open Source*) que tem como principal objetivo o desenvolvimento de componentes a partir da linguagem de programação Java. De acordo com afirmações de seus desenvolvedores, o projeto é “fruto de uma iniciativa conjunta entre estudantes do IFRN (Instituto Federal de Educação, Ciência e Tecnologia Rio Grande do Norte) e a empresa Nordeste Fomento Mercantil” e teve como primeiro componente desenvolvido, o *Bopepo*, um componente Java para geração de boletos bancários personalizados que suporta praticamente todos os bancos do Brasil. O *Bopepo* é utilizado atualmente por várias empresas nacionais como, por exemplo, a Nordeste Fomento Mercantil, já citada, Jornal Diário do Nordeste, Unimed Porto Alegre, dentre outras.

Por se tratar de um projeto ainda novo, apresenta pouca documentação, mas dispõe de vários tutoriais para auxiliar os desenvolvedores. Logo, para o seu

funcionamento, é necessário adicionar na aplicação as seguintes bibliotecas juntamente com suas dependências, de acordo com as versões utilizadas no SCPREVI:

- *iText-2.0.8;*
- *Apache commons-lang-2.4;*
- *Apache log4j-1.2.15;*
- *JRimum-Bopepo-SNAPSHOT-20110209-bin.*

A principal característica do *Bopepo* é a facilidade na geração de boletos bancários, pois exige apenas que sejam passados por parâmetro os dados requeridos para geração do título e, logo após, a chamada a um método que faz todo o trabalho. O *Bopepo*, segue a implementação exigida pela Febraban e posteriormente, específica para cada banco. No caso do SCPREVI, utilizou-se a especificação que segue o padrão exigido pelo banco Caixa Econômica Federal. Logo, não há necessidade de preocupação com a formatação do boleto, pois já será gerado nos padrões exigidos. Ainda assim, o *Bopepo* concede a oportunidade aos desenvolvedores de personalizar o boleto de acordo com as exigências da empresa.

Dessa forma, como o SCPREVI utilizou o *framework Hibernate*, é necessária apenas a passagem das entidades do *Hibernate* para as entidades do *Bopepo* por parâmetro, um processo também chamado por desenvolvedores de *setagem de parâmetros*. Assim, esta configuração foi realizada em dois métodos para cada servidor (LIP ou cedido), denominados “*geraBoleto*” e “*atualizacaoMonetaria*” em suas classes *ManagedBeans* do SCPREVI. Segue abaixo um exemplo desta configuração para servidor cedido:

```

178 public BoletoViewer atualizacaoMonetaria() {
179
180     //multa
181     double percentualMora = ((this.titulo.getValorMora() * this.titulo.getValorOriginal()) / 100);
182     double valorComMulta = (this.titulo.getValorOriginal() + percentualMora);
183     //juros
184     double percentualJuros = ((this.titulo.getValorOriginal() * 0.5) / 100); //Juros sempre de 0,5% ao mês
185     double valorComJuros = (valorComMulta + percentualJuros);
186
187     this.titulo.setValorCobrado(valorComJuros);
188     this.titulo.setIndiceCorrecao("I.N.P.C. - Índice Nacional de Preços ao Consumidor");
189     this.titulo.setAceite("A");
190     this.titulo.setValorJurosAm(valorComJuros);
191     this.titulo.setSituacao("Pendente");
192
193     EnderecoRN rnEnd = new EnderecoRN();
194     PessoaRN rnPessoa = new PessoaRN();
195     TituloRN tituloRN = new TituloRN();
196     BoletoRN boletoRN = new BoletoRN();
197
198     // seta entidades referenciadoras
199
200     this.sacado.setPessoa(rnPessoa.carregar(this.pessoa.getCodPessoa()));
201     this.endereco = rnEnd.carregar(this.sacado.getPessoa().getEndereco().getPessoaCodPessoa());

```

Figura 28 - Configuração de cálculo de atualização monetária para servidor cedido – SCPREVI.

Nesta imagem visualiza-se apenas o início do método “*atualizacaoMonetaria*”, que implementa o cálculo de multa e juros. Perceba que o método apresenta como tipo de retorno a entidade do *Bopepo* “*BoletoViwer*”, entidade responsável por implementação de geração do boleto. Assim, vê-se que a multa será calculada de acordo com um valor percentual informado pelo usuário no formulário. Já os juros sempre serão de 0,5% abatidos encima do valor original do título e posteriormente somados ao valor com multa. Logo após, são setados os valores padrões do título nas entidades da aplicação (SCPREVI). Abaixo, segue a continuação do método, que diz respeito a esta configuração:

```

204 // seta entidades para entidades bopepo // geraBoleto
205 Cedente cedenteB = new Cedente(this.cedente.getNome(), this.cedente.getCnpj());
206 Sacado sacadoB = new Sacado(this.sacado.getPessoa().getNome(), this.sacado.getPessoa().getCpf());
207
208 Endereco endB = new Endereco();
209 endB.setUF(UnidadeFederativa.valueOfSigla(this.endereco.getEstado()));
210 endB.setLocalidade(this.endereco.getCidade());
211 endB.setCep(this.endereco.getCep());
212 endB.setBairro(this.endereco.getBairro());
213 endB.setLogradouro(this.endereco.getLogradouro());
214 endB.setNumero(this.endereco.getNumero());
215 sacadoB.addEndereco(endB);
216
217 ContaBancaria contaBancariaB = new ContaBancaria(BancosSuportados.CAIXA_ECONOMICA_FEDERAL.create());
218 contaBancariaB.setNumeroDaConta(new NumeroDaConta(Integer.parseInt(this.contaBancaria.getNumeroConta()),
219     this.contaBancaria.getDigNumeroConta()));
220 contaBancariaB.setCarteira(new Carteira(this.contaBancaria.getCarteira())); // VALOR PADRÃO PARA TODOS OS BOLETOS
221 contaBancariaB.setAgencia(new Agencia(Integer.parseInt(this.contaBancaria.getNumAgencia()),
222     this.contaBancaria.getDigAgencia()));
223
224 Titulo tituloB = new Titulo(contaBancariaB, sacadoB, cedenteB);
225 tituloB.setNumeroDoDocumento(this.titulo.getNumDocumento());
226 tituloB.setNossoNumero(this.titulo.getNossoNumero());
227 tituloB.setDigitoDoNossoNumero(this.titulo.getDigNossoNumero());
228 tituloB.setValor(BigDecimal.valueOf(this.titulo.getValorOriginal()));
229 tituloB.setDataDoDocumento(this.titulo.getDataDocumento());
230 tituloB.setDataDoVencimento(this.titulo.getDataVencimento());
231 tituloB.setTipoDeDocumento(TipoDeTitulo.ND_NOTA_DE_DEBITO); // VALOR PADRÃO PARA TODOS OS BOLETOS
232 tituloB.setAceite(Titulo.EnumAceite.A); // VALOR PADRÃO PARA TODOS OS BOLETOS
233 tituloB.setDesconto(BigDecimal.valueOf(this.titulo.getValorDesconto()));
234 tituloB.setDeducao(BigDecimal.valueOf(this.titulo.getValorDeducao()));
235 tituloB.setMora(BigDecimal.valueOf(this.titulo.getValorMora()));
236 tituloB.setAcrecimo(BigDecimal.valueOf(this.titulo.getValorAcrecimo()));
237 tituloB.setValorCobrado(BigDecimal.valueOf(valorComJuros));

```

Figura 29 - Continuação do método “*atualizacaoMonetaria*” para servidor cedido – SCPREVI.

Não menos importante, segue logo abaixo, a continuação do método que seta a entidade principal do *Bopepo* (Boleto), persiste os dados na aplicação (SCPREVI) e passa por parâmetro o valor necessário para geração do boleto a partir do método do *Bopepo* “*BoletoViwer*”.

```

239 Boleto boletoB = new Boleto(tituloB);
240 boletoB.setLocalPagamento(this.boleto.getLocalPagamento());
241 boletoB.setInstrucaoAoSacado(this.boleto.getInstrucaoSacado());
242 boletoB.setInstrucao1(this.boleto.getInstrucao1());
243 boletoB.setInstrucao2(this.boleto.getInstrucao2());
244 boletoB.setInstrucao3(this.boleto.getInstrucao3());
245 boletoB.setInstrucao4(this.boleto.getInstrucao4());
246 boletoB.setInstrucao5(this.boleto.getInstrucao5());
247 boletoB.setInstrucao6(this.boleto.getInstrucao6());
248 boletoB.setInstrucao7(this.boleto.getInstrucao7());
249 boletoB.setInstrucao8(this.boleto.getInstrucao8());
250
251 // PERSISTE DADOS ENTIDADES SCPREVI
252 this.titulo.setCedente(cedente);
253 this.contaBancaria.setCedente(cedente);
254 this.titulo.setSacado(this.sacado);
255 this.titulo.setBoleto(boleto);
256 this.boleto.setTitulo(this.titulo);
257
258 tituloRN.salvar(this.titulo);
259 boletoRN.salvar(this.boleto);
260
261 BoletoViewer bol = new BoletoViewer(boletoB);
262 return bol;
263
264 }

```

Figura 30 - Invocação do método “*BoletoViwer*” do *Bopepo* para geração do boleto SCPREVI.

Assim, percebe-se que o método “*atualizacaoMonetaria*” retorna um objeto com todos os dados necessário para geração do boleto. Logo, este método é

invocado em outro método na mesma classe, intitulado “*geraBoleto*”, que ao mesmo tempo em que disponibiliza o arquivo com extensão “.pdf” para o usuário no navegador, persiste o arquivo no banco de dados do SCPREVI e o escreve em um diretório da aplicação para posterior acesso. Segue logo abaixo a **Figura 36**, que demonstra este código-fonte:

```

266 public void geraBoleto(){
267
268     BoletoViewer boletoview = atualizacaoMonetaria();
269     byte[] pdfAsBytes_web = boletoview.getPdfAsByteArray();
270     HttpServletResponse response = (HttpServletResponse)
271         FacesContext.getCurrentInstance().getExternalContext().getResponse();
272
273     try {
274
275         response.setContentType("application/pdf");
276         response.setHeader("Content-Disposition", "attachment; filename =" +
277             this.pessoa.getNome().replaceAll(" ", "_") + "_boleto_" + this.titulo.getMes() + ".pdf");
278
279         String local = FacesContext.getCurrentInstance().getExternalContext().getRealPath("boletos");
280         File diretorio = new File(local);
281         if (!diretorio.exists()) {
282             diretorio.mkdir();
283         }
284         File arqBol = boletoview.getPdfAsFile(this.pessoa.getNome().replaceAll(" ", "_") +
285             "_boleto_" + this.titulo.getMes() + ".pdf");
286         String localArquivo = local + "/" + arqBol.toString();
287         FileOutputStream output_arq = new FileOutputStream(localArquivo);
288         output_arq.write(pdfAsBytes_web);
289         TituloRN rn = new TituloRN();
290         this.titulo.setLocalBoleto(arqBol.toString());
291         rn.atualizar(titulo);
292         output_arq.close();
293
294         OutputStream output = response.getOutputStream();
295         output.write(pdfAsBytes_web);
296         response.flushBuffer();
297         FacesContext.getCurrentInstance().responseComplete();
298
299     } catch (IOException e) {
300         System.out.println("ERRO AO GERAR BOLETO: " + e.getMessage());
301     }
302 }

```

Figura 31 - Método “*geraBoleto*” de servidor cedido – SCPREVI.

Como dito anteriormente, o método faz uma invocação ao método “*atualizacaoMonetaria*” e atribui o objeto à outra variável de mesmo tipo (*BoletoViwer*). Uma das formas que o *Bopepo* disponibiliza para geração de arquivo PDF do boleto é através de *array*²⁵ de *bytes*. Assim, para que seja possível a disponibilização do boleto via protocolo HTTP²⁶, para que o usuário possa fazer o *download* do boleto no próprio *browser*, é preciso utilizar um objeto do tipo *HttpServletResponse*²⁷.

²⁵ Segundo Jandl (2008, p.25), arranjos (*arrays*) são “estruturas que armazenam um número fixo de elementos do mesmo tipo. Cada elemento é como uma variável simples deste tipo, que permite a escrita ou leitura de valores”.

²⁶ Do Inglês *HyperText Transfer Protocol* ou “Protocolo de Transferência de Hipertexto”.

²⁷ Disponível na API *Servlet* do Java.

A obtenção deste objeto pode ser visualizada nas linhas 270 e 271. Logo após, já é possível configurar os atributos necessários para disponibilização do boleto, o que pode ser observado nas linhas 275 e 276. O valor informado para *ContentType* foi “*application/pdf*” pois o arquivo tratado será gerado com a extensão “.pdf”. No *Header*, é informado de que forma o arquivo será demonstrado no *browser*, através do valor “*Content-Disposition*”. Configurou-se então, este atributo com o valor “*attachment*” que significa que o arquivo não será aberto automaticamente após geração, ou seja, será possível antes, o usuário escolher se deseja abrir ou salvar o arquivo em seu computador, vide **Figura A.6**, Anexo A. E, não menos importante, logo após (linha 277), é configurado o nome do arquivo seguido de sua extensão.

Dessa forma, ilustrou-se a parte mais importante do método, que diz respeito à geração do boleto e disponibilização ao usuário no próprio *browser*. As linhas seguintes são referentes à implementação da escrita do arquivo, onde será salvo em um diretório e persistido no banco de dados do SCPREVI, garantindo a segurança desta transação. Um exemplo de visualização do boleto gerado no *browser* pode ser observado na **Figura A.8**, Anexo A.

3.2.10 Projeto *JRimum* - *Texgit*

O projeto *JRimum* apresenta outro *framework* baseado na linguagem Java e que complementa o *Bopepo*, no que diz respeito à transação bancárias. Este componente chama-se *Texgit*, que tem o objetivo de implementar geração de arquivos texto através de objetos. Como informado anteriormente no tópico 1.3 deste projeto, para comunicação da aplicação com o banco é necessária geração do arquivo remessa. Para que seja possível o funcionamento do *Texgit* é necessário adicionar as seguintes bibliotecas que foram utilizadas no SCPREVI, juntamente com suas dependências:

- *Apache commons-io-2.0*;
- *Apache commons-lang-2.4*;
- *JRimum-Texgit-Utilix-0.2.0-SNAPSHOT-20101207*;
- *Texgit-0.2.0-SNAPSHOT-texgit-dependencias-jrimum*.

O *Texgit* funciona da seguinte maneira: para geração do arquivo final, é necessária antes uma configuração de um arquivo XML (*layout*). Neste arquivo, serão parametrizadas todas as configurações do arquivo final como, por exemplo, tipo e tamanho de cada campo. Esta configuração deve ser feita pelo próprio desenvolvedor, e deve ser dividida em grupos de acordo com a exigência do banco como, por exemplo, “*Header-Arquivo*”, “*Registro-Detalhe-Segmento-P*”, dentre outros, em *tags* XML específicas do *Texgit*. Essas *tags* devem seguir uma ordem de hierarquia de acordo com a estrutura exemplificada na imagem abaixo:

```

6  <FlatFile>
7      <layout>
8          <name> </name>
9          <version> </version>
10         <description> </description>
11     </layout>
12     <GroupOfRecords>
13         <Record name="">
14             <GroupOfFields>
15                 <IdType name="" value="" length="" position="" />
16                 <Field name="" type="" length="" />
17                 <Field name="" type="" length="" />
18                 <Field name="" type="" length="" />
19                 <Field name="" type="" length="" />
20
21                 <!-- demais campos do arquivo -->
22             </GroupOfFields>
23         </Record>
24     </GroupOfRecords>
25 </FlatFile>

```

Figura 32 - Estrutura de *Layout* de arquivo XML do *Texgit*.

Portanto, é dentro da *tag* “<GroupOfFields>” que são parametrizados os grupos do arquivo remessa exigidos pelo banco. Dessa forma, desenvolveu-se antes, este *Layout* em um arquivo XML para a geração de arquivo remessa dos boletos do SCPREVI, de acordo com a exigência do banco Caixa Econômica, seguindo o manual do banco descrito no padrão **CNAB-240-SIGCB** e disponível no próprio site do banco. Assim, segue abaixo uma parte do *layout* desenvolvido para o SCPREVI:

```

21 <!-- CAIXA ECONÔMICA: HEADER -->
22 <!-- CAIXA ECONÔMICA: HEADER -->
23 <Record name="Header-Arquivo">
24   <GroupOfFields>
25     <IdType name="IDReg" value="01" length="2" position="1" />
26     <!-- CONTROLE -->
27     <Field name="HA_Banco" type="INTEGER" length="3" /> <!-- Default '104' -->
28     <Field name="HA_Lote" type="INTEGER" length="4" /> <!-- Default '0000' -->
29     <Field name="HA_Registro" type="INTEGER" length="1" /> <!-- Default '0' -->
30     <Field name="HA_CNAB_1" type="FLOAT" length="9" />
31     <!-- Uso exclusivo FEBRABAN/CNAB - Default 'NULL' -->
32
33     <!-- EMPRESA -->
34     <Field name="HA_InscTipo" type="INTEGER" length="1" /> <!-- Tipo de inscrição da empresa -->
35     <Field name="HA_InscNum" type="CHARACTER" length="18" /> <!-- Núm. de inscrição da empresa -->
36     <Field name="HA_UsoExclusivoCAIXA_1" type="INTEGER" length="20" /> <!-- Default '0' -->
37     <Field name="HA_CodIdentAgencia" type="INTEGER" length="5" />
38     <!-- Cód. de identificação da agência da conta da empresa -->
39     <Field name="HA_CodIdentAgenciaDig" type="INTEGER" length="1" />
40     <!-- Dígito verificador da agência da conta da empresa -->
41     <Field name="HA_CodIdentCedente" type="INTEGER" length="6" /> <!-- Código de convênio no banco -->
42     <Field name="HA_UsoExclusivoCAIXA_2" type="INTEGER" length="7" /> <!-- Default '0' -->
43     <Field name="HA_UsoExclusivoCAIXA_3" type="INTEGER" length="1" /> <!-- Default '0' -->
44     <Field name="HA_NomeEmpresa" type="CHARACTER" length="30" /> <!-- Nome da Empresa -->
45     <Field name="HA_NomeBanco" type="CHARACTER" length="30" /> <!-- Nome do Banco -->
46     <Field name="HA_CNAB_2" type="FLOAT" length="10" />
47     <!-- Uso exclusivo FEBRABAN/CNAB - Default 'NULL' -->

```

Figura 33 - Layout de arquivo remessa do SCPREVI.

Após esta configuração, basta apenas buscar o local onde se encontra este *Layout* (local do arquivo XML) e setar cada campo para a classe *Record* do *Texgit*, da mesma forma que foi feita a setagem dos dados para geração do boleto com o *Bopepo*. É importante frisar que para cada campo devem ser informados dois valores: **nome do campo** no *Layout* (que deve ser idêntico ao presente no arquivo XML); **valor para o campo** (que não deverá ultrapassar o limite de tamanho configurado no arquivo XML). Logo, segue abaixo um dos métodos que seta todos os valores descritos na **Figura 38** de acordo com os dados das entidades do SCPREVI e alguns dados padrões descritos no próprio manual da Caixa:

```

118 public Record createHeaderArquivo(FlatFile<Record> ff) {
119
120     Record header = ff.createRecord("Header-Arquivo");
121
122     header.setValue("HA_Banco", 104);
123     header.setValue("HA_Lote", this.sequencialHeader);
124     header.setValue("HA_Registro", 0);
125     header.setValue("HA_CNAB_1", " ");
126     header.setValue("HA_InscTipo", 2);
127     header.setValue("HA_InscNum", this.cedente.getCnpj());
128     header.setValue("HA_UsoExclusivoCAIXA_1", 0);
129     header.setValue("HA_CodIdentAgencia", this.contaBancaria.getNumAgencia());
130     header.setValue("HA_CodIdentAgenciaDig", this.contaBancaria.getDigAgencia());
131     header.setValue("HA_CodIdentCedente", 123456);
132     header.setValue("HA_UsoExclusivoCAIXA_2", 0);
133     header.setValue("HA_UsoExclusivoCAIXA_3", 0);
134     header.setValue("HA_NomeEmpresa", this.cedente.getNome());
135     header.setValue("HA_NomeBanco", this.contaBancaria.getBanco());
136     header.setValue("HA_CNAB_2", " ");
137     header.setValue("HA_CodArq", 1);
138     header.setValue("HA_DtGeracaoArq", this.titulo.getDataDocumento());
139     header.setValue("HA_HrGeracaoArq", this.titulo.getDataDocumento().getTime());
140     header.setValue("HA_NumSeqArq", this.sequencialHeader++);
141     header.setValue("HA_NumVerArq", 050);
142     header.setValue("HA_DensArq", 0);
143     header.setValue("HA_UsoBanco_1", "RETORNO-PRODUÇÃO");
144     header.setValue("HA_UsoEmpresa_1", "REMESSA-PRODUÇÃO");
145     header.setValue("HA_VersaoApp", 1234);
146     header.setValue("HA_CNAB_3", 0);
147     return header;
148
149 }

```

Figura 34 - Método para setar dados de acordo com *layout* XML – SCPREVI.

Após a geração do arquivo remessa, o mesmo poderá ser visualizado no browser pelo usuário. Um exemplo pode ser observado na **Figura A.14**, Anexo A. Portanto, o *Texgit* facilita a vida do desenvolvedor, possibilitando produtividade no desenvolvimento do sistema.

3.2.11 Leitura de Arquivo Retorno

Um dos principais objetivos deste projeto é estar pronto para a comunicação com o banco do cedente, neste caso, a Caixa Econômica Federal. Para tal é necessário não somente gerar o arquivo remessa do boleto para posteriormente enviá-lo ao banco, como também, receber o arquivo retorno enviado pelo banco. Dessa forma, ainda que a comunicação com a aplicação do banco seja um projeto futuro, foi implementada a leitura do arquivo retorno que será recebido pelo SCPREVI, como pode ser observado na **Figura 40**, logo abaixo:

```

82 public String lerArquivoRetorno() throws FileNotFoundException, IOException{
83     // Pega instância do local do arquivo retorno
84     String localArqRetorno = FacesContext.getCurrentInstance().getExternalContext().getRealPath("retorno");
85     // Pega instância do arquivo caso exista (supondo que este arquivo já foi repassado pela app da CAIXA, para este local)
86     File arquivo_retorno = new File(localArqRetorno + "/" + this.titulo.getLocalBoleto() + ".txt");
87     if(arquivo_retorno.exists()){
88         // Cria instância da classe FileReader para obter o arquivo e percorre arquivo com BufferedReader
89         FileReader reader = new FileReader(arquivo_retorno);
90         BufferedReader leitor = new BufferedReader(reader);
91         String linha = null;
92         // busca linha onde se encontra o código do Aceite e testa se foi aceito ou não
93         while(leitor.ready()) {
94             linha = leitor.readLine();
95             String aceite = linha.substring(68, 69);
96             String motivo = linha.substring(69, 71);
97             if (aceite.equals("A")){
98                 this.arqRetorno.setArqRemessa(this.arqRemessa);
99                 this.arqRetorno.setArquivo(arquivo_retorno);
100                 ArquivoRetornoRN rnRetorno = new ArquivoRetornoRN();
101                 rnRetorno.salvar(arqRetorno);
102                 FacesContext context = FacesContext.getCurrentInstance();
103                 FacesMessage msg = new FacesMessage("Arquivo retorno ACEITO.");
104                 context.addMessage(null, msg);
105             } else if (aceite.equals("N")) {
106                 // Busca motivo de rejeição
107                 if (motivo.equals("01")){
108                     FacesContext context = FacesContext.getCurrentInstance();
109                     FacesMessage msg = new FacesMessage(FacesMessage.SEVERITY_ERROR, "Arquivo retorno NÃO ACEITO. "
110                         + "Motivo: Conta Beneficiário Inválida.", null);
111                     context.addMessage(null, msg);
112                 }
113                 if (motivo.equals("02")){
114                     FacesContext context = FacesContext.getCurrentInstance();
115                     FacesMessage msg = new FacesMessage(FacesMessage.SEVERITY_ERROR, "Arquivo retorno NÃO ACEITO. "
116                         + "Motivo: Conta Corrente Inativa para Rateio.", null);
117                     context.addMessage(null, msg);
118                 }
119                 if (motivo.equals("03")){

```

Figura 40 - Método para leitura do Arquivo Retorno – SCPREVI.

Observa-se na imagem, que primeiramente é feita uma busca para conferir se já existe uma instância do arquivo desejado, conforme linhas 84 a 87. Ou seja, supondo que o SCPREVI comunicou-se com a aplicação do banco, logo após o arquivo será salvo na raiz do projeto, no diretório “retorno” conforme linha 84. Caso o arquivo ainda não exista, significa que o banco ainda não implementou o retorno. Logo é mostrada uma mensagem ao usuário informando este fato, como pode ser observado na **Figura A.22**, Anexo A.

Caso o banco tenha enviado o retorno, serão então executadas as linhas após a linha 87, da **Figura 40**. Na linha 89 é criada uma instância para o arquivo retorno utilizando a API Java “*java.io.FileReader*” e, posteriormente, na próxima linha, é criada a instância de “*java.io.BufferedReader*”, outra API do Java disposta para manipulação de arquivos, tornando disponível então, a utilização dos métodos “*read()*” – que ler o arquivo – e “*readLine()*” – que ler uma linha específica – conforme as linhas 93 e 94 respectivamente da **Figura 40**.

Dessa forma, vê-se nas próximas linhas que foram buscados valores fixos de onde encontram-se as informações necessárias, como pode ser observado nas linhas 95 e 96. Assim, cabe uma explicação para tal: a leitura do arquivo foi baseada na especificação **CNAB-240-SIGCB**, contida no documento para geração de arquivo

remessa e retorno do banco Caixa Econômica. Ou seja, para esta implementação, foi gerado um arquivo retorno conforme essa especificação e, de acordo com este arquivo, foi feita a implementação desta leitura. Assim, entende-se que ao receber o arquivo retorno, o SCPREVI executará a leitura da forma correta.

Na linha 97 é feito o teste para conferir se o arquivo foi aceito ou não. Caso não aceito, o SCPREVI busca o motivo, conforme linhas de 105 em diante. Assim percebe-se que a **Figura 40**, não mostra toda a implementação do método. Entretanto, foi feita a conferência para todos os motivos de rejeição, de acordo com o especificado no padrão **CNAB-240-SIGCB**, neste caso, 34 motivos. Assim, será mostrada uma mensagem ao usuário informando que o arquivo não foi aceito e o motivo de rejeição, como pode ser observado na **Figura A.23**, Anexo A. Caso o arquivo tenha sido aceito, o sistema persiste-o no banco de dados e mostra uma mensagem ao usuário informando que o arquivo foi aceito.

3.4 Servidor Web Apache Tomcat

Uma das exigências de uma aplicação com tecnologia *web* é a necessidade de um servidor para que mantenham-se disponíveis os arquivos do sistema. Este servidor é também conhecido como *Servidor Web*. Dessa forma, para que a implantação do sistema seja realizada com sucesso é crucial que esteja instalado e configurado no servidor²⁸, um servidor *web* de qualidade. Assim, todo o desenvolvimento do SCPREVI, e execução de testes locais, foram feitos a partir de um servidor *web* bastante conhecido e aceito pela comunidade de desenvolvedores Java: o *Apache Tomcat*. Neste projeto, foi utilizado a versão **7.0.22**.

O *Apache Tomcat* é um *contêiner Java* e um servidor *web* ao mesmo tempo. Ele suporta a execução das tecnologias *Java Servlet* e *JavaServer Pages* (JSP), o que permite que o Java funcione para ambiente *web*. Embora o *Apache Tomcat* seja robusto o suficiente para ser utilizado em ambiente de produção, ele permite integração com servidores como *Apache HTTP* e *IIS* da *Microsoft*, para fornecer uma capacidade ainda maior de trabalho. (LUCKOW e MELO, 2010, p. 33).

Portanto, durante a implantação do SCPREVI (que será um projeto futuro), pretende-se utilizar este servidor *web* para disponibilizar a aplicação em uma rede local no Previpalmas e, posteriormente, na *Internet*.

²⁸ Neste caso não se trata do servidor *web*, mas sim do servidor físico onde será hospedada a aplicação.

3.5 Ambiente de Desenvolvimento *NetBeans IDE*

O *Netbeans* como o próprio nome já diz, é um IDE (*Integrated Development Environment* ou, em tradução livre, “Ambiente de Desenvolvimento Integrado”), de código aberto (*Open Source*), totalmente gratuito e multiplataforma, desenvolvido pela *Oracle Corporation*. Com ele, o desenvolvimento de sistemas computacionais, sobretudo *softwares* com tecnologia *web*, torna-se totalmente facilitado e produtivo. Esta ferramenta foi utilizada neste projeto para escrever o código do sistema, gozando de todas suas facilidades e organização da aplicação. Através do *NetBeans IDE* a aplicação foi escrita, compilada, depurada e testada.

Um dos recursos mais funcionais do *NetBeans IDE* que foi utilizado para o desenvolvimento da camada de acesso aos dados da aplicação SCPREVI, é a criação de classes de entidade de banco de dados. Este recurso possibilita ao desenvolvedor, após ter criado uma conexão com o banco de dados no IDE, e tendo em vista que o banco já está criado com todas as tabelas e relacionamentos, criar as classes de domínio do sistema.

Em outras palavras, o IDE transforma as tabelas do banco, em classes POJO, possibilitando ainda, que o desenvolvedor escolha qual especificação e framework deseja para gerar as classes anotadas. Neste caso, como já descrito, utilizou-se o *framework Hibernate* com especificação JPA. É um recurso muito interessante que resultou em alta produtividade de desenvolvimento do sistema.

4 CONCLUSÕES E PROJETOS FUTUROS

4.1 CONCLUSÕES

Durante o desenvolvimento deste projeto, foram confirmadas algumas suposições descritas ainda no Pré-Projeto de qualificação. Uma das expectativas alcançadas foi a alta produtividade garantida através da utilização do *framework* de desenvolvimento *web* *JavaServer Faces*, juntamente com sua biblioteca de componentes *PrimeFaces*. A linguagem *Java* é conhecida por sua dificuldade de utilização, sobretudo quando se trata de aplicações *web*. Entretanto, isto não passa de um mito. Atualmente existem vários *frameworks* de desenvolvimento que surgiram justamente com o objetivo de mudar esta realidade.

E realmente o fazem. Com o *PrimeFaces*, por exemplo, ganhou-se tempo, sobretudo em tarefas básicas, como implementar o CRUD²⁹ das entidades na camada de visualização. A união dessas ferramentas com o *framework* *Hibernate* tornou ainda mais produtivo o processo de desenvolvimento, pois este último desvincula o desenvolvedor dos papéis básicos do processo, como por exemplo, as transações com o banco de dados.

Outras tecnologias que ajudaram bastante no alto nível de produtividade foram os componentes do projeto *JRimum*. O *Bopepo* é uma ferramenta incrivelmente fácil de utilizar, sendo possível sua configuração e implementação sem muitos mistérios, o que foi descrito neste projeto. Atrelado a isto, encontra-se outro componente do *JRimum*, o *Texgit*, que também foi utilizado com sucesso no projeto. Assim, entende-se que estas ferramentas foram criadas com o objetivo de simplesmente tornar mais rápido o desenvolvimento do sistema, fazendo com que o desenvolvedor não se preocupe com as partes mais básicas e trabalhosas.

4.2 PROJETOS FUTUROS

Não menos importante, foram criadas idéias durante este projeto com o intuito de implementá-las futuramente. Durante o processo de implantação, que será um projeto futuro, para facilitar a integração dos dados antigos do *Previpalmas* com o sistema, têm-se a idéia de desenvolver um *Importador*. Dessa forma, seria possível

²⁹ *Create, Read, Update and Delete*, ou, em tradução livre, “Inserir, Visualizar, Atualizar e Excluir”.

realizar a importação de todos os dados já existentes através de arquivos de texto, com algumas padronizações exigidas. Assim, o processo de implantação seria mais rápido, pois com o importador, o usuário não teria a necessidade de proceder com todos os cadastros de todos os servidores no sistema.

Este importador teria apenas a necessidade de conexão com o banco de dados da aplicação como particularidade. A idéia é implementar este importador de modo que execute todos os processos de validações da integridade do banco de dados, e caso ocorra algum erro, mostrá-lo ao usuário, assim como os erros de formatação como, por exemplo, quando o usuário informar uma quantidade de campos diferente da exigida. O importador, por se tratar de uma ferramenta específica de sistemas, seria manuseado pelos funcionários do departamento de *Informática* do Instituto.

Aliada a isto, existe também a idéia de desenvolvimento de uma personalização dos boletos gerados pelo SCPREVI. Como descrito neste projeto, o *Bopepo* apresenta uma característica extremamente funcional, que é a personalização de determinados campos do boleto. Assim, o desenvolvedor poderá criar campos, retirar campos, de acordo com a necessidade imposta pelo Instituto. Dessa forma, desenvolveu-se esta idéia, pois, durante a implantação certamente serão buscadas informações para mudanças, pois o sistema será apresentado para os próprios funcionários do Previpalmas, os chamados *usuários finais*.

REFERÊNCIAS

BRASIL. **Lei nº 9.796 de 05 de Maio de 1999**. Dispõe sobre a compensação financeira entre o Regime Geral de Previdência Social e os regimes de previdência dos servidores da União, dos Estados, do Distrito Federal e dos Municípios, nos casos de contagem recíproca de tempo de contribuição para efeito de aposentadoria, e dá outras providências. Diário Oficial da União, Brasília, DF, nº 247, dez., 2006.

DEITEL, H. M.; DEITEL, P. J. **Java Como Programar**. Trad. Edson Furnankiewicz. 6ª Edição. Porto Alegre: Bookman, 2005.

DIAS, Eduardo Rocha e MACÊDO, José L. M. **Curso de Direito Previdenciário**. 3ª ed. São Paulo: Forense, 2010.

Hibernate Community Documentation. **HIBERNATE – Relational Persistence for Idiomatic Java**. Disponível em: <<http://www.docs.jboss.org/hibernate/stable/core/reference/en/html>>. Acesso em 22 nov. 2011.

JANDL, Peter Jr. **Java 6 – Guia de Consulta Rápida**. São Paulo: Novatec, 2008.

LARMAN, Craig. **Utilizando UML e Padrões** – uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo. (Prefácio de Philippe Kruchten). 3ª Edição. Porto Alegre: Bookman, 2007.

LUCKOW, Décio Heinzelmann e MELO, Alexandre Altair. **Programação Java para Web** – Aprenda a desenvolver uma aplicação financeira pessoal com as ferramentas mais modernas da plataforma Java. São Paulo: Novatec, 2010.

PALMAS, Tocantins. **Lei nº 1.414 de 29 de dezembro de 2005 de Palmas**. Dispõe sobre o regime próprio de previdência social do município e sobre a entidade de previdência e dá outras providências.

PRESSMAN, Roger S. **Engenharia de Software**: uma abordagem profissional. 7ª ed. Porto Alegre: AMGH, 2011.

PRIME, Optimus. **PRIMEFACES** – *User's Guide*. Disponível em *PrimeFaces Documentation*: <<http://www.primefaces.org/documentation.html>>. Acesso em: 20 nov. 2011.

SILVA, Maurício S. **AJAX com JQuery** – Requisições AJAX com a simplicidade de JQuery. São Paulo: Novatec, 2009.

SILVA, Maurício S. **JavaScript** – Guia do Programador. São Paulo: Novatec, 2010.

SOMMERVILLE, Ian. **Engenharia de Software**. 8ª ed. São Paulo: Pearson Addison-Wesley, 2007.

PROJETO JRIMUM. **Bopepo** – Documentação. Disponível em: <<http://jrium.org/bopepo/wiki/Componente/Documentacao>>. Acesso em: 02 jan. 2012.

PROJETO JRIMUM. **Texgit** – Documentação. Disponível em: <<http://jrium.org/texgit/wiki/Componente/Download>>. Acesso em: 02 fev. 2012.

ANEXO A – Telas do Caso de Uso Manter Cobrança de Servidor SCPREVI

A.1 TELA 01 – SCPREVI-CDU-MCSERV

SCPREVI
Sistema de Cobrança Previdenciário

Bem vindo ao SCPREVI

Com o SCPREVI, o PREVIPALMAS garante aos seus segurados maior segurança em suas contribuições, a partir das seguintes características:

- Controle de contribuições de servidores;
- Geração de boletos de suas contribuições;
- Geração de arquivo remessa dos boletos;

Informe seus dados

Login: zezo.silva

Senha: ••••••

Entrar Esqueceu sua senha?

Instituto de Previdência Social do Município de Palmas
PREVIPALMAS © 2011 Telefone: (63) 3218-5050 - Fax: (63) 3218-5042

Figura A.1: Processo de autenticação no SCPREVI.

A.2 TELA 02 – SCPREVI-CDU-MCSERV

SCPREVI
Sistema de Cobrança Previdenciário

► Proprietários ► Servidores LIP ► Servidores Cedidos ► Servidores Exonerados ► Usuários ► Cedentes ► Contas Bancárias ► Arquivo Remessa ► Arquivo Retorno ► Sair

Bem vindo ao SCPREVI. Veja abaixo, algumas descrições de atividades:

Cadastro de Proprietários

Para efetuar o cadastro de Proprietários, siga os seguintes passos:

- Vá ao menu horizontal e clique na opção "Proprietários";
- Escolha no menu vertical à esquerda, a opção "Cadastrar Novo";
- Informe os dados do proprietário e atente-se às seguintes validações:
 - Os campos "Nome" e "Sobrenome" são obrigatórios;
 - O campo "E-mail" é obrigatório e deve conter um e-mail válido;
 - O campo "CPF" deve conter um cpf válido.
- Clique no botão "Confirmar".

Cadastro de Servidores

Atualização Monetária

Exportação de Relatórios de Servidores Exonerados

Instituto de Previdência Social do Município de Palmas
PREVIPALMAS © 2011 Telefone: (63) 3218-5050 - Fax: (63) 3218-5042

Figura A.2: Tela principal do SCPREVI para usuário com permissão de administrador.

A.3 TELA 03 – SCPREVI-CDU-MCSERV



SCPREVI
Sistema de Cobrança Previdenciário

Proprietários > Servidores LIP > Servidores Cedidos > Servidores Exonerados > Usuários > Cedentes > Contas Bancárias > Arquivo Remessa > Arquivo Retorno > Sair

Operações

- ✓ Servidores LIP
- ✓ Cadastrar Novo
- ✓ Listar Usuários
- ✓ Atualização Monetária
- ✓ Extrato de Sacado
- ✓ Listar Boletos
- ✓ Gerar Arquivo Remessa
- ✓ Arquivo Retorno

Listagem de Servidores LIP

(1 of 2) 1 2 5

Nome	E-mail	CPF	Opções
Jonerval	jonerval@gmail.com	222.222.222-22	[Edit] [Delete] [Print] [Visualizar]
Silas	silas@gmail.com	777.777.777-77	[Edit] [Delete] [Print] [Visualizar]
João	joao@gmail.com	333.333.333-33	[Edit] [Delete] [Print] [Visualizar]
Jamal	jamal@gmail.com	555.555.555-55	[Edit] [Delete] [Print] [Visualizar]
José	jose@hotmail.com	666.666.666-66	[Edit] [Delete] [Print] [Visualizar]

(1 of 2) 1 2 5

Instituto de Previdência Social do Município de Palmas
PREVIPALMAS © 2011 Telefone: (63) 3218-5050 - Fax: (63) 3218-5042

Figura A.3: Tela principal da área de servidores LIP do SCPREVI.

A.4 TELA 04 – SCPREVI-CDU-MCSERV



SCPREVI
Sistema de Cobrança Previdenciário

Proprietários > Servidores LIP > Servidores Cedidos > Servidores Exonerados > Usuários > Cedentes > Contas Bancárias > Arquivo Remessa > Arquivo Retorno > Sair

Operações

- ✓ Servidores LIP
- ✓ Cadastrar Novo
- ✓ Listar Usuários
- ✓ Atualização Monetária
- ✓ Extrato de Sacado
- ✓ Listar Boletos
- ✓ Gerar Arquivo Remessa
- ✓ Arquivo Retorno

Atualização Monetária [Servidores LIP]

Listagem de Servidores LIP

(1 of 2) 1 2 5

Nome	E-mail	CPF	Opções
Jonerval	jonerval@gmail.com	222.222.222-22	[Boleto]
Silas	silas@gmail.com	777.777.777-77	[Boleto]
João	joao@gmail.com	333.333.333-33	[Boleto]
Jamal	jamal@gmail.com	555.555.555-55	[Atualização Monetária]
José	jose@hotmail.com	666.666.666-66	[Boleto]

(1 of 2) 1 2 5

Instituto de Previdência Social do Município de Palmas
PREVIPALMAS © 2011 Telefone: (63) 3218-5050 - Fax: (63) 3218-5042

Figura A.4: Atualização Monetária de Servidores LIP do SCPREVI.

A.5 TELA 05 – SCPREVI-CDU-MCSERV

Operações

- ✓ Servidores LIP
- ✓ Cadastrar Novo
- ✓ Listar Usuários
- ✓ Atualização Monetária
- ✓ Extrato de Sacado
- ✓ Listar Boleto
- ✓ Gerar Arquivo Remessa
- ✓ Arquivo Retorno

Atualização Monetária

Título/Boleto

Nosso número: 463768786786786 Digito: 5

Número documento: 454654 Valor Original: 400

Valor desconto: 0 Valor dedução: 0

Valor Mora (%): 2 Valor Acréscimo: 0

Data Documento: 04.02.2012 Data Vencimento: 02.03.2012

Local pagamento: qualquer banco/lotérica Mês: Fevereiro

Cedente/Conta

Cedente: Previpalmas

Conta Bancária: 22222

Instruções

Instrução ao sacado: Não receber após vencimento. Instrução: Outras informações.

Instrução: Juros de 2% a.m. Instrução: Outras informações.

Instrução: Outras informações. Instrução: Outras informações.

Instrução: Outras informações. Instrução: Outras informações.

Instrução: Outras informações. Instrução: Outras informações.

Confirmar

Figura A.5: Formulário para geração de boleto do SCPREVI.

A.6 TELA 06 – SCPREVI-CDU-MCSERV

Figura A.6: Download de boleto do SCPREVI no Browser.

Figura A.8: Visualização de Boleto do SCPREVI no *Browser*.

A.9 TELA 09 – SCPREVI-CDU-MCSERV



SCPREVI
Sistema de Cobrança Previdenciário

Listagem de Boletos [Servidores LIP]

Proprietários > Servidores LIP > Servidores Cedidos > Servidores Exonerados > Usuários > Cedentes > Contas Bancárias > Arquivo Remessa > Arquivo Retorno > Sair

Lista de todos os Boletos de Servidores LIP

Exclusão de boleto

Tem certeza que deseja excluir o boleto?
Essa operação não poderá ser desfeita.
Confirma?

Confirmar Cancelar

Servidor	Boleto	Mês	Valor Original	Data Vencimento	Situação	Arquivo_Boleto	Opções
Jonerval	454654	Outubro	400.0	02/11/2011	Pendente	Visualizar	[Editar] [Excluir]
Jonerval	454654	Fevereiro	600.0	02/03/2012	Pago	Visualizar	[Editar] [Excluir]
Jonerval	454654	Março	500.0	02/04/2012	Pendente	Visualizar	[Editar] [Excluir]
Silas	454654	Fevereiro	400.0	04/02/2012	Pendente	Visualizar	[Editar] [Excluir]
Silas	454654	Fevereiro	400.0	04/02/2012	Pendente	Visualizar	[Editar] [Excluir]

(1 of 2) [1] [2] 5

Instituto de Previdência Social do Município de Palmas
PREVIPALMAS © 2011 Telefone: (63) 3218-5050 - Fax: (63) 3218-5042

Figura A.9: Solicitação de confirmação de exclusão de boletos do SCPREVI.

A.10 TELA 10 – SCPREVI-CDU-MCSERV



SCPREVI
Sistema de Cobrança Previdenciário

Extrato do Sacado [Servidores LIP]

Proprietários > Servidores LIP > Servidores Cedidos > Servidores Exonerados > Usuários > Cedentes > Contas Bancárias > Arquivo Remessa > Arquivo Retorno > Sair

Operações

- ✓ Servidores Cedidos
- ✓ Cadastrar Novo
- ✓ Listar Usuários
- ✓ Atualização Monetária
- ✓ Extrato de Sacado
- ✓ Listar Boletos
- ✓ Gerar Arquivo Remessa
- ✓ Arquivo Retorno

Listagem de Servidores LIP

(1 of 2) [1] [2] 5

Nome	E-mail	CPF	Opções
Jonerval	jonerval@gmail.com	222.222.222-22	[Editar] [Excluir]
Silas	silas@gmail.com	777.777.777-77	[Editar] [Excluir]
João	joao@gmail.com	333.333.333-33	[Editar] [Excluir]
Jamal	jamal@gmail.com	555.555.555-55	[Editar] [Excluir]
José	jose@hotmail.com	666.666.666-66	[Editar] [Excluir]

(1 of 2) [1] [2] 5

Instituto de Previdência Social do Município de Palmas
PREVIPALMAS © 2011 Telefone: (63) 3218-5050 - Fax: (63) 3218-5042

Figura A.10: Extrato de Boletos de Servidor LIP do SCPREVI.

A.11 TELA 11 – SCPREVI-CDU-MCSERV

SCPREVI
Sistema de Cobrança Previdenciário

Extrato de Sacado [Servidores LIP]

Proprietários ▸ Servidores LIP ▸ Servidores Cedidos ▸ Servidores Exonerados ▸ Usuários ▸ Cedentes ▸ Contas Bancárias ▸ Arquivo Remessa ▸ Arquivo Retorno ▸ Sair

Lista de todos os boletos por servidor
(1 of 1) 1 5

Sacado	Boleto	Mês	Valor Original	Multa	Juros (am)	Valor Cobrado	Data Emissão	Data Vencimento	Situação
Jonerval	454654	Outubro	400.0	2.0	410.0	410.0	04/10/2011	02/11/2011	Pendente
Jonerval	454654	Fevereiro	600.0	2.0	615.0	615.0	04/02/2012	02/03/2012	Pago
Jonerval	454654	Março	500.0	2.0	512.5	512.5	04/03/2012	02/04/2012	Pendente

(1 of 1) 1 5

Exportar todos Exportar por página

Instituto de Previdência Social do Município de Palmas
PREVIPALMAS © 2011 Telefone: (63) 3218-5050 - Fax: (63) 3218-5042

Figura A.11: Listagem de todos os títulos por servidor do SCPREVI.

A.12 TELA 12 – SCPREVI-CDU-MCSERV

SCPREVI
Sistema de Cobrança Previdenciário

Geração Arquivo Remessa [Servidores LIP]

Proprietários ▸ Servidores LIP ▸ Servidores Cedidos ▸ Servidores Exonerados ▸ Usuários ▸ Cedentes ▸ Contas Bancárias ▸ Arquivo Remessa ▸ Arquivo Retorno ▸ Sair

Lista de todos os Boletos de Servidores LIP
(1 of 2) 1 2 5

Sacado	Boleto	Mês	Valor Original	Multa	Juros (am)	Valor Cobrado	Data Emissão	Data Vencimento	Situação	Arquivo_Boleto	Opções
Jonerval	454654	Outubro	400.0	2.0	410.0	410.0	04/10/2011	02/11/2011	Pendente	Visualizar	Visualizar
Jonerval	454654	Fevereiro	600.0	2.0	615.0	615.0	04/02/2012	02/03/2012	Pago	Visualizar	Visualizar
Jonerval	454654	Março	500.0	2.0	512.5	512.5	04/03/2012	02/04/2012	Pendente	Visualizar	Gerar Arquivo Remessa
Silas	454654	Fevereiro	400.0	2.0	410.0	410.0	04/02/2012	02/03/2012	Pendente	Visualizar	Visualizar
Silas	454654	Fevereiro	400.0	2.0	410.0	410.0	04/02/2012	02/03/2012	Pendente	Visualizar	Visualizar

(1 of 2) 1 2 5

Instituto de Previdência Social do Município de Palmas
PREVIPALMAS © 2011 Telefone: (63) 3218-5050 - Fax: (63) 3218-5042

Figura A.12: Listagem de títulos de servidor para geração de arquivo remessa do SCPREVI.

A.13 TELA 13 – SCPREVI-CDU-MCSERV

SCPREVI
Sistema de Cobrança Previdenciário

Listagem de Arquivos Remessa

Proprietários > Servidores LIP > Servidores Cedidos > Servidores Exonerados > Usuários > Cedentes > Contas Bancárias > Arquivo Remessa > Arquivo Retorno > Sair

Operação realizada com sucesso.

Módulo Arquivo Remessa

(1 of 1) 1 5

Sacado	Mês	Arquivo Remessa	Opções
Jonerval	Outubro	Visualizar	
Jonerval	Fevereiro	Visualizar	
Silas	Fevereiro	Visualizar	

(1 of 1) 1 5

Instituto de Previdência Social do Município de Palmas
PREVIPALMAS © 2011 Telefone: (63) 3218-5050 - Fax: (63) 3218-5042

Figura A.13: Mensagem de sucesso e listagem de arquivos remessa cadastrados no SCPREVI.

A.14 TELA 14 – SCPREVI-CDU-MCSERV

localhost:8084/SCPREVI_1.3/remessa/Jonerval_boleto_Fevereiro.pdf.txt

011041	0	205.278.848/0001-090	2222221234560	0Previpalmas	Caixa Econômica Federal
104/02/201213283244000001	40	0	RETORNO-PRODUÇÃO	REMESSA-PRODUÇÃO	12340
021042	1R1	0 24 0205.278.848/0001-091234560	22222222222	12345670Previpalmas	
0	04/02/201204/02/2012123456				
03104999932	100	22222222222	0	0 1 1	111230
0	0	0	0	02/03/2012615.0	22222219104/02/2012204/02/20121
04104999932	000	1222.222.222-22	Jonerval	15 05	0 0
05104999950	9999	9999 615.0	0	463768786786786	Al. 09
06104999990	9999	9999 0	0	9999 615.0	0

Plano Diretor Sul 89865-686 0 Palmas

Figura A.14: Visualização de arquivo remessa do SCPREVI no Browser.

A.15 TELA 15 – SCPREVI-CDU-MCSERV

SCPREVI
Sistema de Cobrança Previdenciário

Listagem de Arquivos Remessa

Proprietários > Servidores LIP > Servidores Cedidos > Servidores Exonerados > Usuários > Cedentes > Contas Bancárias > Arquivo Remessa > Arquivo Retorno > Sair

Exclusão de arquivo

Tem certeza que deseja excluir o arquivo remessa? ATENÇÃO: Ao excluir este arquivo remessa, também serão excluídos seus arquivos retorno referentes. Essa operação não poderá ser desfeita. Confirma?

Confirmar Cancelar

Sacado	Mês	Arquivo Remessa	Opções
Jonerval		Visualizar	
Jonerval		Visualizar	
Silas		Visualizar	

(1 of 1) 1 5

Instituto de Previdência Social do Município de Palmas
PREVIPALMAS © 2011 Telefone: (63) 3218-5050 - Fax: (63) 3218-5042

Figura A.15: Solicitação de confirmação de exclusão de arquivo remessa do SCPREVI.

A.16 TELA 16 – SCPREVI-CDU-MCSERV

SCPREVI
Sistema de Cobrança Previdenciário

Módulo Arquivo Retorno [Servidores LIP]

Proprietários ▸ Servidores LIP ▸ Servidores Cedidos ▸ Servidores Exonerados ▸ Usuários ▸ Cedentes ▸ Contas Bancárias ▸ Arquivo Remessa ▸ Arquivo Retorno ▸ Sair

Lista de todos os Arquivos Remessa de Servidores LIP

(1 of 1) 1 5

Sacado	Mês	Arquivo Remessa	Opções
Jonerval	Outubro	Visualizar	
Jonerval	Fevereiro	Visualizar	
Sílas	Fevereiro	Visualizar	

(1 of 1) 1 5

Instituto de Previdência Social do Município de Palmas
PREVIPALMAS © 2011 Telefone: (63) 3218-5050 - Fax: (63) 3218-5042

Figura A.16: Listagem de arquivos remessa para geração de arquivo retorno do SCPREVI.

A.17 TELA 17 – SCPREVI-CDU-MCSERV

SCPREVI
Sistema de Cobrança Previdenciário

Proprietários ▸ Servidores LIP ▸ Servidores Cedidos ▸ Servidores Exonerados ▸ Usuários ▸ Cedentes ▸ Contas Bancárias ▸ Arquivo Remessa ▸ Arquivo Retorno ▸ Sair

Operações

- ✓ Cedentes
- ✓ Cadastrar Novo Cedente
- ✓ Contas Bancárias
- ✓ Cadastrar Nova Conta

Módulo Cedentes

(1 of 1) 1 5

Nome	CNPJ	Opções
Previpalmas	05.278.848/0001-09	

(1 of 1) 1 5

Instituto de Previdência Social do Município de Palmas
PREVIPALMAS © 2011 Telefone: (63) 3218-5050 - Fax: (63) 3218-5042

Figura A.17: Módulo Cedente do SCPREVI.

A.18 TELA 18 – SCPREVI-CDU-MCSERV



The screenshot shows the 'Módulo Cedentes' (Contributors Module) of the SCPREVI system. The header includes the SCPREVI logo and the text 'Sistema de Cobrança Previdenciário'. A navigation bar at the top lists various user roles. The main form area is titled 'Cadastro de Cedente' and contains a section labeled 'Dados Cedente'. This section has two input fields: 'Nome Cedente' with the value 'Previpalmas' and 'CNPJ' with a masked input field. A 'Confirmar' button is located at the bottom of the form. The footer contains the institution's name, address, and contact information.

SCPREVI
Sistema de Cobrança Previdenciário

Módulo Cedentes

Proprietários ▸ Servidores LIP ▸ Servidores Cedidos ▸ Servidores Exonerados ▸ Usuários ▸ Cedentes ▸ Contas Bancárias ▸ Arquivo Remessa ▸ Arquivo Retorno ▸ Sair

Cadastro de Cedente

Dados Cedente

Nome Cedente: Previpalmas

CNPJ: [Masked Input]

Confirmar

Instituto de Previdência Social do Município de Palmas
PREVIPALMAS © 2011 Telefone: (63) 3218-5050 - Fax: (63) 3218-5042

Figura A.18: Formulário de cadastro de novo cedente do SCPREVI.

A.19 TELA 19 – SCPREVI-CDU-MCSERV



The screenshot shows the 'Módulo Contas Bancárias' (Bank Accounts Module) of the SCPREVI system. The header is identical to the previous screen. The navigation bar includes 'Contas Bancárias'. The main form area is titled 'Cadastro de Conta Bancária' and contains a section labeled 'Dados Conta'. This section has several input fields: 'Cedente' (a dropdown menu with 'Previpalmas' selected), 'Número Conta' (with '555' entered), 'Número Agência', and 'Banco'. There are also 'Digito' (digit) input fields next to the account and agency numbers. A 'Confirmar' button is at the bottom. The footer is the same as the previous screen.

SCPREVI
Sistema de Cobrança Previdenciário

Módulo Contas Bancárias

Proprietários ▸ Servidores LIP ▸ Servidores Cedidos ▸ Servidores Exonerados ▸ Usuários ▸ Cedentes ▸ Contas Bancárias ▸ Arquivo Remessa ▸ Arquivo Retorno ▸ Sair

Cadastro de Conta Bancária

Dados Conta

Cedente: Previpalmas

Número Conta: 555 Digito: [Input]

Número Agência: [Input] Digito: [Input]

Banco: [Input]

Confirmar

Instituto de Previdência Social do Município de Palmas
PREVIPALMAS © 2011 Telefone: (63) 3218-5050 - Fax: (63) 3218-5042

Figura A.19: Formulário de cadastro de nova conta bancária de cedente do SCPREVI.

A.20 TELA 20 – SCPREVI-CDU-MCSERV

Figura A.20: Validação de campos do formulário de geração de boleto do SCPREVI.

A.21 TELA 21 – SCPREVI-CDU-MCSERV

Servidor	Boleto	Mês	Valor Original	Multa (%)	Valor Cobrado	Data Emissão	Data Vencimento	Situação	Arquivo_Boleto	Opções
Jonerval	454654	Outubro	400.0	2.0	410.0	04/10/2011	02/11/2011	Pendente	Visualizar	
Jonerval	454654	Fevereiro	600.0	2.0	615.0	04/02/2012	02/03/2012	Pago	Visualizar	
Jonerval	454654	Março	500.0	2.0	512.5	04/03/2012	02/04/2012	Pendente	Visualizar	
Silas	454654	Fevereiro	400.0	2.0	410.0	04/02/2012	02/03/2012	Pendente	Visualizar	
Silas	454654	Fevereiro	400.0	2.0	410.0	04/02/2012	02/03/2012	Pendente	Visualizar	

Figura A.21: Validação de regra de negócio RN06 do SCPREVI.

A.22 TELA 22 – SCPREVI-CDU-MCSERV

The screenshot shows the SCPREVI system interface. At the top, there is a logo with a dollar sign and the text "SCPREVI Sistema de Cobrança Previdenciário". To the right, the title "Arquivos Retorno" is displayed. Below the header, a navigation menu includes: Proprietários, Servidores LIP, Servidores Cedidos, Servidores Exonerados, Usuários, Cedentes, Contas Bancárias, Arquivo Remessa, Arquivo Retorno, and Sair. A yellow message box states: "Ainda não existe um retorno do banco para este título." Below this, a table titled "Módulo Arquivo Retorno" is shown. The table has columns for "Sacado", "Mês", "Arquivo Retorno", and "Opções". The "Mês" column contains the value "5". The table indicates "(1 of 1)" records. Below the table, it says "Nenhum registro encontrado." and "(1 of 1)". At the bottom, the footer text reads: "Instituto de Previdência Social do Município de Palmas PREVIPALMAS © 2011 Telefone: (63) 3218-5050 - Fax: (63) 3218-5042".

Figura A.22: Mensagem ao usuário do SCPREVI caso o arquivo retorno do boleto ainda não exista.

A.23 TELA 23 – SCPREVI-CDU-MCSERV

The screenshot shows the SCPREVI system interface. At the top, there is a logo with a dollar sign and the text "SCPREVI Sistema de Cobrança Previdenciário". To the right, the title "Arquivos Retorno" is displayed. Below the header, a navigation menu includes: Proprietários, Servidores LIP, Servidores Cedidos, Servidores Exonerados, Usuários, Cedentes, Contas Bancárias, Arquivo Remessa, Arquivo Retorno, and Sair. A red message box states: "Arquivo retorno NÃO ACEITO. Motivo: Rateio Não Efetuado, Conta Débito Cedente Bloqueada." Below this, a table titled "Módulo Arquivo Retorno" is shown. The table has columns for "Sacado", "Mês", "Arquivo Retorno", and "Opções". The "Mês" column contains the value "5". The table indicates "(1 of 1)" records. Below the table, it says "Nenhum registro encontrado." and "(1 of 1)". At the bottom, the footer text reads: "Instituto de Previdência Social do Município de Palmas PREVIPALMAS © 2011 Telefone: (63) 3218-5050 - Fax: (63) 3218-5042".

Figura A.23: Mensagem ao usuário do SCPREVI caso arquivo retorno não tenha sido aceito.

ANEXO B – Diagramas do SCPREVI

B.1 Diagrama de Casos de Uso SCPREVI

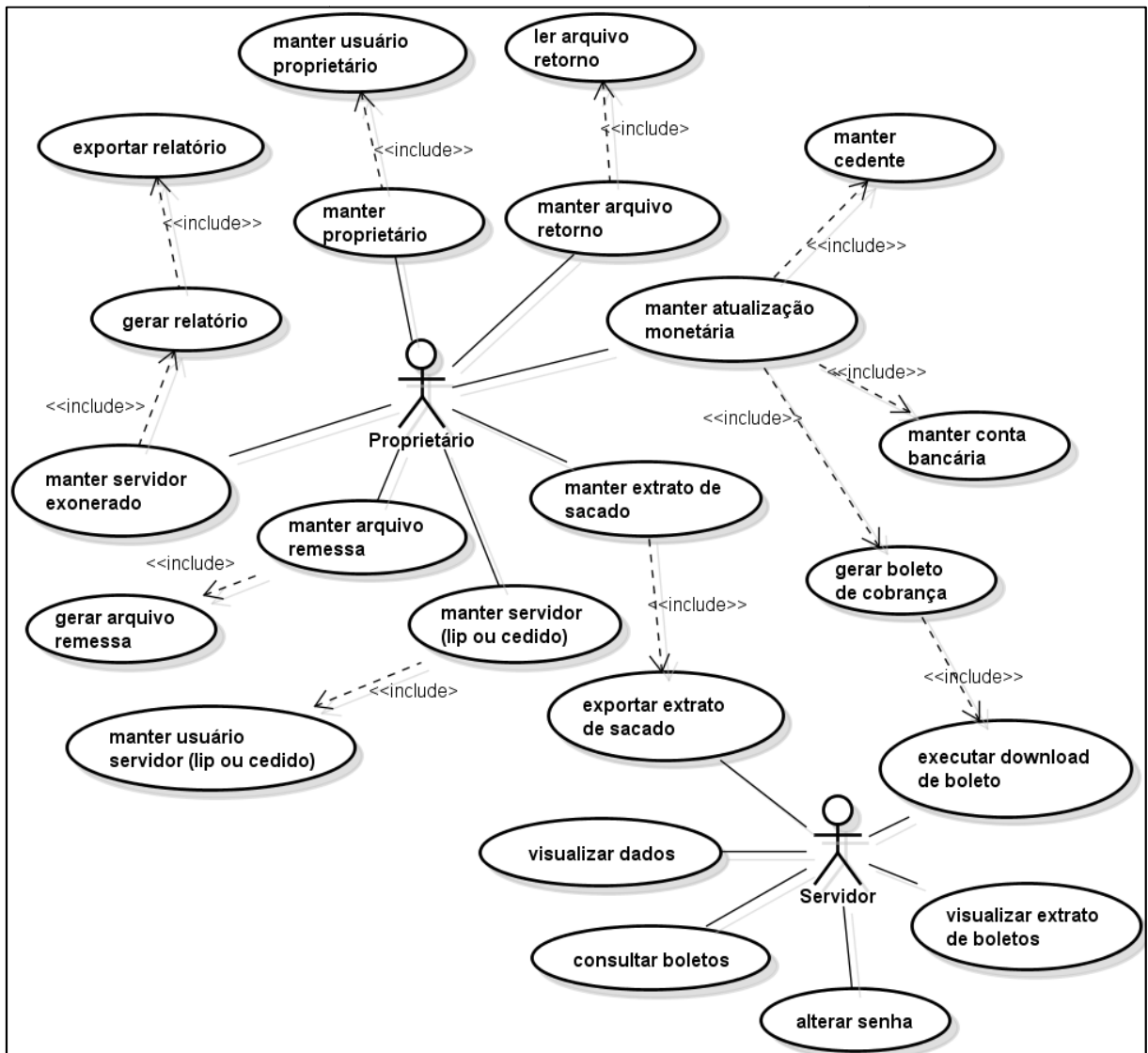


Figura B.1 - Diagrama de Caso de Uso SCPREVI.

B.2 Diagrama de Atividades SCPREVI

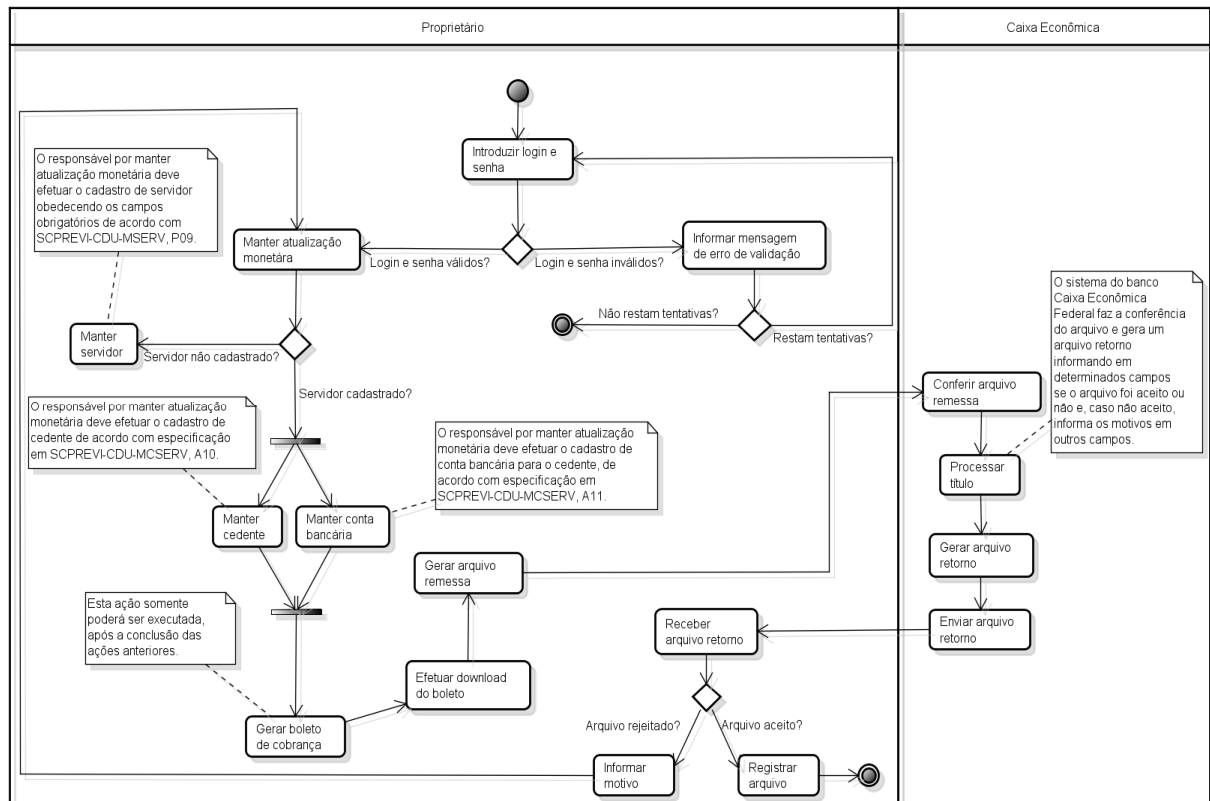


Figura B.2 - Diagrama de Atividades – SCPREVI - Manter Cobrança de Servidor.

B.3 Diagrama de Classes de Domínio SCPREVI

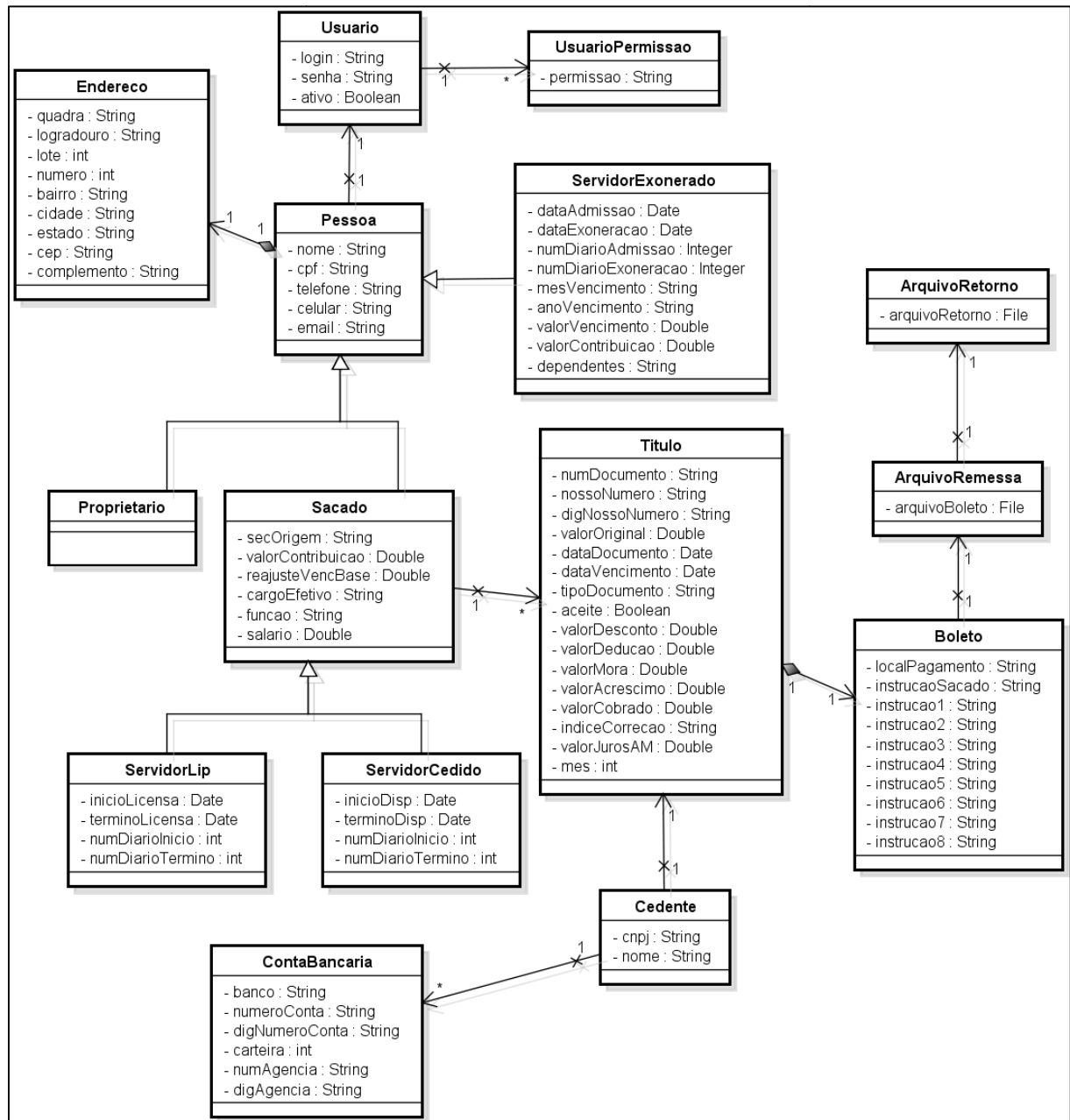


Figura B.3 - Diagrama de Classes de Domínio SCPREVI.

B.4 Diagrama de Sequência SCPREVI I

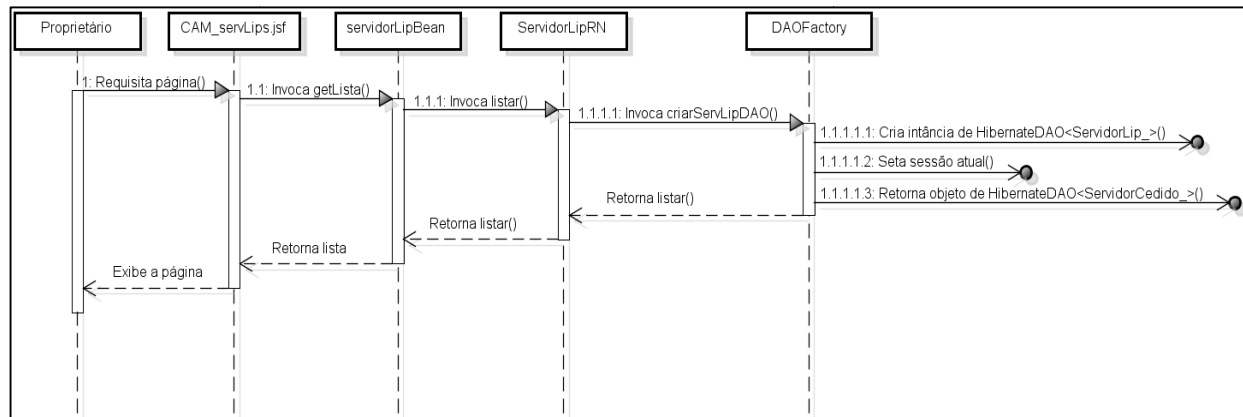


Figura B.4 - Diagrama de Sequência – SCPREVI – Gerar Boleto – P01.

B.5 Diagrama de Sequência SCPREVI II

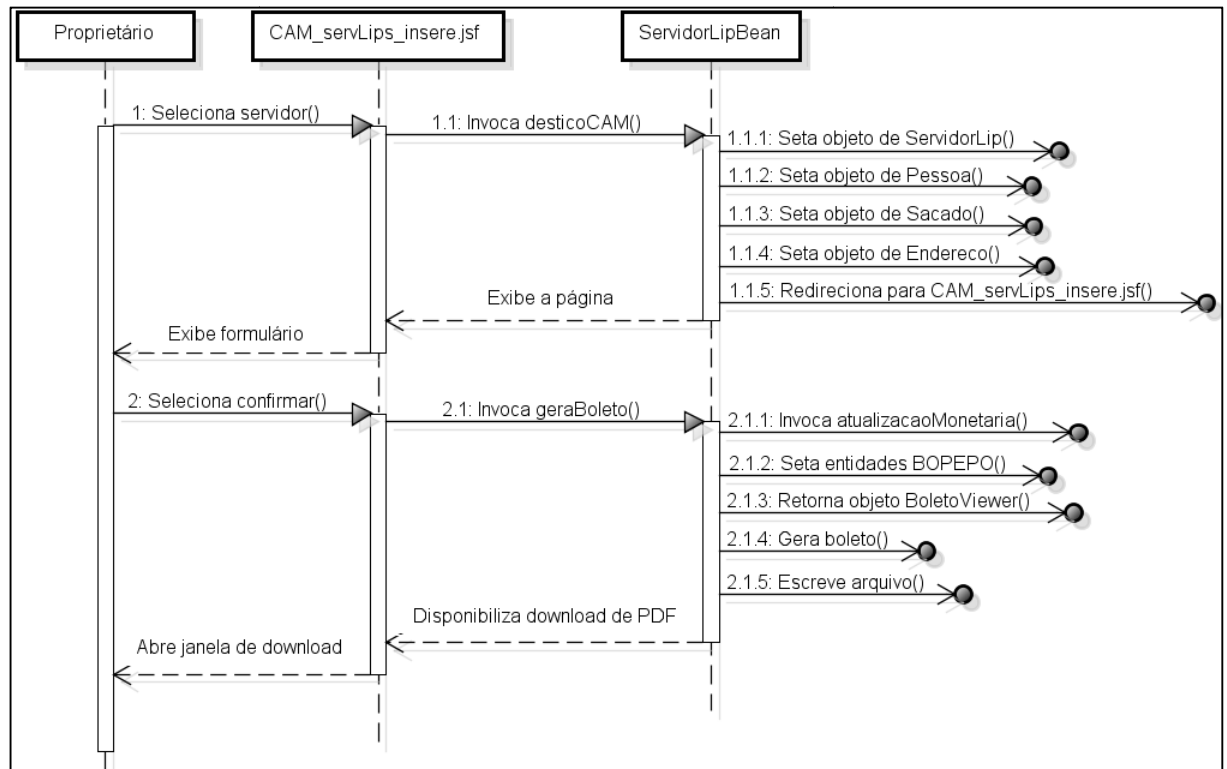


Figura B.5 - Diagrama de Sequência – SCPREVI – Gerar Boleto – P02.