CRISTOPHER MARCELINO SANCHES

ESPECIFICAÇÃO DO SISTEMA DE CONTRATOS E CONVÊNIOS E DO SISTEMA PARA A UNIÃO

JOÃO PESSOA
2005

CRISTOPHER MARCELINO SANCHES

ESPECIFICAÇÃO DO SISTEMA DE CONTRATOS E CONVÊNIOS E DO SISTEMA PARA A UNIÃO

Relatório de estágio apresentado à coordenação do curso de Ciências da Computação do Centro Universitário de João Pessoa – UNIPÊ, como exigência para a obtenção do titulo de Bacharel em Ciências da Computação, sob orientação do Profº. Ms. Tiago Lima Massoni.

JOÃO PESSOA
2005

FOLHA DE APROVAÇÃO

CRISTOPHER MARCELINO SANCHES

ESPECIFICAÇÃO DO SISTEMA DE CONTRATOS E CONVÊNIOS E DO SISTEMA PARA A UNIÃO

Relatório de estágio apresentado ao curso de Ciências da Computação do UNIPÊ – Centro Universitário de João Pessoa, como pré-requisito para obtenção do grau de bacharel pela Banca Examinadora composta pelos membros:

( ) Aprovado

( ) Reprovado

Data: 06/12/2005

_____

Profº. Ms. Tiago Lima Massoni
(Orientador)

_____

Profº. Ms. Thiago José Marques Moura
(Membro)

_____

Profº. Ms. César Rocha Vasconcelos
(Membro)

OBS.:

Dedico este trabalho a todas as pessoas que diretamente ou indiretamente contribuíram para sua realização.

Agradeço a minha família, pelo incentivo que sempre me deram. Ao meu orientado Tiago Lima Massoni pela paciência e orientação. Aos meus colegas, professores e a todos que colaboraram para a conclusão deste trabalho.

"Os homens pensam que possuem uma mente,
mas é a mente que os possui".
Bob Marley

SANCHES, Cristopher Marcelino. **Especificação do Sistema de Contratos e Convênios e do Sistema para A União.** 2005. 91 f. Relatório (Estágio Supervisionado). Centro Universitário de João Pessoa – UNIPÊ, João Pessoa.

RESUMO

Este trabalho relata o processo de desenvolvimento de dois sistemas de informação, o Sistema de Contratos e Convênios, que é um modulo *web* de um sistema *desktop* (Sistema de Gerenciamento de Contratos - SGC) utilizado por funcionários da CODATA e tem como principal funcionalidade a disponibilização de dados pela Internet. O segundo é um sistema que está em desenvolvimento para o jornal A União, integrando o faturamento de quatro setores, tornando mais fácil à contabilidade do jornal. O objetivo principal deste relatório é mostrar a metodologia de desenvolvimento utilizada nos dois sistemas, que apresenta a união de Processo Unificado (PU) com uma metodologia ágil. Também serão abordadas todas as fases do desenvolvimento dos sistemas, passando desde levantamento de requisitos até a implementação.

PALAVRAS CHAVES: Engenharia de software, Java, UML, sistema de informação.

LISTA DE ABREVIATURAS E SIGLAS

API – *Application Program Interface*

CASE - *Computer-Aided Software Engineering*

CSS – *Cascading Style Sheets*

HQL – Hibernate *Query Language*

J2EE – Java 2 *Enterprise Edition*

J2SE – Java 2 *Standard Edition*

JSP – JavaServer *Pages*

RUP – Rational *Unified Process*

SC2 – Sistema de Contratos e Convênios

SGC – Sistema de Gerenciamento de Contratos

SGML - *Standard Generalized Markup Language*

UML – *Unified Modeling Language*

W3C – *World Wide Web Consortium*

XP – *Extreme Programming*

XML – *Extensible Markup Language*

XSL – *Extensible Style Language*

LISTA DE FIGURAS

FIGURA 1: O modelo de ciclo de vida em Cascata. 18
FIGURA 2: Modelo de desenvolvimento evolucionário. 19
FIGURA 3: Modelo de desenvolvimento de software em espiral 20
FIGURA 4: Modelo de desenvolvimento orientado a reuso 21
FIGURA 5: Modelo de desenvolvimento iterativo e incremental 22
FIGURA 6: Modelo de ciclo de vida do RUP 24
FIGURA 7: Ciclo de vida do XP. 27
FIGURA 8: Diagrama de caso de uso. 34
FIGURA 9: Diagrama de classe 37
FIGURA 10: Diagrama de seqüência 38
FIGURA 11: Arquitetura do Hibernate 40
FIGURA 12: Diagrama de casos de uso. 46
FIGURA 13: Arquitetura do SC2. 50
FIGURA 14: Diagrama de classe SC2. 51
FIGURA 15: Diagrama de seqüência para consulta de cliente. 53
FIGURA 16: Modelo relacional do banco de dados utilizado pelo SC2 54
FIGURA 17: Tela de autenticação do SC2. 60
FIGURA 18: Tela principal do SC2. 61
FIGURA 19: Tela de alterar senha de usuário. 62
FIGURA 20: Tela de consulta de convênio. 63
FIGURA 21: Tela de relatório dos convênios. 64
FIGURA 22: Diagrama de casos de uso do sistema para A União. 70
FIGURA 23: Arquitetura do sistema para A União 76
FIGURA 24: Diagrama de classe do sistema para A União 78
FIGURA 25: Diagrama de seqüência para o cadastro de assinatura 79
FIGURA 26: Diagrama de seqüência para cadastrar cliente 81
FIGURA 27: Modelo do banco de dados do sistema para A União 82
FIGURA 28: Tela de autenticação do sistema para A União 85
FIGURA 29: Tela principal do sistema para A União 86
FIGURA 30: Tela de cadastro de clientes do sistema para A União. 87
FIGURA 31: Tela de cadastro de assinaturas do sistema para A União. 87

LISTA DE TABELAS

TABELA 1: Lista dos requisitos funcionais do SC2. 44
TABELA 2: Caso de uso consultar cliente. 47
TABELA 3: Caso de uso consultar contrato dos clientes. 47
TABELA 4: Caso de uso consultar nota fiscal dos clientes. 48
TABELA 5: Caso de uso alterar senha do usuário. 48
TABELA 6: Lista de requisitos funcionais do sistema para A União. 68
TABELA 7: Caso de uso cadastrar usuário. 71
TABELA 8: Caso de uso cadastrar forma de pagamento. 72
TABELA 9: Caso de uso cadastrar produto. 73
TABELA 10: Caso de uso cadastrar cliente. 74
TABELA 11: Caso de uso cadastrar assinatura 75
TABELA 12: Caso de uso cadastrar orçamento gráfico. 75

SUMÁRIO

INTRODUÇÃO **12**

CAPÍTULO 1: CONSIDERAÇÕES INICIAIS **13**
1.1 Definição do campo de estágio **13**
1.2 Caracterização do campo de estágio **13**
1.3 Estrutura do trabalho **14**

CAPÍTULO 2: FUNDAMENTAÇÃO TEÓRICA **15**
2.1 Engenharia de software **15**
2.1.1 Definição 16
2.1.2 Processos 17
2.1.2.1 Modelo Cascata 17
2.1.2.2 Desenvolvimento Evolucionário 18
2.1.2.3 Desenvolvimento em espiral 20
2.1.2.4 Desenvolvimento orientado a reuso 21
2.1.2.5 Desenvolvimento iterativo e incremental 21
2.2 Metodologias de desenvolvimento **23**
2.2.1 *Rational Unified Process* (RUP) 23
2.2.2 *Extreme Programming* (XP) 25
2.3 *Unified Modeling Language* (UML) **27**
2.4 Orientação a objetos **28**
2.5 Java **29**
2.6 *Framework* **30**

CAPÍTULO 3: ASPECTOS GERAIS DE DESENVOLVIMENTO UTILIZADOS **31**
3.1 Processo de desenvolvimento **31**
3.2 Análise dos requisitos **33**
3.2.1 Atores 34
3.2.2 Casos de uso 34
3.3 Projeto **35**
3.3.1 Arquitetura do sistema 35
3.3.2 Diagrama de classe 36
3.3.3 Diagrama de seqüência 37
3.3.4 Modelo do banco de dados 38
3.4 Implementação **38**
3.4.1 *Extensible Markup Language* (XML) 39
3.4.2 Hibernate 39
3.4.3 PostgreSQL 41

CAPÍTULO 4: SISTEMA DE CONTRATOS E CONVÊNIOS (SC2) **42**
4.1 Processo de desenvolvimento **43**
4.2 Levantamento dos requisitos **43**
4.2.1 Requisitos funcionais 44
4.2.2 Requisitos não-funcionais 45

4.2.3 Atores 45
4.2.4 Casos de uso 46
4.3 Projeto 49
4.3.1 Arquitetura do sistema 49
4.3.2 Diagrama de classe 51
4.3.3 Diagrama de seqüência 52
4.3.4 Modelo do banco de dados 54
4.4 Implementação 55
4.4.1 Tecnologia utilizada 55
4.4.2 Ferramentas utilizadas 58
4.5 Exemplo de uso do SC2 59
4.6 Considerações finais 65

CAPÍTULO 5: SISTEMA PARA A UNIÃO 66
5.1 Processo de desenvolvimento 67
5.2 Levantamento dos requisitos 67
5.2.1 Requisitos funcionais 67
5.2.2 Requisitos não-funcionais 68
5.2.3 Atores 69
5.2.4 Casos de uso 70
5.3 Projeto 76
5.3.1 Arquitetura do sistema 76
5.3.2 Diagrama de classe 77
5.3.3 Diagrama de seqüência 79
5.3.4 Modelo do banco de dados 82
5.4 Implementação 83
5.4.3 Tecnologia utilizada 83
5.4.2 Ferramentas utilizadas 84
5.5 Exemplo de uso do sistema para A União 85
5.6 Considerações finais 88

CONCLUSÃO 89
6.1 Trabalhos futuros 90

REFERÊNCIAS 91

INTRODUÇÃO

Este relatório de estágio tem o propósito de apresentar as atividades de desenvolvimento realizadas na Companhia de Processamento de Dados da Paraíba – CODATA, objetivando mostrar a metodologia de desenvolvimento utilizada para desenvolver dois sistemas.

A necessidade de disponibilização de informações fora das instalações de empresas vem tornando o desenvolvimento de softwares para *web* muito procurado. Na tentativa de melhorar a disponibilidade de dados de um sistema em perfeito funcionamento, mas com certas limitações na parte de disponibilização de informações, foi proposto o desenvolvimento de um módulo *web* para o Sistema de Gerenciamento de Contratos (SGC), este módulo que teve como nome Sistema de Contratos e Convênios (SC2) trata justamente da disponibilização de dados, gerados pelo SGC, pela *web*. O SC2 irá proporcionar uma melhor apresentação dos dados contidos no banco de dados do SGC. Para que isso seja possível, foram realizadas todas as fases necessárias para desenvolvimento deste sistema mostrando os resultados obtidos.

A integração do faturamento de uma empresa é uma prática comum no mercado, mas a informatização do faturamento levou ao pessoal do jornal A União procurar a CODATA para realizar este tipo de serviço. O sistema para A União proporcionará um melhor gerenciamento do faturamento. Seguindo o mesmo processo do SC2, este sistema realiza todas as fases necessárias para o desenvolvimento, mostrando cada resultado obtido.

Ao longo deste documento serão mostradas as metodologias de desenvolvimento e as formas de modelagens utilizadas nos sistemas propostos.

CAPÍTULO 1

CONSIDERAÇÕES INICIAIS

1.1 Definição do campo de estágio

O estágio foi realizado na gerência de desenvolvimento da Companhia de Processamento de Dados da Paraíba – CODATA. O principal objetivo do estágio foi o conhecimento sobre o fluxo de trabalho em uma empresa de desenvolvimento, podendo adquirir conhecimentos práticos do desenvolvimento de aplicações.

1.2 Caracterização do campo de estágio

A Companhia de Processamento de Dados da Paraíba, representada também pela sigla CODATA, é uma sociedade por ações, economia mista de direito privado, sendo o maior acionista o Governo do Estado, vinculada à Secretaria de Estado da Administração.

O principal objetivo da CODATA é prestar serviços de informática provendo soluções através de tecnologia da informação, que contribuam para a modernização de órgãos centralizados e descentralizados que integram a administração pública estadual e as iniciativas privadas.

A CODATA tem como comprometimento a qualidade na execução de seus serviços e no atendimento a seus clientes, como com a prática de preços competitivos. Possui instalações na cidade de João Pessoa e encontrasse distribuída em dois locais:

- CODATA – SEDE: Funciona a administração da empresa e os serviços relativos ao desenvolvimento de sistemas de informática.
- CODATA – CPD: Funciona as atividades de produção e suporte técnico dos serviços prestados.

1.3 Estrutura do trabalho

O restante deste relatório está dividido em mais cinco capítulos, organizados da seguinte forma:

O Capítulo 2 apresenta ao leitor uma fundamentação da literatura mostrando as principais referências aplicadas ao relatório. No Capítulo 3 são abordados os aspectos relativos ao desenvolvimento dos dois sistemas. O Capítulo 4 mostra todas as fases realizadas para o desenvolvimento do Sistema de Contratos e Convênios. O Capítulo 5 apresenta as etapas do desenvolvimento do sistema para A União. Finalizando com a conclusão obtida após a realização do trabalho.

CAPÍTULO 2

FUNDAMENTAÇÃO TEÓRICA

2.1 Engenharia de software

No início do desenvolvimento de sistemas computadorizados, todos os esforços eram concentrados no desenvolvimento de hardware para reduzir o custo de processamento e armazenamento de dados, mas na medida em que a tecnologia de hardware começou a ser dominada, os desenvolvedores de sistemas voltaram suas atenções no desenvolvimento de softwares.

Ajudado pelo constante crescimento das quedas dos preços dos hardwares e com eficiência e utilidade comprovada, a produção de software começou a crescer, mas junto com o crescimento apareceram os problemas, tais como estimativas de prazos imprecisas, custos maximizados, produtividade baixa dos desenvolvedores, falta de confiança dos clientes e a não adequação dos métodos existentes naquela época. Para tentar solucionar estes problemas nasceu uma área da computação chamada engenharia de software.

2.1.1 Definição

Engenharia de software é uma área da informática que se dedica ao estudo do desenvolvimento dos softwares. Preocupando-se com todos os aspectos necessários para a produção de um software com alta qualidade a um baixo custo. Para que isso seja possível é necessário aplicar técnicas, métodos e ferramentas apropriadas, mas não se preocupando apenas com os processos técnicos de desenvolvimento, se preocupando também com o gerenciamento de projetos, desenvolvimento de novas ferramentas, métodos e técnicas que auxiliam o desenvolvimento de softwares (PRESSMAN, 1995).

Os métodos são conjuntos de tarefas que proporcionam os detalhes de como deve ser feito para construir o software. Estas tarefas incluem: planejamento e estimativa do projeto, análise de requisitos de software e de sistema, projeto do sistema, codificação, teste e manutenção.

As ferramentas servem de apoio para automatizar ou semi-automatizar o desenrolar de todas as tarefas que foram descritas acima. Os softwares que apóiam o projeto de desenvolvimento são conhecidos por CASE (*Computer-Aided Software Engineering*) ou engenharia de software auxiliada por computador, estes softwares têm como objetivo oferecer documentação, automação e racionalização do projeto de software e de sua implementação, exemplos de ferramentas CASE são os geradores automáticos de aplicação e as ferramentas de modelagem, projeto e implementação.

As técnicas constituem a ligação entre os métodos e as ferramentas definindo a seqüência em que os métodos serão aplicados.

2.1.2 Processos

Processo de software compreende um conjunto de etapas e resultados associados, que na maioria das vezes são executados por engenheiros de softwares para gerar um produto de software. Embora existam muitos processos de software, existem quatro etapas principais que são comuns a todos os processos, como:

- Especificação;
- Desenvolvimento;
- Validação;
- Evolução do software.

Para representar um processo de software será utilizada uma forma abstrata que chamaremos de modelo de desenvolvimento, onde cada modelo representa um processo a partir de uma perspectiva particular, proporcionando apenas informações parciais do processo, considerando as atividades, os produtos de software e o papel das pessoas envolvidas.

2.1.2.1 Modelo Cascata

Este modelo foi idealizado por Royce em 1970, sendo o primeiro a ser adotado pelos desenvolvedores. Tem como característica principal à seqüencialidade das atividades, e um conjunto de fases bem definidas que correspondem às etapas do ciclo de vida de um software, não havendo mistura nas fases, ou seja, uma fase só começa quando a outra termina,

facilitando a gestão dos projetos. Ao final de cada fase produtos são gerados, servindo de entrada para a nova fase. Embora este processo seja o mais antigo, ele continua sendo o modelo procedimental mais amplamente utilizado pela engenharia de software. Esse modelo é ilustrado na FIGURA 1.

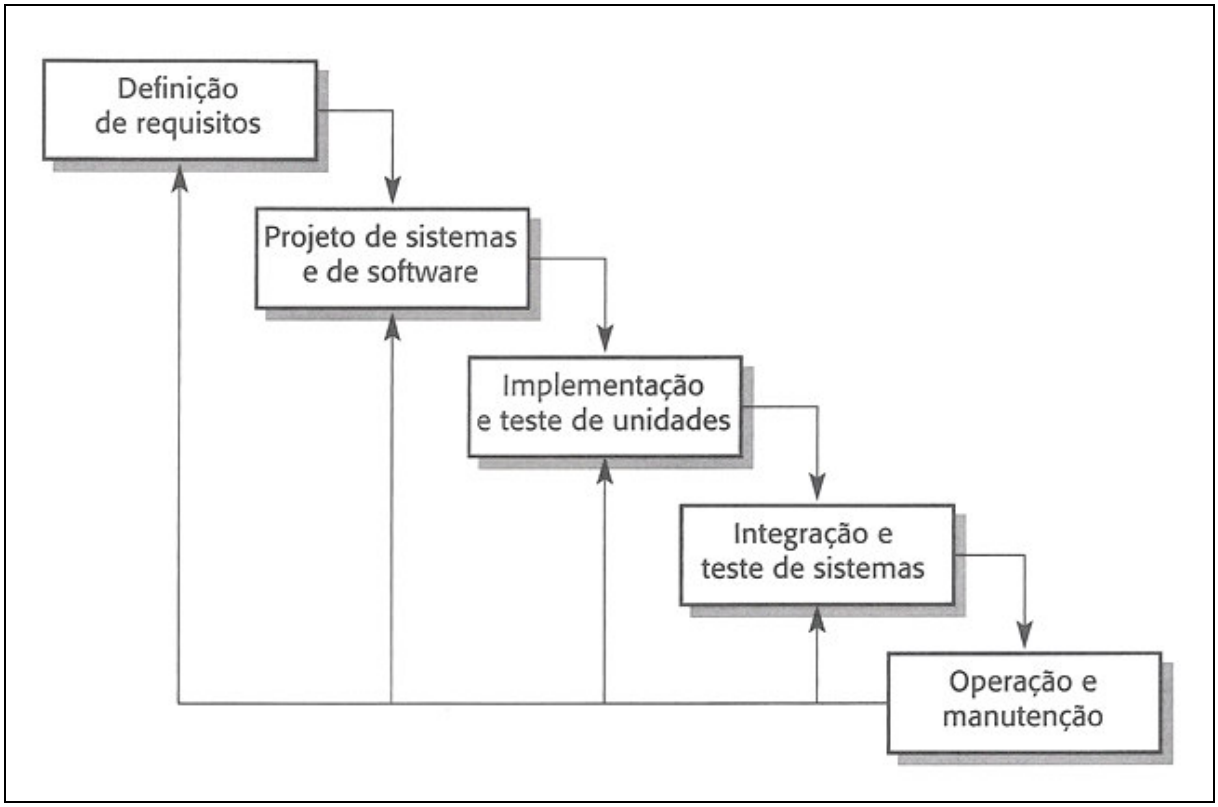

FIGURA 1: O modelo de ciclo de vida em Cascata.

Fonte**:** SOMMERVILE, Ian. **Engenharia de Software.** 6. ed. São Paulo: Addison Wesley, 2003.

2.1.2.2 Desenvolvimento Evolucionário

O desenvolvimento evolucionário também pode ser chamado de prototipação, é muito bem utilizado quando o cliente ainda não definiu todos os requisitos de entrada, ou

quando o desenvolvedor não tem a certeza da eficiência de um algoritmo, ou da adaptabilidade de um sistema operacional e em muitas outras situações. Este modelo tem como base a idéia de desenvolver uma implementação inicial, mostrar para o usuário e fazer seu aprimoramento por meio de várias versões, ate chegar ao resultado esperado, como ilustrado na FIGURA 2.

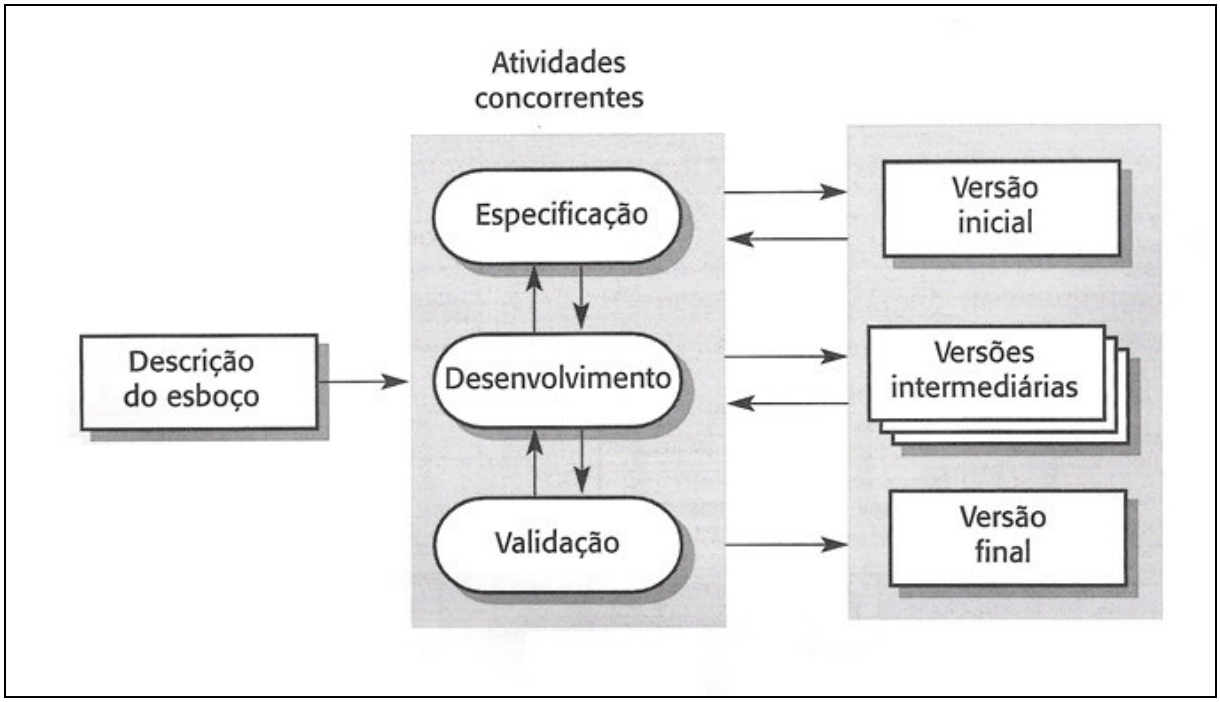

FIGURA 2: Modelo de desenvolvimento evolucionário.
Fonte: SOMMERVILE, Ian. **Engenharia de Software.** 6. ed. São Paulo: Addison Wesley, 2003.

Há dois tipos de desenvolvimento evolucionário:

- **Desenvolvimento exploratório:** trabalha com os requisitos compreendido do cliente e evolui com o acréscimo de novas funcionalidades, à medida que o cliente propõe.
- **Desenvolvimento de protótipos descartáveis:** a partir de protótipos experimentais tenta compreender os requisitos do cliente, se concentrando em fazer experimentos com os requisitos mal compreendidos.

2.1.2.3 Desenvolvimento em espiral

Este modelo foi desenvolvido juntando as “coisas boas” do modelo cascata junto com as do desenvolvimento evolucionário, acrescentando a análise dos riscos. Este modelo é divido em quatro atividades principais: definição de objetivos, avaliação e redução dos riscos, desenvolvimento e validação, e planejamento. Mas em vez de representar o processo de software como uma seqüência de atividades, o processo é representado como um espiral, ou seja, cada loop no espiral representa uma fase completa do projeto de software, passando pelas quatro atividades principais. A FIGURA 3 ilustra esse modelo.

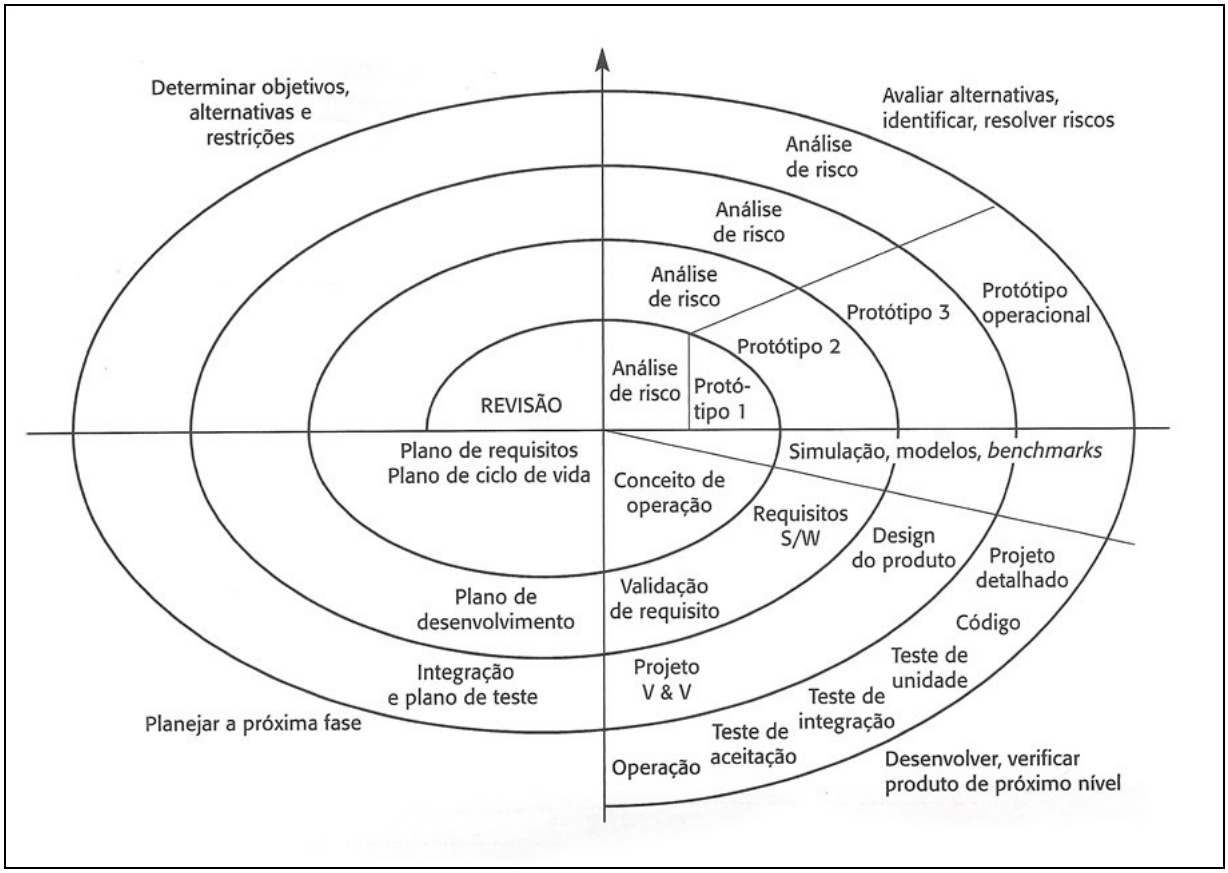

FIGURA 3: Modelo de desenvolvimento de software em espiral

Fonte: SOMMERVILE, Ian. **Engenharia de Software.** 6. ed. São Paulo: Addison Wesley, 2003.

2.1.2.4 Desenvolvimento orientado a reuso

Esse modelo, como o nome já diz, utiliza reutilização para gerar novas aplicações. Portanto para utilizar esta abordagem o desenvolvedor deve contar com uma ampla base de componentes de softwares reutilizáveis que será de alguma forma integrada a nova aplicação. A FIGURA 4 mostra o modelo genérico para o desenvolvimento orientado a reuso.

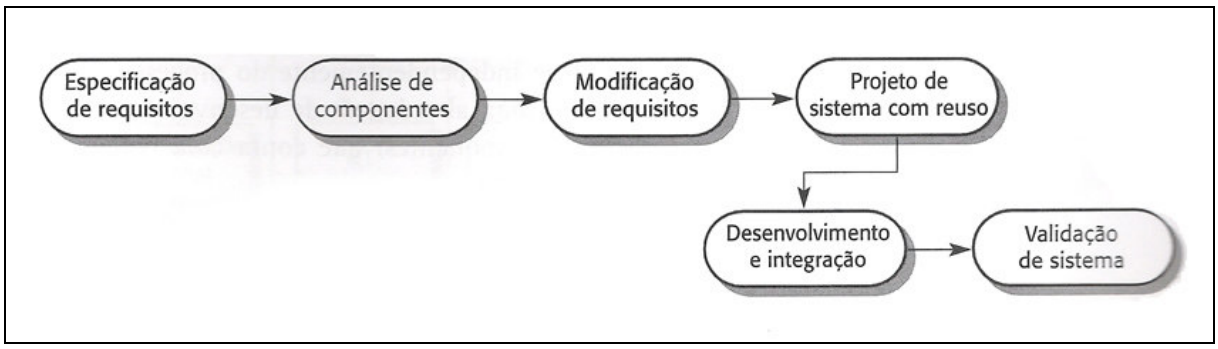

FIGURA 4: Modelo de desenvolvimento orientado a reuso

Fonte: SOMMERVILE, Ian. **Engenharia de Software.** 6. ed. São Paulo: Addison Wesley, 2003.

2.1.2.5 Desenvolvimento iterativo e incremental

Esse modelo foi sugerido por Mills em 1980 para reduzir o retrabalho, é uma junção do modelo espiral com o modelo de prototipação, ou seja, ele junta o que há de melhor neste dois modelos.

No desenvolvimento iterativo e incremental em vez de entregar o sistema como um todo, o desenvolvedor junto com o cliente identifica quais as funções com maior

prioridade e a medida em que as funcionalidades vão sendo implementadas serão entregues ao cliente em forma de incremento, permitindo assim uma constante avaliação do produto. Em cada iteração os requisitos são analisados, projetados, implementados, testados e implantados. Na próxima iteração novos requisitos são desenvolvidos gerando assim uma nova versão para o sistema, ate chegar na versão final. É iterativo porque há repetições do ciclo e é incremental porque ao final de cada ciclo possui um protótipo pronto para ser entregue ao cliente.

Com esse modelo de desenvolvimento o sistema cresce pela adição de novas funcionalidades e do refinamento das existentes.

Como vantagens desse tipo de desenvolvimento, têm-se que os riscos mais comprometedores são atacados primeiro, acomodação mais fácil em caso de mudança dos requisitos, integração continua, reuso facilitado e que uma porção do sistema pode ser desenvolvida e entregue enquanto uma outra parte do mesmo ainda está em desenvolvimento. Por estes motivos que no desenvolvimento dos sistemas utilizou este tipo de desenvolvimento. O modelo iterativo e incremental é ilustrado na FIGURA 5.

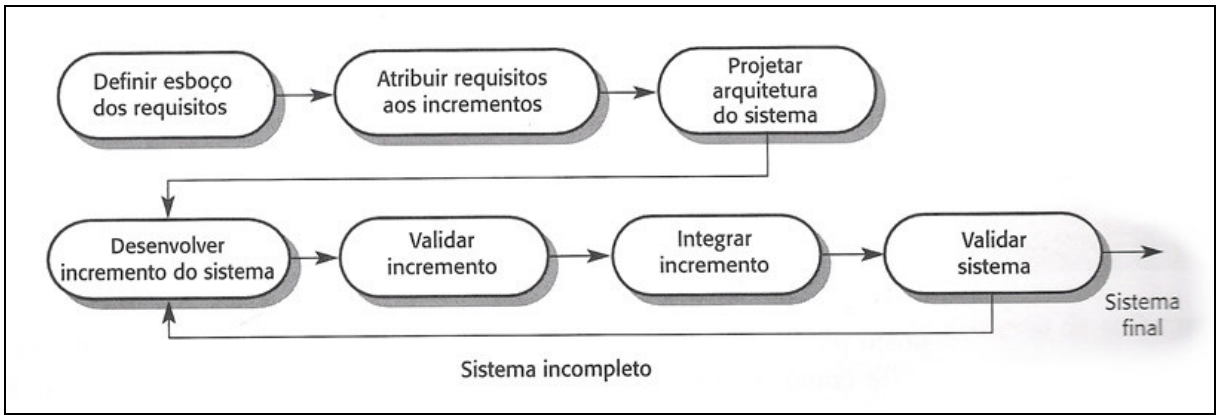

FIGURA 5: Modelo de desenvolvimento iterativo e incremental

Fonte: SOMMERVILE, Ian. **Engenharia de Software.** 6. ed. São Paulo: Addison Wesley, 2003.

2.2 Metodologias de desenvolvimento

Metodologias de desenvolvimento é uma especialização de um processo, onde são incluídos vários elementos específicos, objetivando uma produção mais fácil de software de alta qualidade. Os elementos inseridos são padrões, métodos, linguagens, ferramentas e comportamento organizacional, ajudando na realização, modelagem, codificação, automatização do processo de desenvolvimento. As metodologias apresentadas nesta seção são *Rational Unified Process* (RUP)[1] e *Extreme Programming* (XP)[2].

2.2.1 *Rational Unified Process* (RUP)

RUP é uma metodologia completa criada pela Rational Software Corporation para viabilizar que grandes projetos de softwares sejam bem sucedidos. Tem como principal característica um processo de desenvolvimento iterativo e incremental, com modelo de ciclo de vida guiado por casos de uso e com desenvolvimento centrado na arquitetura do sistema.

O RUP possui um processo de software bem definido e bem estruturado, ele define quem é o responsável pelo o que deve ser feito e quem deve fazer, provendo um ciclo de vida bem definido deixando bem claro os marcos essenciais e os pontos de decisões, alem

[1] http://www-306.ibm.com/software/awdtools/rup/
[2] http://www.extremeprogramming.org/

de utilizar UML[3] para especificar, modelar e documentar os artefatos gerados apresentando assim uma modelagem visual que facilita a comunicação de diferentes artefatos do sistema.

O ciclo de vida de um projeto segundo RUP pode ser descrito em duas dimensões como mostrado na FIGURA 6, onde o eixo horizontal representa o tempo e os aspectos dinâmico expresso em fases e iterações e o eixo vertical representam os aspectos estáticos descrito em termos de fluxo de atividades.

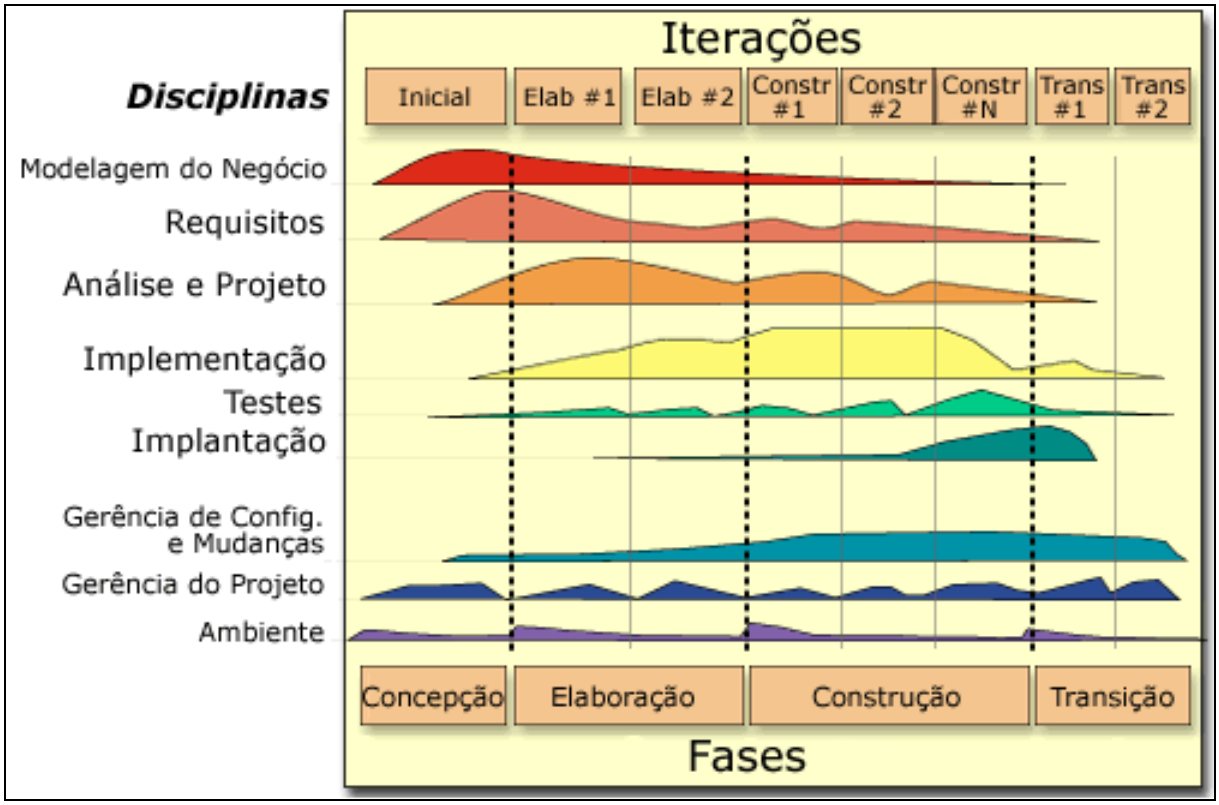

FIGURA 6: Modelo de ciclo de vida do RUP

Fonte: **PDS-DATASUS::Histórico.** Disponível em: <http://pds.datasus.gov.br/geral/historico.html>. Acesso em: 26/11/2005.

[3] http://www.uml.org/

2.2.2 *Extreme Programming* (XP)

XP é uma metodologia ágil de desenvolvimento de software que prima pela satisfação do cliente, qualidade do software e agilidade, destina-se a equipes pequenas ou médias que precisam desenvolver softwares rapidamente e em um ambiente de exigência em que os requisitos são vagos e em constantes mudanças. Para conseguir a qualidade e a agilidade necessária se baseia em quatro valores principais: simplicidade, comunicação, *feedback* e coragem.

Para aplicar os valores durante o desenvolvimento, XP propõe uma série de práticas:

- **Jogo de planejamento** (*Planning game*): tem como propósito determinar o escopo da próxima iteração, definindo as prioridades do negócio e as estimativas de custos fornecidas pelos programadores, com estas informações é decidido o que é necessário ser feito e o que pode ser adiado.
- **Lançamentos curtos** (*Small releases*): é colocado uma versão simples em produção, e esta versão é freqüentemente atualizada em ciclos muito curtos.
- **Metáfora** (*Metaphor*): guia todo o desenvolvimento a partir de simples estórias que mostra como o sistema funciona, ajudando que todos entendam os elementos básicos e seu relacionamento.
- **Projeto simples** (*Simple design*): o sistema deve ser feito o mais simples possível que satisfaça os atuais requisitos, provendo valor de negocio o mais rápido possível.
- **Testes** (*Testing*): os programadores desenvolvem software escrevendo primeiro os teste para só então escreverem os software que atendam aos requisitos destes testes e depois criam mais testes para demonstrar que as funcionalidades estão prontas.

- **Reestruturação de código** (*Refactoring*): programadores reestruturam o sistema sem mudar o comportamento eliminando códigos duplicados, melhorando a comunicação, mantendo sua clareza, mantendo-o simples porém completo.
- **Programação em dupla** (*Pair programming*): toda a produção deve ser feito em pares, dois programadores trabalhando na mesma máquina.
- **Propriedade coletiva** (*Collective ownership*): qualquer um pode mudar o código do sistema a qualquer momento, ou seja, o código é de todos.
- **Integração continua** (*Continuous integration*): toda vez que uma tarefa é completa, esta tarefa é integrada ao sistema.
- **40 horas de trabalho semanal** (40-*hour week*): trabalhe não mais que 40 horas por semana.
- **Cliente dedicado** (*On-site custumer*): o cliente é um membro da equipe disponível a todo tempo para responder a qualquer duvida.
- **Código padrão** (*Coding standards*): os programadores escrevem código seguindo regras, enfatizando assim uma melhor clareza e fácil entendimento do código.

A FIGURA 7 mostra de forma simplificada o ciclo de vida do XP.

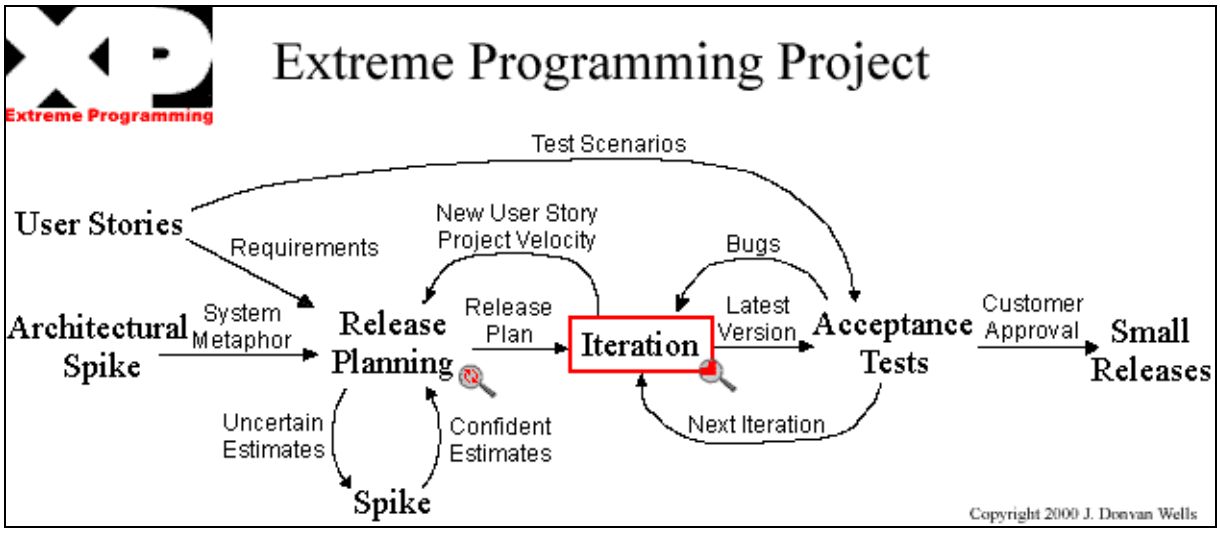

FIGURA 7: Ciclo de vida do XP.

Fonte: **XP flow Chart**. Disponível em: <http://www.extremeprogramming.org/map/project.html>. Acesso em: 14/12/2005.

2.3 *Unified Modeling Language* (UML)

UML significa linguagem de modelagem unificada, ela é uma linguagem gráfica que serve para especificar, construir, visualizar e documentar os artefatos de um sistema de software orientado a objetos. Oferecendo para os modeladores uma linguagem expressiva e visual. Para conseguir isso a UML oferece um conjunto extenso de diagramas, dividido em visões:

- **Estrutural:** Diagramas de classes e de objetos.
- **Comportamental:** Diagramas de casos de uso, de seqüência, de colaboração, de estados e de atividades.
- **Arquitetural:** Diagramas de componentes e de implantação.

2.4 Orientação a objetos

Orientação a objetos é um paradigma da programação onde a idéia que um programa de computador é composto de uma coleção de unidades individuais ou objetos, e foca nas estruturas de dados, ao invés de operações. Para este paradigma propõe-se uma arquitetura de software baseada nos objetos que o sistema manipula, onde esses objetos são baseados no mundo real (estados + comportamento). Alguns conceitos enfatizados pela orientação a objetos são:

- **Classe:** representa um conjunto de objetos com características afins. Uma classe é o local onde encontrasse definido o comportamento dos objetos, através de métodos, e suas propriedades, através de atributos.
- **Objetos:** é uma estrutura que possui dados e funcionalidades encapsuladas, sendo à base da modularidade e estrutura na orientação a objetos.
    - Atributos: são as características do objeto, pode ser outro objeto.
    - Métodos: é como os objetos implementam suas funcionalidades.
- **Abstração:** processo de identificação dos objetos e seus relacionamentos, ignorando alguns detalhes da informação que está manipulando, focalizando apenas no essencial.
- **Encapsulamento:** é uma forma de separar a parte interna e externa do objeto, permitindo que apenas os métodos internos mudem o estado do objeto.
- **Herança:** mecanismo que permite derivar novas classes (subclasses) a partir de classes já existentes (superclasse), reutilizando todos os comportamentos (métodos) e estados possíveis (atributos).
- **Polimorfismo:** é a capacidade de um objeto se comportar de várias formas dependendo de como ele é instanciado.

- **Composição:** é um mecanismo em que um objeto tem um atributo que é uma referencia a outro objeto.

2.5 Java[4]

Java é uma linguagem de programação orientada a objetos de propósito geral. É uma linguagem simples, distribuída, interpretada, robusta, segura, independente de plataforma, portável e concorrente.

- **Simples:** por possui sintaxe baseada em C e C++.
- **Orientada a objetos:** pois oferecendo suporte aos principais conceitos de orientação a objetos, como classes, objetos, encapsulamento, herança e polimorfismo.
- **Distribuída:** por dar suporte a Internet/WWW, objetos distribuídos, acesso a arquivos remotos, banco de dados e etc.
- **Robusta:** por ser uma linguagem fortemente tipada, sem ponteiros, com mecanismo de coleta de lixo (*garbage collector*) automático e tratamento de exceções.
- **Segura:** apresenta uma série de restrições de segurança, possuindo uma biblioteca pronta para tratar de segurança.
- **Independente de plataforma:** utiliza instruções de uma linguagem de máquina virtual (*bytecode*), onde o código intermediário da máquina virtual é interpretado em tempo de execução para a plataforma real.

[4] http://java.sun.com/

- **Portável:** pois pode ser executa em qualquer maquina que possua o interpretador Java (JVM), ou seja, um único programa executa uniformemente em qualquer plataforma.
- **Concorrente:** através de suporte a *threads* e mecanismo de sincronização.

2.6 *Framework*

Framework é uma técnica de reuso orientada a objetos que provê reutilização tanto de projeto quanto de código, constitui em uma aplicação quase completa que captura os conceitos mais gerais de um domínio e utiliza um conjunto de classes e interfaces para decompor o problema (OLIVEIRA, 2005). *Framework* tem como objetivo manter o conhecimento sobre o domínio da aplicação, reduzir o numero de linhas de código e otimizar generalizações e especializações ganhando assim uma maior modularidade, reusabilidade e extensibilidade.

- **Modularidade:** por obter encapsulamento de implementações flexíveis através de uma interface estável.
- **Reusabilidade:** definindo componentes genéricos que podem ser reaplicáveis.
- **Extensibilidade:** provendo métodos explícitos que possibilitam as aplicações estenderem suas interfaces.

CAPÍTULO 3

ASPECTOS GERAIS DE DESENVOLVIMENTO UTILIZADOS

Neste capítulo será apresentada uma série de definições que servirá como fundamento para os próximos.

Na seção 3.1 será apresentado o processo de desenvolvimento utilizado nos sistemas. A seção 3.2 mostra as definições necessárias para que se entenda a fase de análise dos requisitos dos sistemas. Na seção 3.3 serão mostradas as definições da fase de projeto dos sistemas.

3.1 Processo de desenvolvimento

Acreditando que o processo de desenvolvimento seja um fator fundamental para garantir a qualidade de um sistema, decidiu-se que o desenvolvimento deveria ser orientado por uma metodologia de Engenharia de Software que tornasse possível a produção de software compatível com a qualidade exigida pelo sistema. Com isso, optou-se por utilizar uma metodologia baseada na combinação de metodologia ágil (XP) e processo unificado (RUP). Esta metodologia utiliza um modelo de processo iterativo e incremental, utilizando RUP para fase de levantamento dos requisitos e projeto gerando toda a documentação necessária e o XP para a fase de implementação incorporando todos os seus valores e utilizando algumas de suas práticas. A escolha por esta metodologia deveu-se ao fato de que

se precisava optar por uma metodologia da Engenharia de Software que fosse ágil, com boa estrutura, orientada a objetos e com documentação e diagramação baseada nos padrões da UML.

O processo é dividido em três etapas:

- **Análise dos requisitos:** esta etapa foi baseada no RUP gerando uma lista dos requisitos funcionais, não-funcionais e o diagrama de casos de uso.
- **Projeto:** esta etapa também é baseada no RUP, mas como é um processo que tende pela agilidade foram cortados diversos artefatos do RUP tornando esta etapa mais curta onde será definida a arquitetura, o diagrama de classes, o diagrama de seqüência e o modelo do banco de dados.
- **Implementação:** etapa baseada no XP, onde serão utilizadas diversas praticas, as principais utilizadas são: *releases* curtos, refatoramento, código padrão e cliente dedicado.

O motivo que levou a optar por uma metodologia ágil foi pelo curto espaço de tempo que se teve para realizar a implementação e documentação dos sistemas, pois como se trata de um estágio, após o término o sistema será mantido por outra pessoa, portanto para que essa pessoa fique por dentro do sistema, este tem que estar bem documentado.

Deve-se destacar que o ciclo de desenvolvimento é totalmente voltado para uma produção rápida do sistema, ou seja, o ciclo mais longo do desenvolvimento é o de implementação. A seguir será descrita cada etapa do desenvolvimento.

3.2 Análise dos requisitos

O objeto deste ciclo é entender o problema capturando as reais necessidades que o cliente deseja informatizar, pois é de suma importância o perfeito entendimento do que se vai fazer, antes de fazê-lo. Caso este ciclo não seja bem feito, podem-se obter problemas quanto à distorção das funcionalidades, preços e prazos mal estimados e o descontentamento do cliente. Para tanto uma lista dos requisitos funcionais e não-funcionais do sistema e um modelo de casos de uso será gerado nessa etapa.

Os requisitos são as funcionalidades e as condições do sistema que o cliente deseja para atender suas necessidades. Portanto para poder chegar a este grau de concepção dos requisitos, foram realizadas diversas entrevistas com os usuários dos sistemas e as pessoas que possuem interesse ou direitos sobre os sistemas (*stackholder*). Os requisitos estão divididos em dois grupos:

- **Requisitos funcionais:** são aqueles que descrevem o processamento a ser realizado pelo sistema.
- **Requisitos não funcionais:** são aqueles que descrevem as qualidades do sistema, não estando diretamente ligado às operações do sistema.

3.2.1 Atores

São papéis desempenhados por pessoas, dispositivos ou sistemas dentro do sistema podendo até ser o próprio sistema, ou seja, qualquer coisa que possui interesse na operação bem sucedida do sistema.

3.2.2 Casos de uso

Modelo de casos de uso é uma técnica da UML utilizada para melhorar a entendimento dos requisitos de um sistema. Tem como objetivo descrever uma seqüência de eventos de um ator utilizando um sistema para completar o processo. Para representar o modelo de casos de uso utiliza um diagrama composto por atores, casos de uso e os relacionamentos entre eles. Na FIGURA 8 são mostrados os elementos básicos de um diagrama de casos de uso.

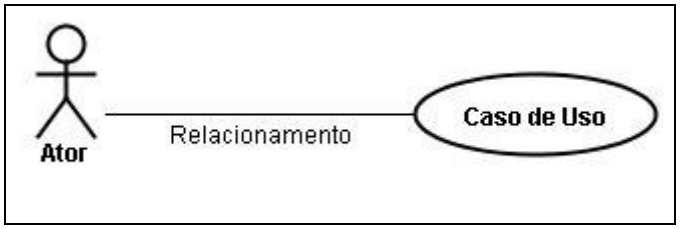

FIGURA 8: Diagrama de caso de uso.

Fonte: Pesquisa direta

3.3 Projeto

Na seção 3.2 foram apresentados os passos que foram utilizados para realizar a análise dos requisitos nos sistemas, garantindo assim um bom entendimento dos problemas que os sistemas tem a resolver. Nesta seção serão apresentadas as definições que são importantes para compreender a fase de projeto dos sistemas.

Na fase de projeto os requisitos gerados na fase de análise dos requisitos serão transformados em um projeto de sistema, sugerindo uma arquitetura robusta e já adaptando o projeto para o ambiente de desenvolvimento, definindo aqui uma solução lógica para o sistema. Nesta etapa serão apresentados o modelo da arquitetura do sistema, uma demonstração dos principais casos de uso através de diagramas de seqüência, e o modelo de dados através do diagrama de classes.

3.3.1 Arquitetura do sistema

A arquitetura do sistema define em um alto nível de abstração o domínio e o funcionamento do sistema, mostrando a estrutura geral e as dependências existentes entre os componentes.

Para que pudesse ser gerada uma solução na qual o sistema tivesse uma boa performance, segurança, proteção, disponibilidade e manutenabilidade houve a necessidade de utilizar uma arquitetura que minimizasse a comunicação entre os sub-sistemas, isolasse os componentes, permitisse o encapsulamento das informações, possibilitasse o reuso dos

componentes e facilitasse as manutenções, portanto foi utilizada uma arquitetura de múltiplas camadas, onde basicamente o sistema é dividido em 3 partes:

- **Apresentação:** é a camada de interface com o usuário, ou seja, é a camada encarregada de repassar solicitações de tarefas para a camada intermediária ou de mostrar resultados de processamento.
- **Lógica da aplicação:** é a camada intermediária, que faz a comunicação entre a camada de apresentação e a de dados. Esta camada é dividida em duas sub camadas:
    - Camada de domínio: camada que está relacionada ao domínio do problema.
    - Camada de serviço: camada que constitui dos aspectos de serviços (segurança, interação com o banco de dados, etc).
- **Dados:** é a camada que constitui o mecanismo que irá armazenar todos os dados relativos ao sistema.

Como saída desta etapa será mostrado um diagrama de pacotes que representa a arquitetura do sistema.

3.3.2 Diagrama de classe

Este artefato é um dos mais importantes, pois demonstra a estrutura estática das classes listando todos os conceitos do domínio que será implementado. Este diagrama fornece uma visão das classes e seus relacionamentos como componentes de software do sistema. As classes se relacionam umas com as outras por meios de associações, dependências e especializações como mostrado na FIGURA 9.

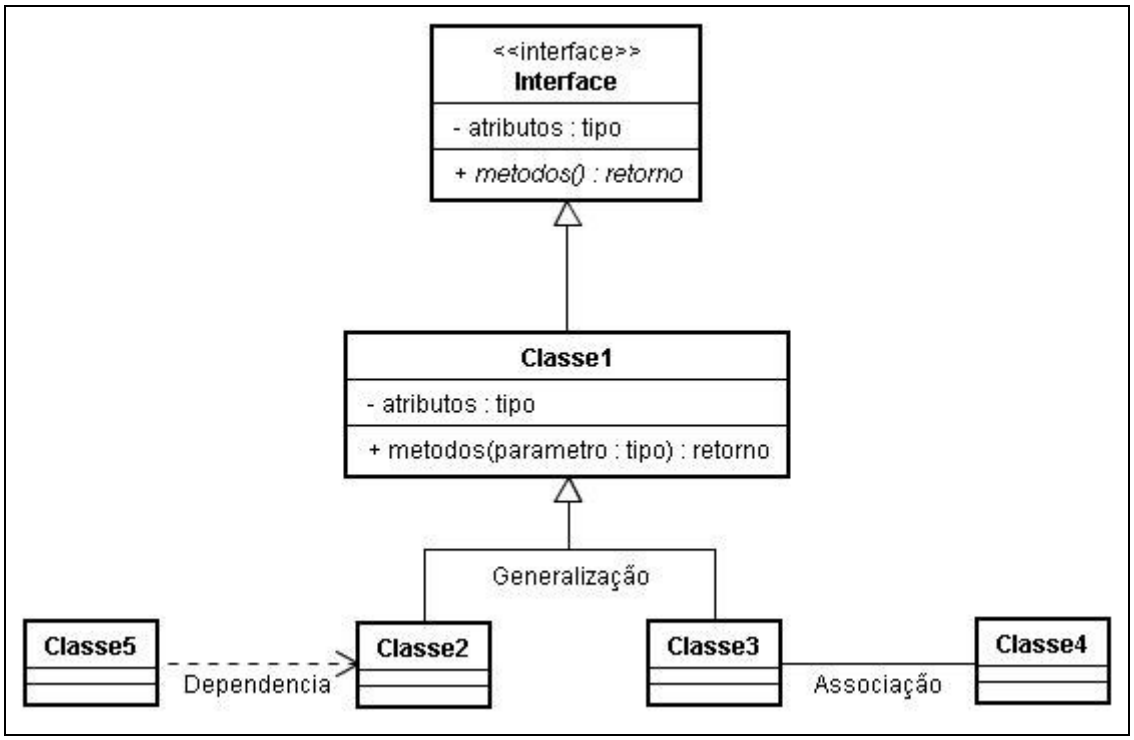

FIGURA 9: Diagrama de classe

Fonte: Pesquisa direta

3.3.3 Diagrama de seqüência

Os diagramas de seqüência são usados para mostrar com maior detalhamento a troca de mensagens realizadas pelas classes com outros objetos, para modelar a interação entre objetos em um sistema no comportamento de um único caso de uso, enfatizando a ordem e os momentos em que as mensagens são enviadas aos objetos. Os objetos são representados por linhas verticais enquanto as mensagens entre dois objetos são representadas por vetores horizontais como é exibido na FIGURA 10.

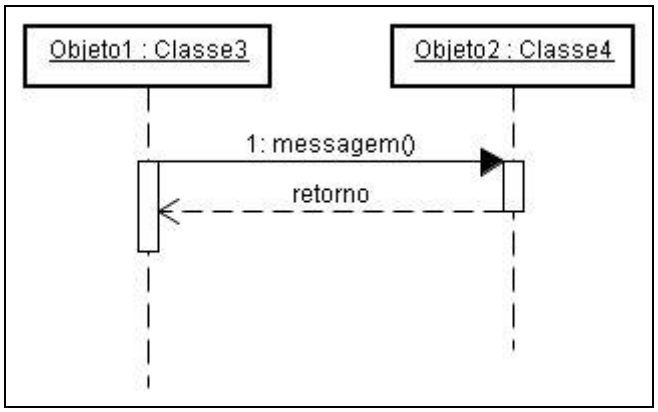

FIGURA 10: Diagrama de seqüência

Fonte: Pesquisa direta

3.3.4 Modelo do banco de dados

O modelo do banco de dados resume-se em especificar as tabelas do banco de dados e seus inter-relacionamentos. Nesta etapa será apresentado um diagrama de entidade e relacionamento.

3.4 Implementação

Passando pela fase de projeto onde todas as decisões mais difíceis foram tomadas. A fase de implementação constitui na tradução do projeto para o código. Nesta etapa será mostrada as tecnologias utilizadas e as ferramentas que ofereceram suporte a geração de

código para os sistemas. Para explicar melhor algumas das tecnologias utilizadas nos dois sistemas adiantarei a explicação de XML[5] e do *framework* Hibernate[6].

3.4.1 *Extensible Markup Language* (XML)

Extensible Markup Language (XML) é uma tecnologia para criar linguagem de marcação designada para descrever tipos diferentes de dados. XML é uma versão simplificada da SGML (*Standard Generalized Markup Language* - Padrão de marcação generalizada) e tem como propósito a facilidade de compartilhar dados em diferentes tipos de sistemas.

Esta tecnologia é usada para fazer o mapeamento das tabelas do banco com as classes Java.

3.4.2 Hibernate

Hibernate é um poderoso *framework* para persistência objeto-relacional, de código livre (*Open Source*) e gratuito (*Free*) possuindo um alto desempenho e de fácil manuseio. É um serviço de persistência que armazena objetos Java em banco de dados e também provê uma visão orientada a objeto dos dados do banco, ou seja, é uma maneira simples e eficiente de trabalhar com dados de um banco relacional em forma de objetos Java. Hibernate permite

[5] http://www.w3.org/XML/
[6] http://www.hibernate.org

desenvolver classes persistentes em Java convencional, possibilitando o uso de associações, herança, polimorfismo, composição e de coleções em Java. Hibernate oferece serviço de consultas possuindo uma própria linguagem de consulta orientada a objetos que se chama HQL (Hibernate *Query Language*), mas também permite o uso de SQL Nativo e de uma API chamada Criteria para realizar consultas.

O Hibernate persiste objetos Java comuns, utilizando reflexão para acessar as propriedades persistentes deste objeto. As classes persistentes são definidas em documentos de mapeamentos utilizando arquivos XML para descrever os campos, associações e subclasses. A FIGURA 11 mostra uma visão de alto nível da arquitetura do Hibernate.

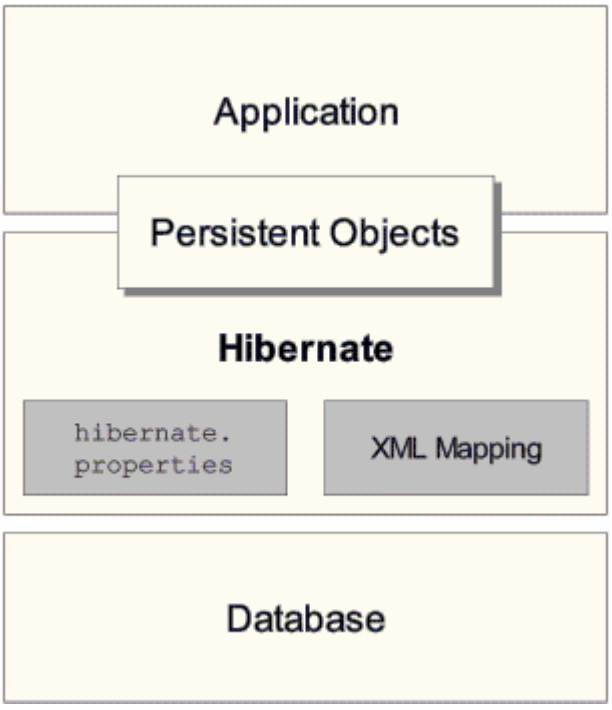

FIGURA 11: Arquitetura do Hibernate

Fonte: SAUVÉ, Jacques. **Persistência usando Hibernate.** Disponível em: <http://jacques.dsc.ufcg.edu.br>. Acesso em: 17/11/2005.

3.4.3 PostgreSQL[7]

PostgreSQL é um sistema gerenciador de banco de dados objeto-relacional (SGBDOR) desenvolvido pelo Departamento de Ciência da Computação da Universidade da Califórnia em Berkeley. É um banco de dados gratuito, de código fonte aberto e que oferece muitas funcionalidades modernas:

- *Queries* complexas
- Chaves estrangeiras
- *Triggers*
- *Views*
- Índices
- Integridade transacional
- Criação de novos tipos de dados
- Criação de funções

[7] http://www.postgresql.org

CAPÍTULO 4

SISTEMA DE CONTRATOS E CONVÊNIOS (SC2)

Neste capítulo será apresentado o Sistema de Contratos e Convênios (SC2), este sistema é um módulo web do Sistema de Controle de Contratos (SGC). O SGC é um sistema que controla o funcionamento de três gerências da CODATA, sendo elas a gerência financeira, de negócios e de recursos humanos. Este sistema permite que os funcionários destas gerências realizem cadastros, alterações, exclusões, consultas e emitam relatórios tanto para contratos como para convênios, além de gerar e emitir notas fiscais. Este é um sistema cliente servidor feito em Delphi que utiliza como banco de dados o PostgreSQL.

O problema do SGC é falta de informação sobre os contratos e convênios do órgão quando os diretores e gerentes estão fora das instalações, perdendo assim muito tempo para o fechamento de contratos e convênios.

Como já foi mencionado acima o SC2 é um módulo web do SGC, ou seja, uma extensão do SGC na WEB viabilizando algumas consultas já existentes no SGC e outras novas, sendo apenas um sistema que consulta a base de dados do SGC. Este módulo como o SGC é para uso dos funcionários da CODATA e tem como principal objetivo levar os dados que o SGC armazena no banco de dados para a web resolvendo assim o principal problema do SGC, tornando mais fácil a visualização destes dados fora das instalações da CODATA, sempre oferecendo uma melhor apresentação, melhorando assim os fechamentos de contratos e convênios realizados fora do órgão, que antigamente não possuía informações para fechar tanto os contratos quanto os convênios, tendo que esperar dias e dias para poder tomar decisões.

O SC2 possui apenas um usuário, ou seja, todos os funcionários da empresa possuem as mesmas permissões no sistema. Ele é basicamente um sistema de consulta de dados de um banco de dados, portanto sendo um sistema dependente do SGC para alimentar a base de dados. Permite que os usuários agilmente obtenham informações sobre os contratos e convênios da empresa.

4.1 Processo de desenvolvimento

O processo de desenvolvimento utilizado para este sistema foi descrito na seção 3.1, sendo basicamente uma mescla de metodologia ágil (XP) e processo unificado (RUP). Os artefatos do RUP na fase de análise dos requisitos foi o modelo de casos de uso, na fase de projeto utilizou-se de um diagrama da arquitetura, diagrama de classes e do diagrama de seqüência. Na fase de implementação foram utilizadas as seguintes técnicas de XP: *releases* curtos, refatoramento, código padrão e cliente dedicado.

4.2 Levantamento dos requisitos

Após diversas entrevistas com os usuários e *stackholder* do sistema, também levando em consideração toda a base de conhecimento adquirida nas manutenções realizadas no SGC, podemos chegar as seguintes necessidades:

4.2.1 Requisitos funcionais

A TABELA 1 mostra os requisitos funcionais do SC2:

TABELA 1: Lista dos requisitos funcionais do SC2.

Identificação	Descrição
RF01	Consulta clientes
RF02	Consulta contratos dos clientes
RF03	Consulta notas fiscais dos clientes
RF04	Consulta fornecedores
RF05	Consulta contratos dos fornecedores
RF06	Consulta notas fiscais dos fornecedores
RF07	Consulta conveniado
RF08	Consulta convênios
RF09	Altera senha do usuário

Fonte: Pesquisa direta.

4.2.2 Requisitos não-funcionais

No SC2 foram abordados os seguintes requisitos não-funcionais:

- **Facilidade de uso:** implementado com tecnologia que permite um simples acesso aos recursos do sistema através do mouse e necessitando apenas de um *browser* (navegador) para utilizar o sistema.
- **Segurança:** controle dos usuários através de uma tela de *login*/senha, com senhas criptografadas.
- **Portabilidade:** o sistema foi implementado com tecnologia que permite a implantação do sistema na maioria dos sistemas operacionais.
- **Interface:** dispõe de uma interface gráfica amigável e de fácil aprendizado.

4.2.3 Atores

Para este sistema apenas um tipo de ator foi necessário que será descrito logo abaixo:

- **Usuário:** são todos os funcionários das gerências envolvidas no sistema mais os diretores e gerentes da CODATA.

4.2.4 Casos de uso

Nesta etapa será mostrado o diagrama de casos de uso que oferece uma visão gráfica das funcionalidades do sistema como mostra a FIGURA 12.

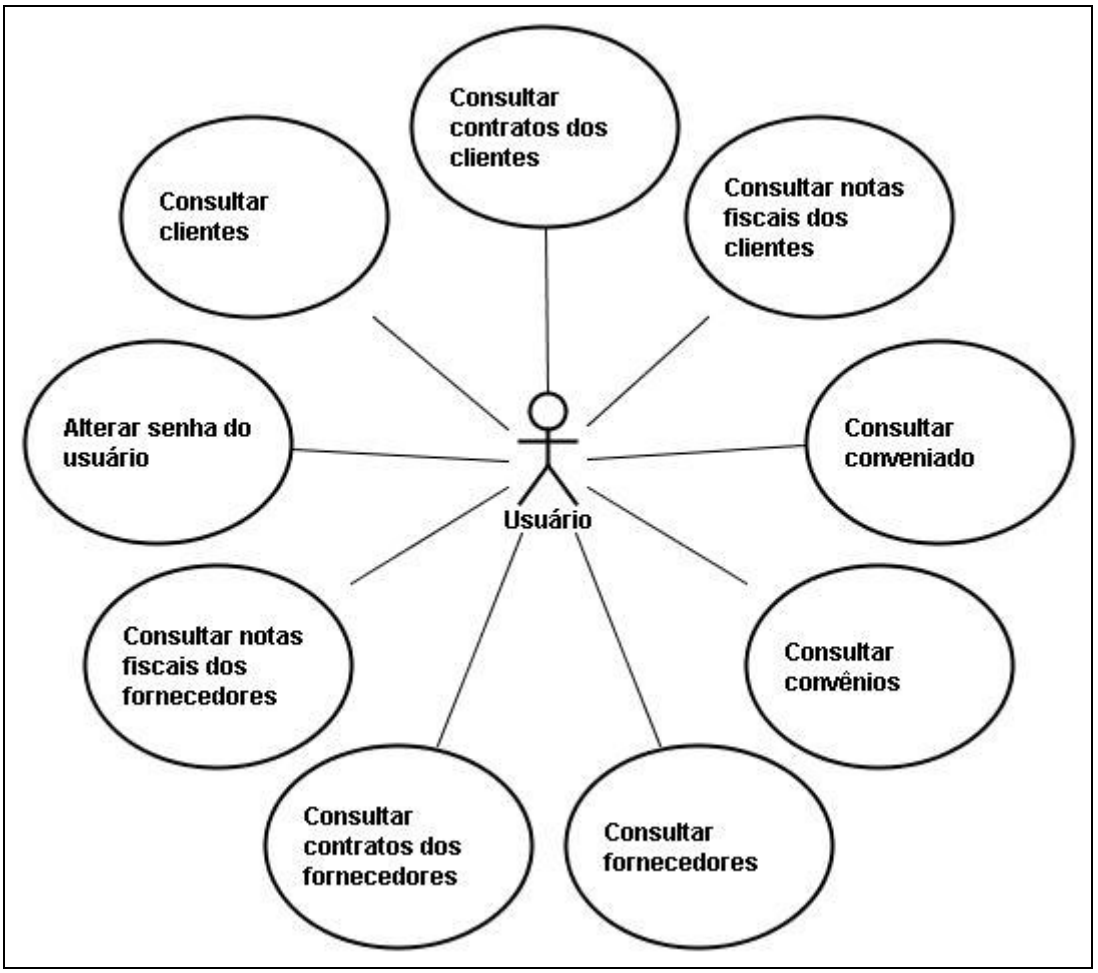

FIGURA 12: Diagrama de casos de uso.

Fonte: Pesquisa direta.

Os casos de uso detalhados abaixo fornecem uma visão do procedimento que o cliente almeja sobre os principais requisitos funcionais do sistema:

TABELA 2: Caso de uso consultar cliente.

Nome:	Consultar cliente
Ator:	Usuário
Pré-condição:	1) Está logado no sistema.
Pós-condição:	1) Consulta realizada.
Fluxo básico:	1) Acessar a opção de consulta de cliente. 2) Escolhe o tipo da consulta (Se é por nome ou CPF/CNPJ). 3) Fornecer o nome ou o CPF/CNPJ do cliente. 4) Sistema processa e mostra o resultado.

Fonte: Pesquisa direta.

TABELA 3: Caso de uso consultar contrato dos clientes.

Nome:	Consultar contrato dos clientes
Ator:	Usuário
Pré-condição:	1) Está logado no sistema.
Pós-condição:	1) Consulta realizada.
Fluxo básico:	1) Acessar a opção de consulta de contrato dos clientes. 2) Escolhe o tipo da consulta (Se é por contrato vigente, contrato encerrado ou contrato a vencer). 3) Fornecer um intervalo de datas para filtrar apenas os contratos que estejam neste intervalo. 4) Sistema processa e mostra o resultado.

Fonte: Pesquisa direta.

TABELA 4: Caso de uso consultar nota fiscal dos clientes.

Nome:	Consultar nota fiscal dos clientes
Ator:	Usuário
Pré-condição:	1) Está logado no sistema.
Pós-condição:	1) Consulta realizada.
Fluxo básico:	1) Acessar a opção de consulta de nota fiscal dos clientes. 2) Escolhe o cliente. 3) Escolhe o tipo da consulta (Se é por nota fiscal paga, em aberto ou cancelada). 4) Fornecer um intervalo de datas para filtrar apenas as notas que possui data neste intervalo. 5) Sistema processa e mostra o resultado.

Fonte: Pesquisa direta.

TABELA 5: Caso de uso alterar senha do usuário.

Nome:	Alterar senha do usuário
Ator:	Usuário
Pré-condição:	1) Está logado no sistema.
Pós-condição:	1) Consulta realizada.
Fluxo básico:	1) Acessar a opção de alterar senha. 2) Fornece a senha atual. 3) Fornece a nova senha 4) Confirma a nova senha. 5) Sistema mostra mensagem de senha alterada com sucesso.
Fluxo Excepcional	3) Senha atual incorreta 4) Sistema mostra mensagem de senha invalida

Fonte: Pesquisa direta.

Os casos de uso de consultar fornecedor e consultar conveniado são iguais com o caso de uso consultar cliente, portanto não será mostrado, levando em consideração que só muda onde tem cliente para fornecedor ou conveniado. Os casos de uso consultar contratos de fornecedor e consultar convênio são iguais ao consultar contrato dos clientes, e para finalizar o consultar nota fiscal de fornecedor é igual ao consultar nota fiscal de cliente.

4.3 Projeto

Com os requisitos e os casos de uso definidos, sabendo o que tem que ser feito, começa aqui à parte de projeto do SC2 que mostra a solução lógica que atende aos requisitos pré-estabelecidos.

4.3.1 Arquitetura do sistema

Utilizando a abordagem de pacotes, a arquitetura do SC2 foi organizada como pode ser vista na FIGURA 13.

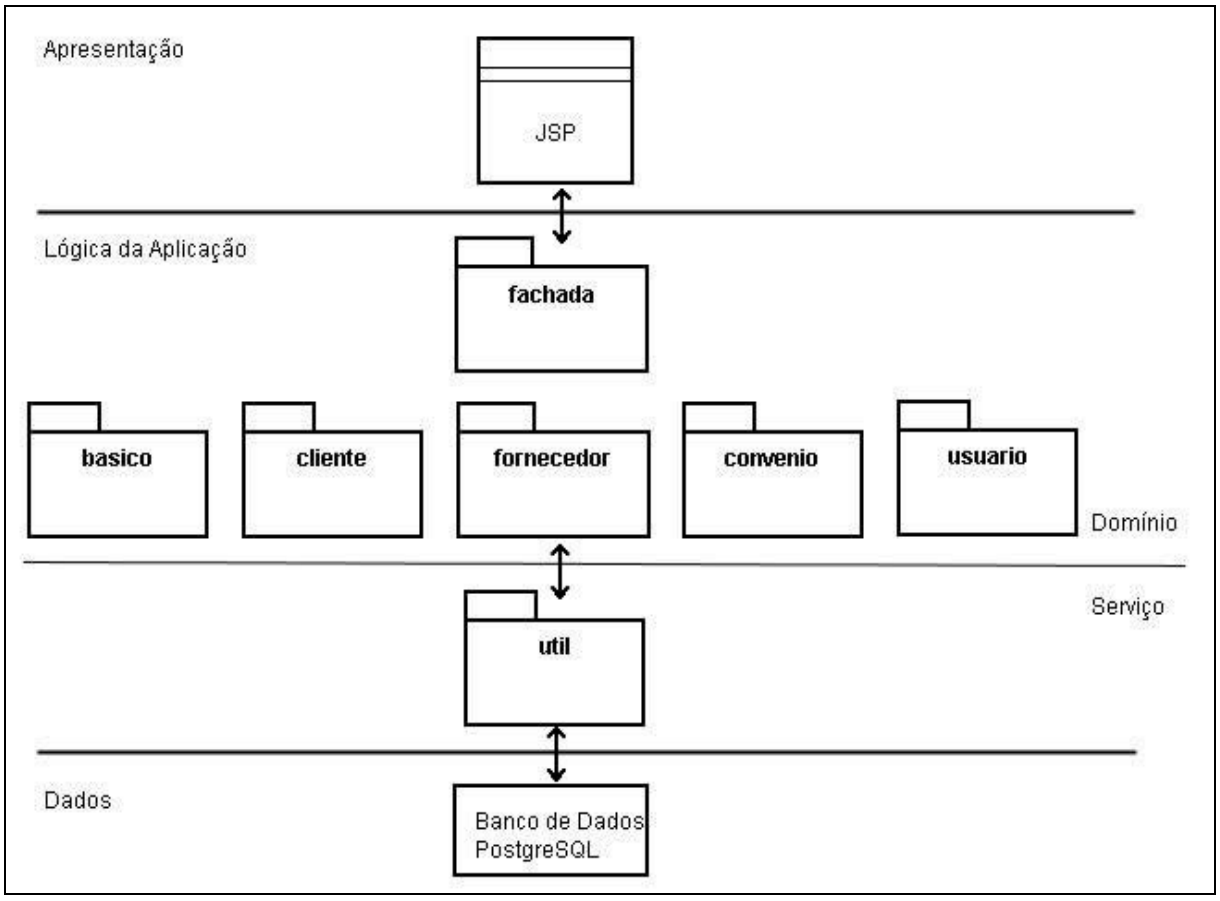

FIGURA 13: Arquitetura do SC2.

Fonte: Pesquisa direta.

A camada de apresentação corresponde aos JSPs que são responsáveis pela interação homem-máquina e por mostrar os resultados das consultas. Essa camada se comunica com a camada de domínio através do pacote fachada. Na camada de domínio estão os pacotes que recebem os dados do banco e passam para a camada de apresentação ou da apresentação para o banco, esta troca de informação é feita também pelo pacote fachada. O pacote útil é o responsável por realizar a comunicação com o banco, gerar os XMLs e os PDFs. E para finalizar a arquitetura a camada de dados corresponde ao banco de dados objeto relacional PostgreSQL.

4.3.2 Diagrama de classe

Para entender melhor a lógica da aplicação a FIGURA 14 mostra o diagrama de classe do SC2.

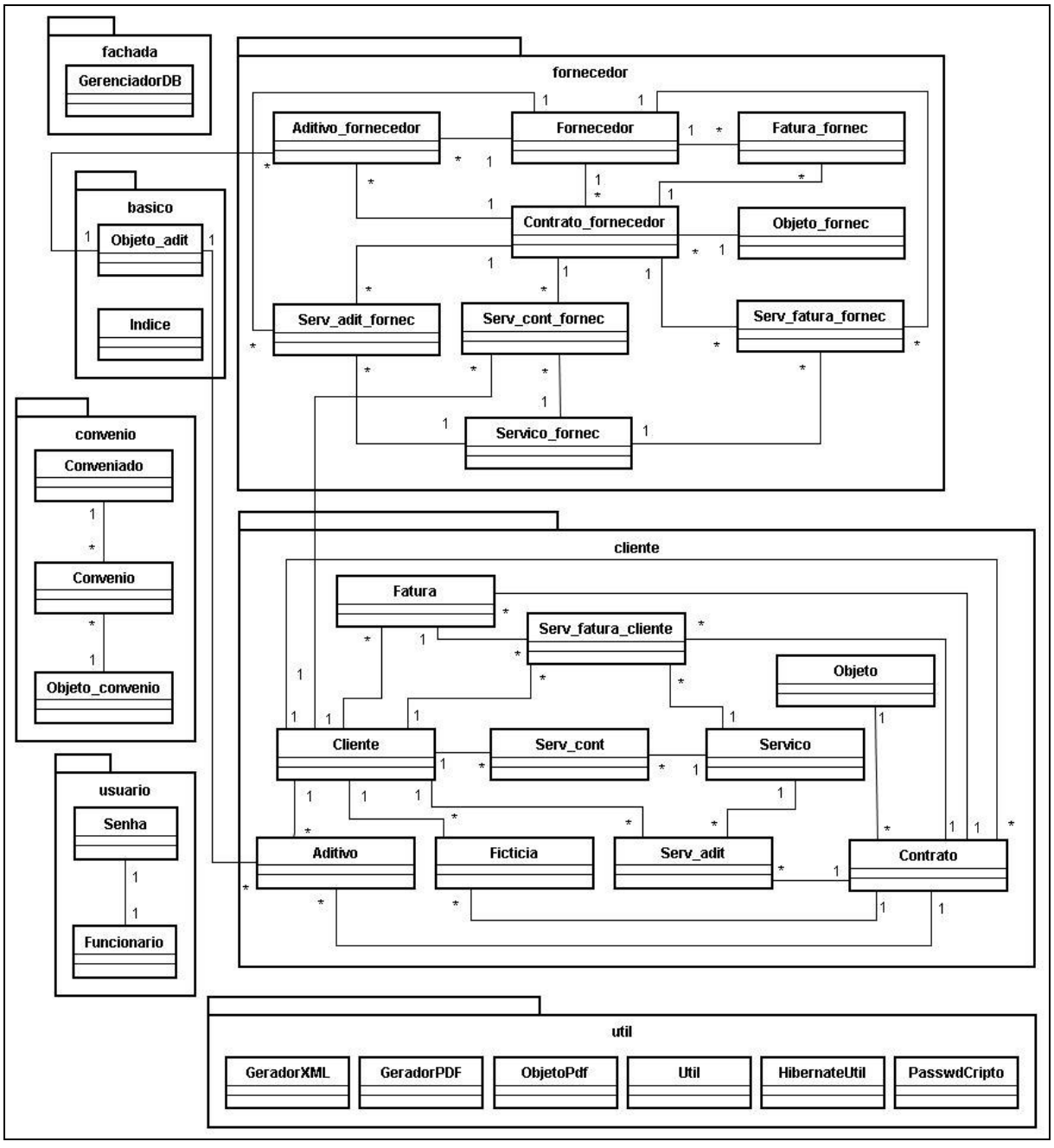

FIGURA 14: Diagrama de classe SC2.

Fonte: Pesquisa direta.

Os pacotes cliente, fornecedor, convênio, usuário e básico possuem as classes persistentes, onde cada classe é associada a uma tabela do banco de dados através de JavaBeans (classes Java que possuem apenas métodos *getters* e *setters* para seus atributos).

No pacote útil, a classe útil possui apenas alguns métodos estáticos que servem para a formatação de texto como datas e números. A classe HibernateUtil é a responsável por fazer a comunicação com o banco de dados. A classe PasswdCripto é responsável pela encriptação das senhas. A classe GeradorXML utiliza o *framework* Castor para transformar objetos em documentos XML. Já a classe GeradorPDF recebe um arquivo XML e uma folha de estilo no formato XSL[8], utilizando o processador de formatação de objetos chamado FOP[9] gera um PDF[10] com o estilo pré-defino do XSL e os dados do XML.

4.3.3 Diagrama de seqüência

Para demonstrar a troca de mensagens dos objetos do SC2 será mostrado um diagrama de seqüência. Este diagrama mostra a troca de mensagem que todas as consultas realizam. Primeiro a tela JSP chama o método consultar da classe GerenciadorDB passando como parâmetro o nome da consulta e em alguns casos um *array* de parâmetros contendo os valores para realizar a consulta, a classe GerenciadorDB chama o método getSession da classe HibernateUtil que retorna uma Session que será através desta que a classe GerenciadorDB abre uma transação com o banco através do método beginTransaction da classe Session, depois de abrir uma transação chama-se o método getNamedQuery que recebe como o

[8] http://www.w3.org/Style/XSL/
[9] http://xmlgraphics.apache.org/fop/
[10] http://www.adobe.com/products/acrobat/adobepdf.html

parâmetro o nome da consulta, este método retorna um *list* de *object* contendo o resultado da consulta. Este *list* é percorrido através de um *iterator* que gera uma pagina HTML através do JSP. Um exemplo da troca de mensagem de uma consulta é mostrado no diagrama de seqüência apresentado na FIGURA 15.

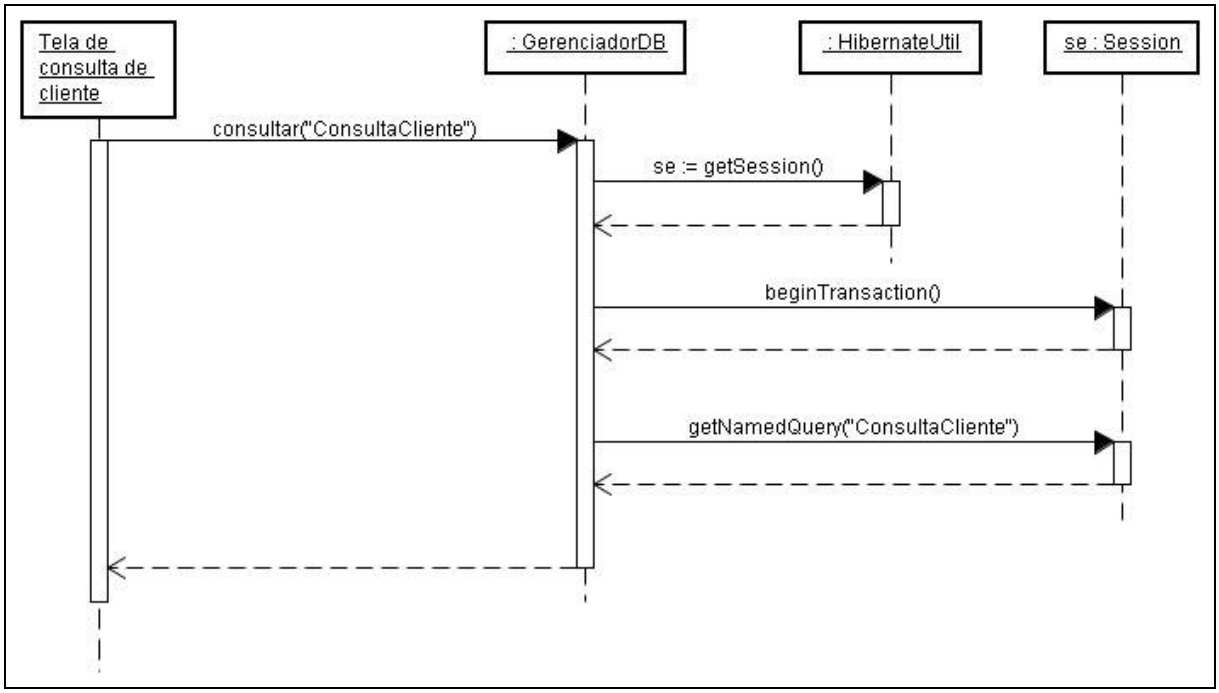

FIGURA 15: Diagrama de seqüência para consulta de cliente.

Fonte: Pesquisa direta.

Os diagramas de seqüência para as outras consultas são muitos parecidos com o diagrama da FIGURA 15, mudando apenas o nome da consulta.

4.3.4 Modelo do banco de dados

O modelo de banco de dados mostrado na FIGURA 16 foi desenvolvido realizando uma engenharia reversa no banco de dados do SGC.

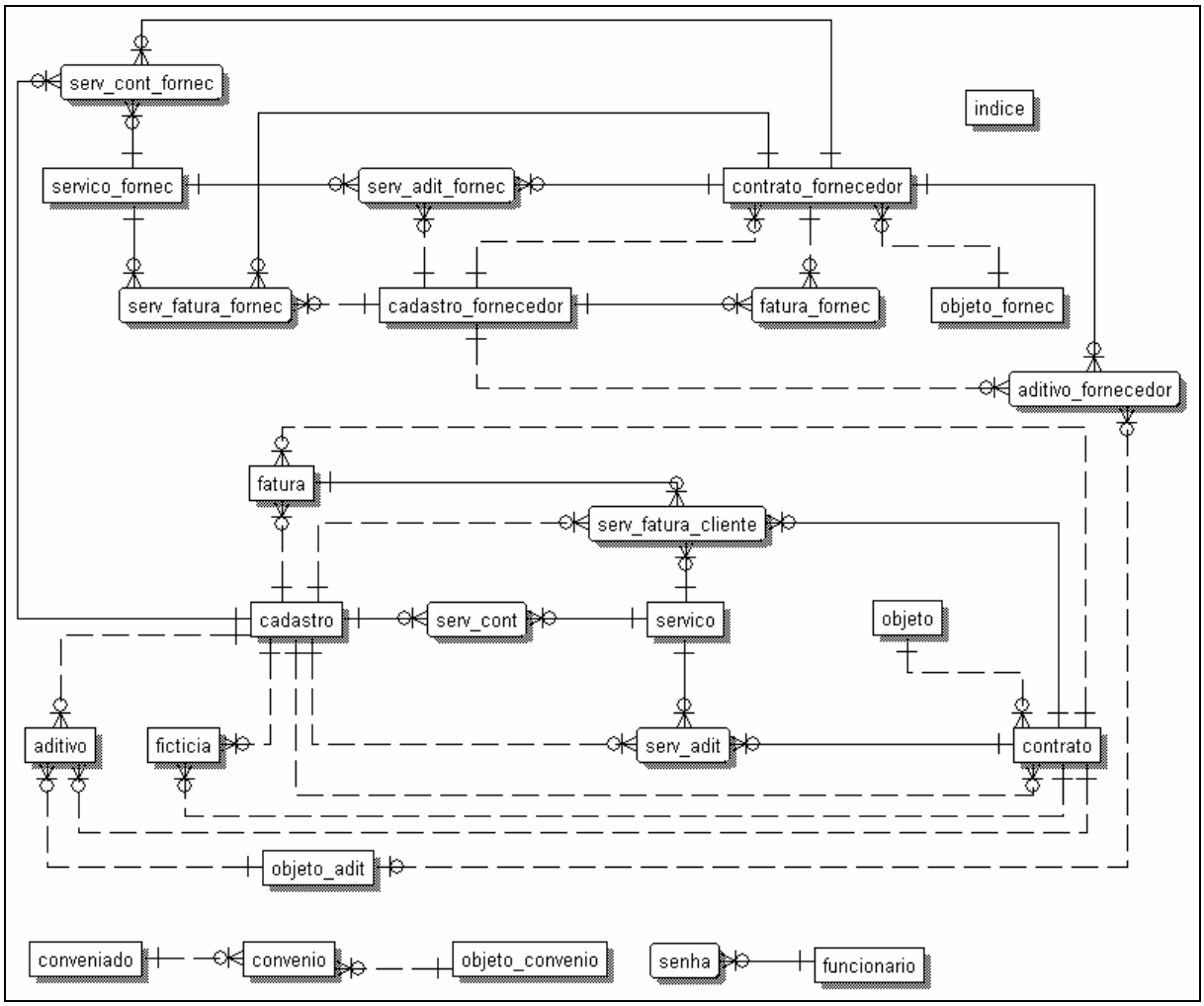

FIGURA 16: Modelo relacional do banco de dados utilizado pelo SC2.

Fonte: Pesquisa direta.

4.4 Implementação

Nesta seção serão abordados os principais aspectos relacionados à implementação do sistema.

4.4.1 Tecnologia utilizada

O SC2 foi desenvolvido para ser acessada via *web*, portanto para isso foi necessário usar a plataforma J2EE de Java. A plataforma J2EE é uma especificação para servidores de aplicações que define padrão de suporte a componentes e serviços, sendo constituída de um pacote de APIs e ferramentas.

Para que uma aplicação J2EE possa ser executa esta precisa estar implantada em um container. O container é a interface entre o componente e as funções de baixo nível do sistema operacional onde a aplicação é executada, sendo também o responsável por chamar os métodos que controlam o ciclo de vida dos componentes.

No SC2 o container de aplicação utilizado foi o Apache Tomcat[11]. O Tomcat é um container de Servlets utilizado na referência oficial da implementação das tecnologias Java Servlets e JavaServer Pages.

Como a aplicação está dividida em múltiplas camadas, serão mostradas as tecnologias utilizadas em cada camada do SC2.

[11] http://tomcat.apache.org/

- **Apresentação:**
    - **JavaServer Pages (JSP):** é uma tecnologia padrão, baseada em *templates* para Servlets. Utiliza paginas de texto contendo código Java embutido, sendo compilado após a instalação ou durante a execução. Utilizado no SC2 para gerar todas as telas da aplicação.
    - ***HyperText Markup Language*** **(HTML)[12]:** é uma linguagem de marcação de texto usada para formatação de pagina *web*, bem como adicionar elementos de multimídia e interação com o usuário. Utilizado no SC2 em conjunto com as paginas JSP para gerar todas as telas da aplicação.
    - ***Cascading Style Sheets*** **(CSS)[13]:** linguagem que cuida exclusivamente da aparência das paginas, usada para definir folhas de estilos para formatar todas as paginas do SC2.
    - **JavaScript:** é uma linguagem de script, utilizada para manipular componentes de paginas na Internet, onde o código é interpretado diretamente pelo *browser* em tempo de execução no computador do cliente. Utilizado no SC2 para fazer a validação de campos dos formulários.
    - ***Portable Document Format*** **(PDF):** é uma especificação disponível publicamente usada por entidades de padronização do mundo inteiro para a distribuição e a troca mais seguras e confiáveis de documentos eletrônicos. No SC2 o PDF foi utilizado para servir como formato de visualização e impressão de relatórios.

[12] http://www.w3.org/MarkUp/
[13] http://www.w3.org/Style/CSS/

- **Domínio:**
    - **Servlets:** são programas Java pré-compilados, executados em um container *web*, que processam requisições HTTP e devolvem respostas HTTP de qualquer tipo. No SC2, Servlets foi utilizado como controlador segurança da aplicação, disponibilizando acesso apenas para quem possui.
    - **JavaBeans:** também conhecidos como *Beans*, são componentes de software que são projetados para serem unidades reutilizáveis. Os *Beans* são utilizados no SC2 para implementação das classes persistentes da aplicação.
- **Serviço:**
    - **XML:** como já mencionado no capitulo 2, XML é uma maneira de representar informações, no SC2 ele esta sendo utilizado para fazer o mapeamento do banco de dados e dos *Beans* que fornecem informações para gerar relatórios.
    - **Castor XML[14]:** é um *framework* para fazer o processo de representar as informações de um documento XML em forma de objeto e de objeto para documento XML. No SC2 utilizou-se Castor XML para transformar objetos em documentos XML.
    - **XSL:** é uma família de linguagens que serve para formatar e apresentar um documento XML. A linguagem XSL utilizada no SC2 foi a XSL-FO que é a linguagem de formatação de documentos XML, foi utilizado para gerar folhas de estilo que será utilizado pelo FOP.
    - **FOP:** é um formatador de folhas de estilos XSL para gerar documentos PDF. O FOP foi utilizado no SC2 para transformar os documentos XML gerado pelo Castor em um documento PDF.

[14] http://www.castor.org/

- **Hibernate:** foi utilizado para mapear classes Java em tabelas do banco de dados, facilitando assim as operações de inserção, remoção, alteração e de consulta entre a aplicação SC2 e o banco de dados.

- **Dados:**
  - **PostgreSQL:** é um sistema gerenciador de banco de dados objeto-relacional (SGBDOR) de código aberto. Este SGBDOR foi utilizado na implementação do banco de dados do SGC, portanto o SC2 realiza neste SGDBOR todas as transações necessárias.

4.4.2 Ferramentas utilizadas

As ferramentas utilizadas no desenvolvimento do SC2:

- **Modelagem:**
  - **Jude Community 2.4:** é uma ferramenta *free* para apresentar a modelagem orientada a objetos de um sistema que utiliza os diagramas da UML, esta ferramenta deu suporte a todos os diagramas gerados nas fases de levantamento dos requisitos e projeto.
  - **ERwin 4.0:** é uma ferramenta visual para construir e manter banco de dados, utilizada para gerar o modelo do banco de dados do SC2.

- **Apresentação:**
  - **Macromedia Dreamweaver MX:** é uma ferramenta para desenvolvimento *web*, permitindo uma facilidade na hora de desenvolver, manter o design de uma aplicação *web*, foi utilizado no SC2 como suporte para a geração das paginas JSPs.
  - **Adobe Photoshop CS:** ferramenta criativa e inovadora para edição de imagens, permitindo ao SC2 qualidade nas imagens.
  - **Adobe *Illustrator* CS:** ferramenta para criação de gráficos vetoriais que oferece mais liberdade de criação permitindo o desenvolvimento de imagens de forma rápida e avançada. Utilizado para a criação de todas as imagens pertencentes ao SC2.
- **Lógica da aplicação:**
  - **Netbeans 4.1:** ambiente de desenvolvimento integrado (IDE) com suporte a aplicações Java. Utilizado para o desenvolvimento de todo o código Java existente no SC2.
- **Dados:**
  - **EMS PostgreSQL *Manager* 3:** é uma poderosa ferramenta de administração e desenvolvimento do banco de dados PostgreSQL. Utilizada na administração do banco de dados da aplicação SC2.

4.5 Exemplo de uso do SC2

Nesta seção será demonstrado o funcionamento do SC2, reproduzindo a interface utilizada e as principais características do funcionamento do sistema. Na FIGURA 17 é

mostrada a tela de autenticação do sistema, onde é necessário fornecer o nome do usuário e a sua respectiva senha para poder ter acesso ao sistema.

FIGURA 17: Tela de autenticação do SC2.

Fonte: Pesquisa direta.

Após realizado a autenticação, a tela mostrada na FIGURA 18 será exibida. Essa é a tela principal do sistema, onde encontrasse o menu de acesso a todas as funcionalidades do sistema.

FIGURA 18: Tela principal do SC2.

Fonte: Pesquisa direta.

Ao acessar o menu usuário e a opção alterar senha será mostrada a tela representada pela FIGURA 19, que corresponde à alteração da senha do usuário, para realizar esta operação basta fornecer a senha atual, a nova senha, confirmar a nova senha e pressionar o botão alterar, qualquer erro será mostrada uma mensagem avisando.

FIGURA 19: Tela de alterar senha de usuário.

Fonte: Pesquisa direta.

A FIGURA 16 mostra a tela com o resultado de uma consulta de convênio, infelizmente não foram mostrados os dados, pois a empresa não permite a divulgação dos dados pertencentes ao banco de dados. As telas dos outros tipos de consultas são muito similar à tela da FIGURA 20.

FIGURA 20: Tela de consulta de convênio.

Fonte: Pesquisa direta.

Para obter os relatórios das consultas basta clicar no botão versão impressa localizado no canto inferior direito das consultas. Ao pressionar o botão abre imediatamente uma nova janela do *browser* contendo o documento PDF resultante da consulta conforme mostra a tela da FIGURA 21.

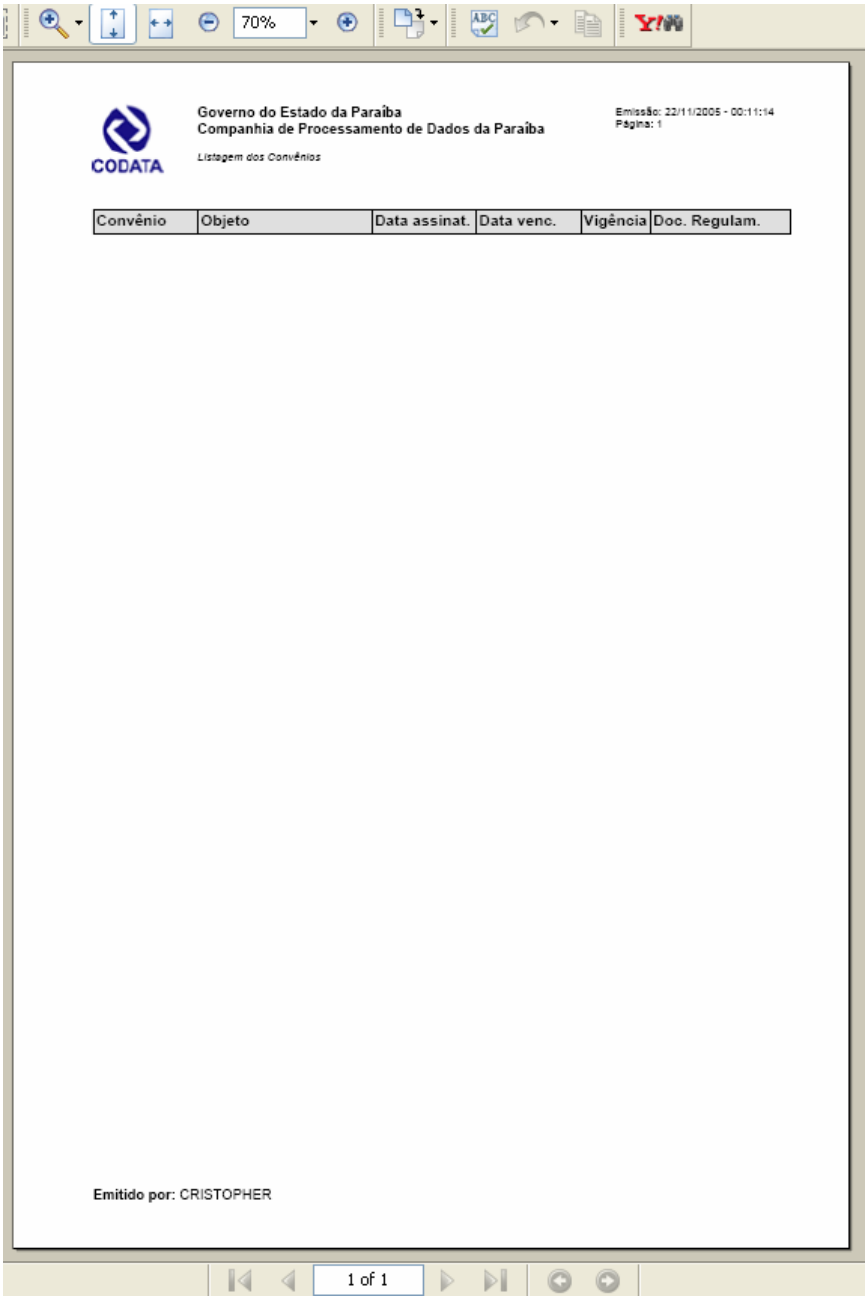

FIGURA 21: Tela de relatório dos convênios.

Fonte: Pesquisa direta.

4.6 Considerações finais

Neste capítulo aspectos inerentes a todo o desenvolvimento de um protótipo para um módulo *web* foram desenvolvidos. Mostrando o que realmente foi necessário para a análise dos requisitos, projeto e implementação, e ainda mostrando o funcionamento do protótipo do SC2.

O SC2 infelizmente não foi concluído impossibilitando assim a sua implantação. O motivo de sua não conclusão foi à necessidade de desenvolver um sistema com maior prioridade para a CODATA tornando assim impossível de concluir o SC2 e de implantá-lo. Portanto o SC2 ficou apenas como protótipo para que futuramente alguém possa terminar.

CAPÍTULO 5

SISTEMA PARA A UNIÃO

Neste capítulo será apresentado à especificação do sistema que esta sendo desenvolvido para o jornal A União, onde serão descritos todos os artefatos necessários para a compreensão deste sistema.

A União, solicitante do sistema, sofre com o problema da falta de informatização dos seus setores, portanto pediu para a CODATA um sistema que integrasse o faturamento de quatro setores (faturamento, gráfica, publicidade, assinatura), resolvendo assim o problema de dados redundantes e do trabalho manual realizado por ele até hoje em dia.

O sistema basicamente é um sistema de cadastro de faturamento de três setores: gráfica, publicidade e assinatura, facilitando o trabalho do funcionário do faturamento que pelo sistema recebe estes dados e dá o rumo certo para que o devido faturamento seja realizado com sucesso.

O sistema possui cinco usuários, um para cada setor, mais o administrador do sistema, onde cada um possui nível diferente de acesso.

5.1 Processo de desenvolvimento

O processo de desenvolvimento utilizado para este sistema foi descrito na seção 3.1, sendo basicamente uma mescla de metodologia ágil (XP) e processo unificado (RUP), utilizando o mesmo processo de desenvolvimento do SC2.

5.2 Levantamento dos requisitos

Para chegar as reais necessidades do sistema o levantamento dos requisitos foi obtido através de diversas entrevistas com os usuários e *stackholder* do sistema.

5.2.1 Requisitos funcionais

A TABELA 6 mostra os requisitos funcionais propostos para o sistema da União.

TABELA 6: Lista de requisitos funcionais do sistema para A União.

Identificação	Descrição
RF01	Cadastro de usuário
RF02	Cadastro de produto
RF03	Cadastro de forma de pagamento
RF04	Cadastro de portador
RF05	Cadastro de agência
RF06	Cadastro de contato
RF07	Cadastro de Cliente
RF08	Cadastro de publicidade
RF09	Cadastro de assinatura
RF10	Cadastro de orçamento gráfico
RF11	Gera nota fiscal
RF12	Confirma pagamento
RF13	Fecha lote
RF14	Baixa agenciador

Fonte: Pesquisa direta.

5.2.2 Requisitos não-funcionais

No Sistema para a união foram abordados os seguintes requisitos não-funcionais:

- **Facilidade de uso:** implementado com tecnologia que permite um simples acesso aos recursos do sistema através do mouse e teclado.

- **Segurança:** controle dos usuários através de uma tela de login/senha, com senhas criptografadas.
- **Portabilidade:** o sistema foi implementado com tecnologia que permite a implantação do sistema na maioria dos sistemas operacionais.
- **Interface:** dispõe de uma interface gráfica amigável e de fácil aprendizado.

5.2.3 Atores

Para este sistema foram necessários cinco tipos diferentes de atores, estes atores serão descritos logo abaixo:

- **Design:** ator que representa o setor da gráfica, responsável pelo cadastro dos orçamentos gráficos e dos clientes da gráfica.
- **Vendedor:** ator que representa o setor de assinatura, responsável pelo cadastro das assinaturas e dos clientes do setor de assinatura.
- **Publicitário:** ator que representa o setor de publicidade, responsável pelo cadastro das publicidades e dos clientes do setor de publicidade.
- **Faturamento:** ator que representa o setor de faturamento, responsável por gerar notas fiscais para os setores de publicidade, assinatura e gráfica, pelo confirmamento dos pagamentos das notas e do controle de pagamento dos agenciadores.
- **Administrador:** ator que faz tudo no sistema, alem de fazer o que os atores acima faz, ele é responsável pelo cadastro de usuário, produto, forma de pagamento, portador, agencia e contato.

5.2.4 Casos de uso

Na FIGURA 22 será mostrado o diagrama de casos de uso que oferece uma visão gráfica das funcionalidades do sistema.

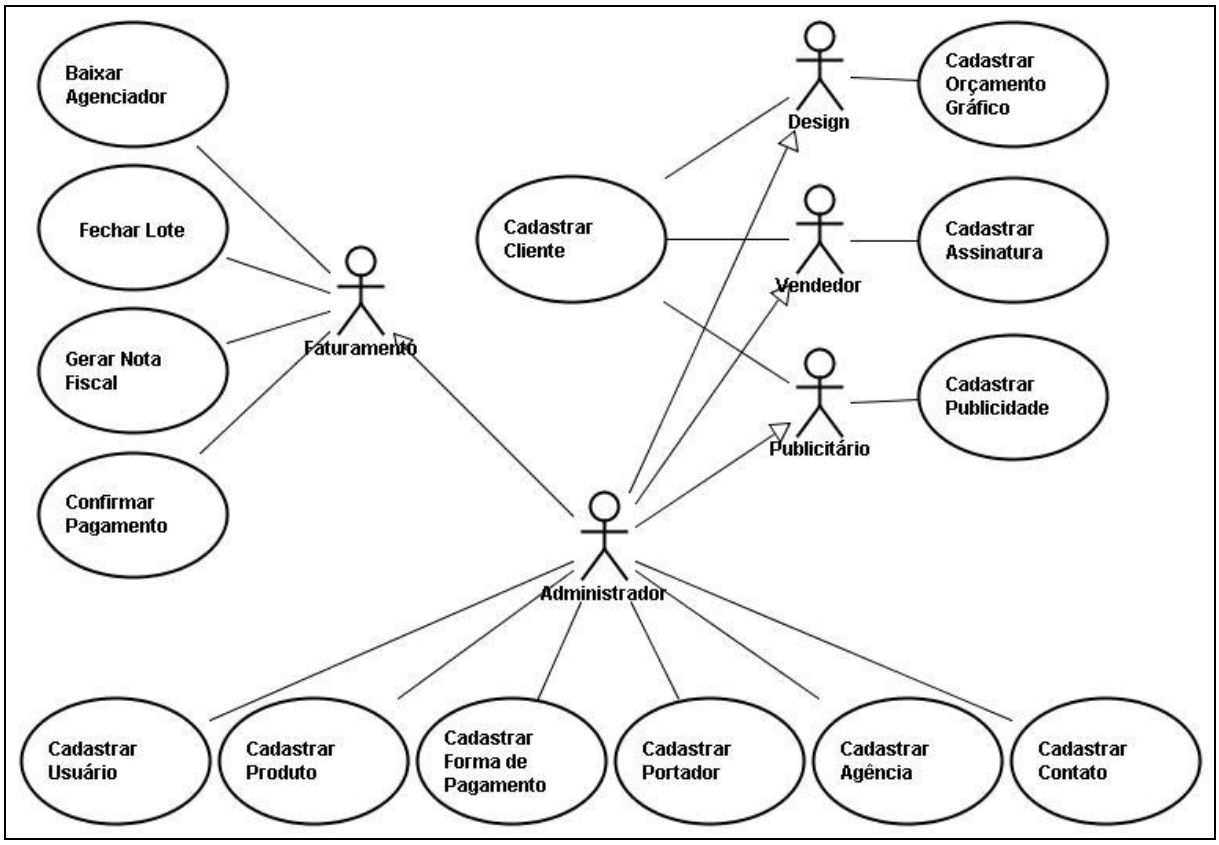

FIGURA 22: Diagrama de casos de uso do sistema para A União.

Fonte: Pesquisa direta

Os casos de uso detalhados abaixo fornecem uma visão do procedimento que o cliente almeja sobre os principais requisitos funcionais do sistema:

TABELA 7: Caso de uso cadastrar usuário.

Nome:	Cadastrar usuário
Atores:	Administrador
Pré-condição:	2) Está logado no sistema.
Pós-condição:	2) Usuário cadastrado com sucesso. 3) Usuário alterado com sucesso. 4) Operação cancelada.
Fluxo básico:	5) Acessar a opção de cadastro de usuário. 6) Fornecer o número da matricula do usuário. 7) Sistema verifica se o usuário já existe. 8) Fornecer os dados do usuário (nome, login, senha, grupo). 9) Sistema grava os dados.
Fluxo alternativo:	4) Sistema mostra os dados do usuário existente. 5) Alterar os dados do usuário. 6) Sistema grava os dados alterados.

Fonte: Pesquisa direta

TABELA 8: Caso de uso cadastrar forma de pagamento.

Nome:	Cadastrar forma de pagamento
Atores:	Administrador
Pré-condição:	1) Está logado no sistema.
Pós-condição:	1) Forma de pagamento cadastrado com sucesso. 2) Forma de pagamento alterado com sucesso. 3) Operação cancelada.
Fluxo básico:	1) Acessar a opção de cadastro de forma de pagamento. 2) Fornecer a descrição de forma de pagamento. 3) Sistema grava os dados.
Fluxo alternativo:	3) Sistema mostra mensagem de forma de pagamento já existente. 4) Alterar a descrição da forma de pagamento. 5) Sistema grava os dados alterados.

Fonte: Pesquisa direta

TABELA 9: Caso de uso cadastrar produto.

Nome:	Cadastrar produto
Atores:	Administrador
Pré-condição:	1) Está logado no sistema.
Pós-condição:	1) Produto cadastrado com sucesso. 2) Produto alterado com sucesso. 3) Operação cancelada.
Fluxo básico:	1) Acessar a opção de cadastro de produtos. 2) Fornecer os dados do produto (nome, categoria). 3) Sistema grava os dados.
Fluxo alternativo:	3) Sistema mostra os dados do produto já existente. 4) Alterar os dados do produto. 5) Sistema grava os dados alterados.

Fonte: Pesquisa direta

TABELA 10: Caso de uso cadastrar cliente.

Nome:	Cadastrar cliente
Atores:	Administrador, Design, Vendedor, Publicitário, Financeiro.
Pré-condição:	1) Está logado no sistema.
Pós-condição:	1) Cliente cadastrado com sucesso. 2) Cliente alterado com sucesso. 3) Operação cancelada.
Fluxo básico:	1) Acessar a opção de cadastro de clientes. 2) Fornecer o CNPJ ou o CPF do cliente. 3) Sistema verifica se o cliente já existe. 4) Fornecer os dados do cliente (nome) 5) Sistema grava os dados.
Fluxo alternativo:	4) Sistema mostra os dados do cliente já existente. 5) Alterar os dados do cliente. 6) Sistema grava os dados alterados.

Fonte: Pesquisa direta

TABELA 11: Caso de uso cadastrar assinatura

Nome:	Cadastrar assinatura
Atores:	Administrador, Vendedor.
Pré-condição:	1) Está logado no sistema.
Pós-condição:	1) Assinatura cadastrada com sucesso. 2) Operação cancelada.
Fluxo básico:	1) Acessar a opção de cadastro de assinaturas. 2) Fornecer os dados da assinatura (período, valor). 3) Sistema grava os dados.

Fonte: Pesquisa direta

TABELA 12: Caso de uso cadastrar orçamento gráfico.

Nome:	Cadastrar orçamento gráfico.
Atores:	Administrador, Vendedor.
Pré-condição:	1) Está logado no sistema.
Pós-condição:	1) Assinatura cadastrada com sucesso. 2) Operação cancelada.
Fluxo básico:	1) Acessar a opção de cadastro de assinaturas. 2) Fornecer os dados da assinatura (período, valor). 3) Sistema grava os dados.

Fonte: Pesquisa direta

5.3 Projeto

Nesta etapa será mostrada a parte relacionada ao projeto do sistema para A União demonstrando a solução lógica que atende aos requisitos pré-estabelecidos.

5.3.1 Arquitetura do sistema

Para demonstrar a arquitetura do sistema para A União foi utilizando a abordagem de pacotes, como pode ser vista na FIGURA 23.

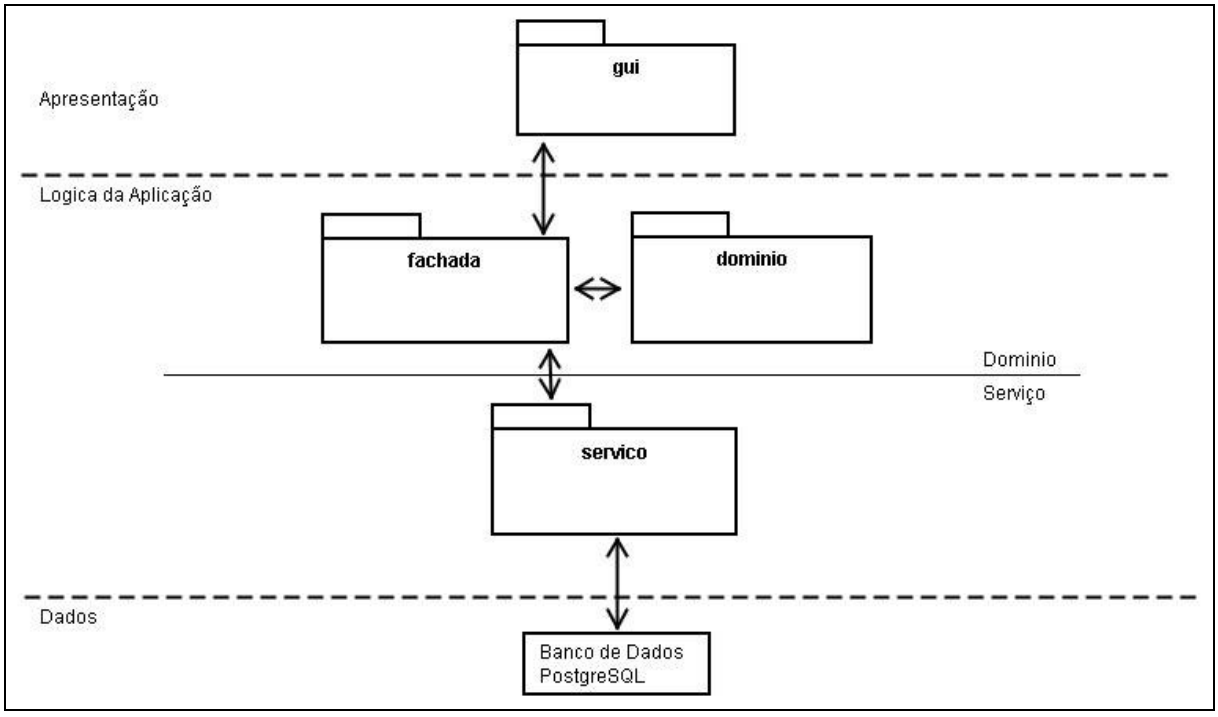

FIGURA 23: Arquitetura do sistema para A União

Fonte: Pesquisa direta.

A camada de apresentação possui o pacote gui que contém todos as classes da interface gráfica, esta camada se comunica com a camada de domínio por meio do pacote fachada. A camada de domínio possui um pacote chamado domínio que tem como objetivo armazenar os dados em memória do banco de dados, ele recebe dados da interface que será armazenado no banco e também recebe dados do banco que será mostrado pela interface, o pacote fachada é o responsável por levar os dados da interface para a camada de serviço, e da camada de serviço para a interface. A camada de serviço possui o pacote serviço que é responsável por fazer a comunicação com o banco de dados. E a camada de dados corresponde ao banco de dados objeto relacional PostgreSQL.

5.3.2 Diagrama de classe

Para entender melhor como a lógica da aplicação funciona a FIGURA 24 mostra o diagrama de classe do sistema para A União.

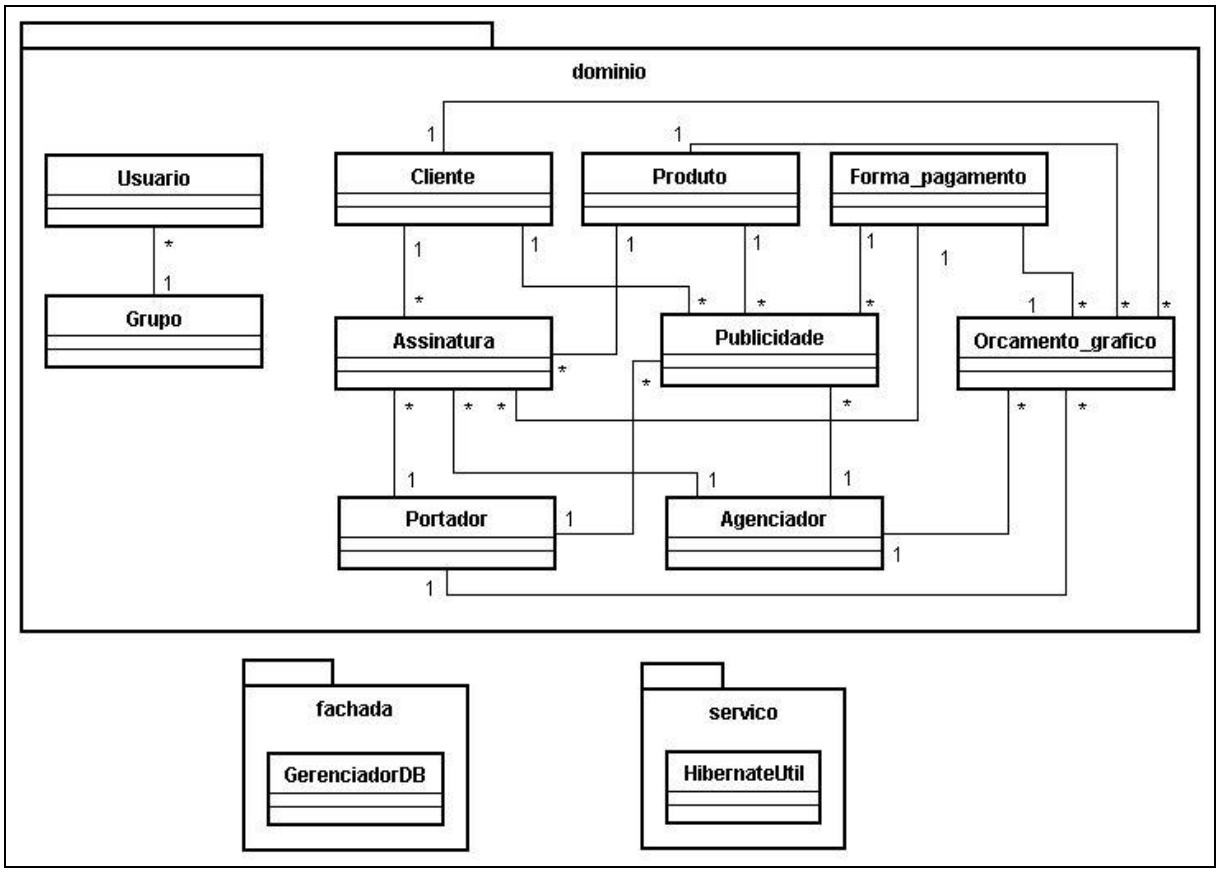

FIGURA 24: Diagrama de classe do sistema para A União

Fonte: Pesquisa direta.

O pacote domínio possui as classes persistentes, onde cada classe é associada a uma tabela do banco de dados através de JavaBeans (classes Java que possuem apenas métodos *getters* e *setters* para seus atributos). O pacote fachada é o responsável por pegar os dados do banco e armazenar em objetos e de armazenar no banco os objetos.

No pacote serviço, a classe HibernateUtil é a responsável por fazer a comunicação com o banco de dados.

5.3.3 Diagrama de seqüência

Para demonstrar a troca de mensagens dos objetos do sistema serão mostrados alguns diagramas de seqüências.

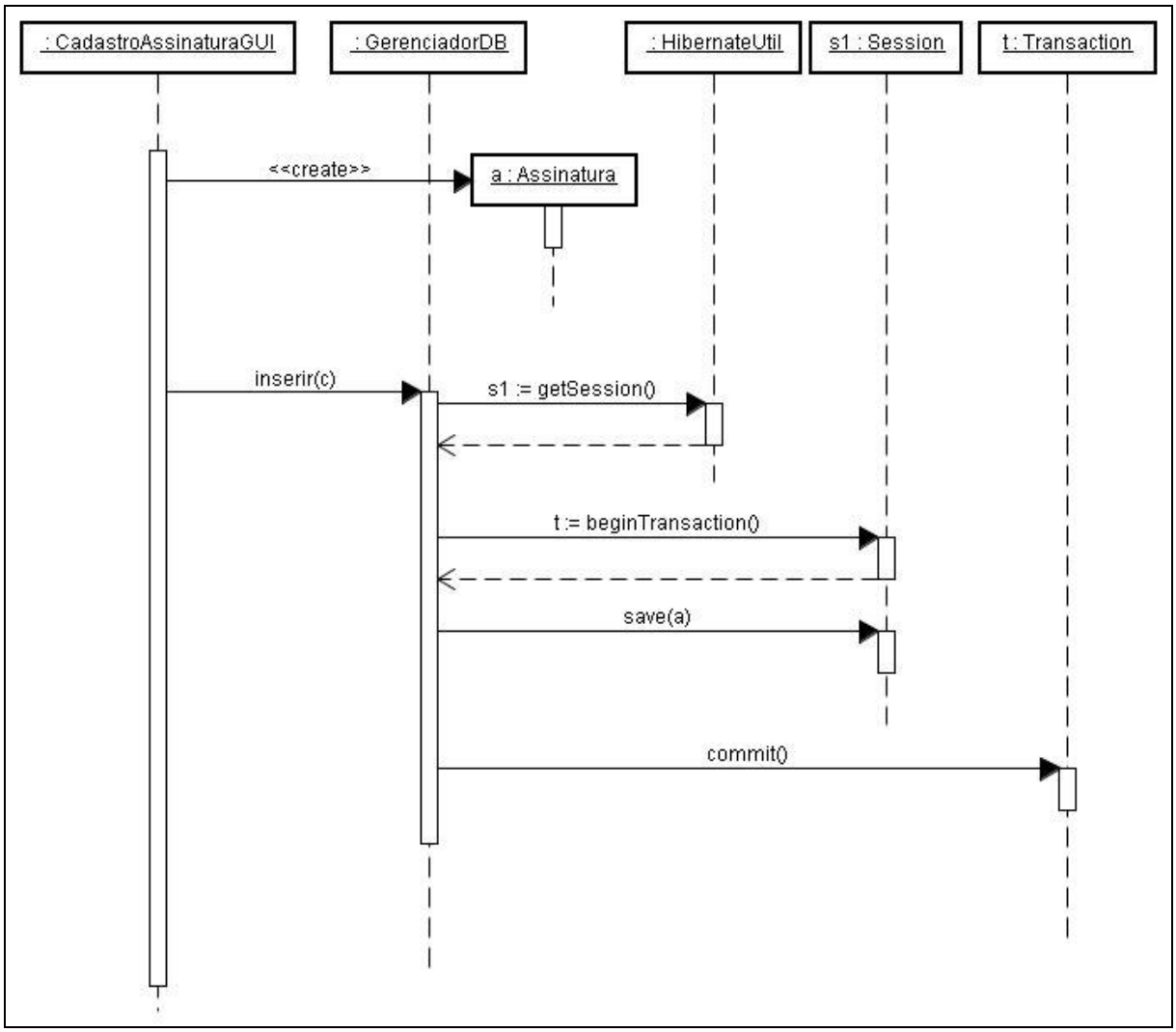

FIGURA 25: Diagrama de seqüência para o cadastro de assinatura

Fonte: Pesquisa direta.

O diagrama de seqüência da FIGURA 25 mostra a troca de mensagem para um cadastro de assinatura. Ao disparar um evento para cadastrar uma assinatura, uma classe Assinatura é criada capturando todos os dados da interface, após isso é chamado o método inserir da classe GerenciadorDB. A classe GerenciadorDB chama o método getSession da classe HibernateUtil que retorna uma Session. Com a Session aberta à classe GerenciadorDB chama o método beginTransaction da classe Session, depois de obter uma transação a salva o objeto no banco chamando o método save e passando como argumento o objeto que será inserido no banco. Para finalizar a transação é chamado o método *commit* da classe Transaction.

Os diagramas de seqüência dos outros tipos de cadastro são muito semelhantes ao diagrama da FIGURA 25.

O diagrama da FIGURA 26 mostra o diagrama para cadastro de cliente, pois esse necessita de uma verificação antes de inserir os dados no banco. Ele é muito semelhante ao da FIGURA 25 possuindo a mais apenas uma consulta no banco que retorna se o cliente já existe ou não caso o cliente já exista o processo de inserção é o mesmo mostrado na FIGURA 25.

.

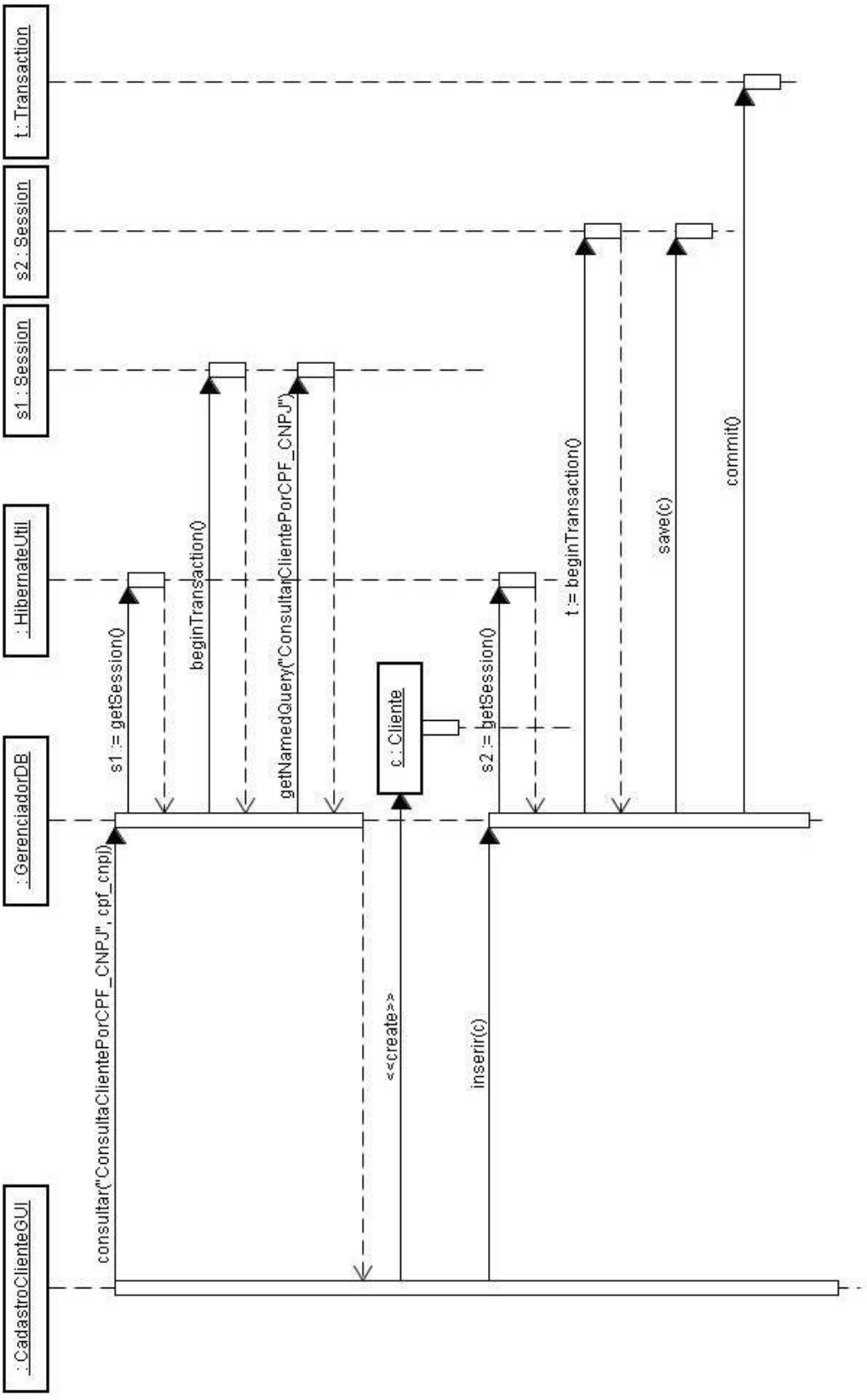

FIGURA 26: Diagrama de seqüência para cadastrar cliente

Fonte: Pesquisa direta.

5.3.4 Modelo do banco de dados

A FIGURA 27 mostra o modelo de banco de dados desenvolvido para o sistema para A União.

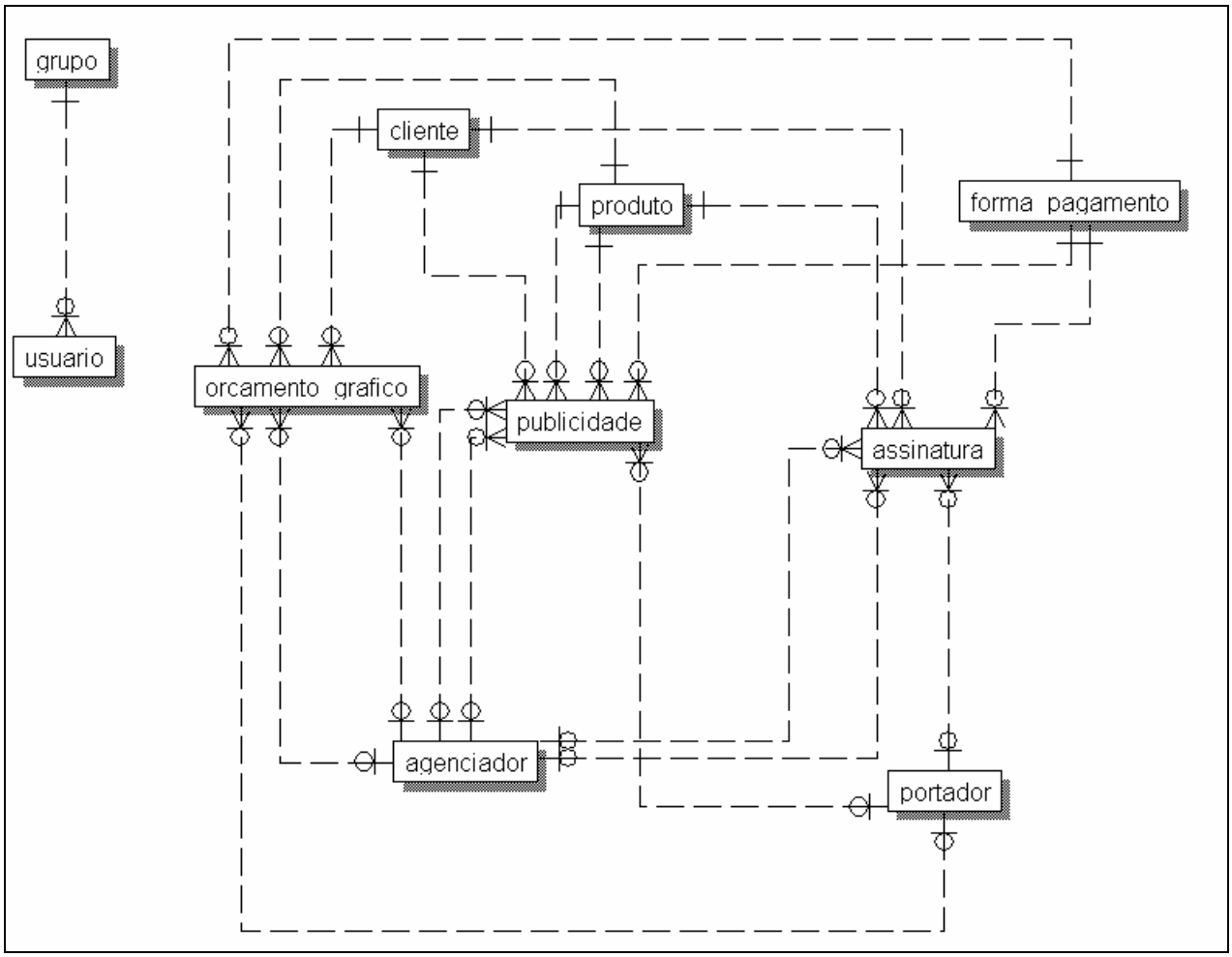

FIGURA 27: Modelo do banco de dados do sistema para A União.

Fonte: Pesquisa direta.

A tabela usuário representa todos os funcionários do jornal A União que utilizara o sistema, onde cada usuário possui um grupo. Para representar os orçamentos gráficos, as

assinaturas e as publicidades foram criadas tabelas para cada tipo de fatura, onde cada uma dessa tabelas possui um cliente, um produto, um agenciador, um portador e uma forma de pagamento.

5.4 Implementação

Nesta seção serão abordados os principais aspectos relacionados à implementação do sistema.

5.4.3 Tecnologia utilizada

O sistema para a união esta sendo desenvolvido como aplicação *desktop*, portanto utiliza a plataforma J2SE de Java. A plataforma J2SE fornece um ambiente completo para o desenvolvimento de aplicações *desktop*.

Como o sistema para a união está dividido em múltiplas camadas, abaixo serão mostradas todas as tecnologias utilizadas em cada camada.

- **Apresentação:**
    - **Swing:** é uma API Java para criação de interfaces gráficas. Usada no sistema da união para implementação das interfaces gráficas.

- **Domínio:**
    - **JavaBeans:** Os JavaBeans são utilizados no sistema da união para implementação das classes persistentes da aplicação.
- **Serviço:**
    - **XML:** Utilizado para fazer o mapeamento do banco de dados.
    - **Hibernate:** É utilizado para mapear classes Java em tabelas do banco de dados, facilitando assim as operações de inserção, remoção, alteração e de consulta entre a aplicação e o banco de dados.
- **Dados:**
    - **PostgreSQL:** É utilizado este SGDBOR para as implementação e transações de banco de dados do sistema.

5.4.2 Ferramentas utilizadas

As ferramentas utilizadas no desenvolvimento do sistema para A União:

- **Modelagem:**
    - **Jude Community 2.4:** Ferramenta deu suporte a todos os diagramas gerados nas fases de levantamento dos requisitos e projeto.
    - **ERwin 4.0:** Ferramenta visual utilizada para gerar o modelo do banco de dados do sistema.
- **Apresentação e lógica de apresentação:**
    - **Netbeans 4.1:** ambiente de desenvolvimento integrado (IDE) com suporte a aplicações Java. Utilizado para o desenvolvimento da aplicação.

- **Dados:**
  - **EMS PostgreSQL Manager 3:** é uma poderosa ferramenta de administração e desenvolvimento do banco de dados PostgreSQL. Utilizada na administração do banco de dados da aplicação.

5.5 Exemplo de uso do sistema para A União

Nesta seção será demonstrado o funcionamento do sistema para A União, reproduzindo a interface utilizada e as principais características do funcionamento do sistema. Na FIGURA 28 é mostrada a tela de autenticação do sistema, onde é necessário fornecer o nome do usuário e a sua respectiva senha para poder ter acesso ao sistema.

FIGURA 28: Tela de autenticação do sistema para A União

Fonte: Pesquisa direta.

Após realizar a autenticação o usuário é encaminhado para a tela principal do sistema, essa possui menu diferente para cada tipo de usuário. A tela principal do sistema é mostrada na FIGURA 29.

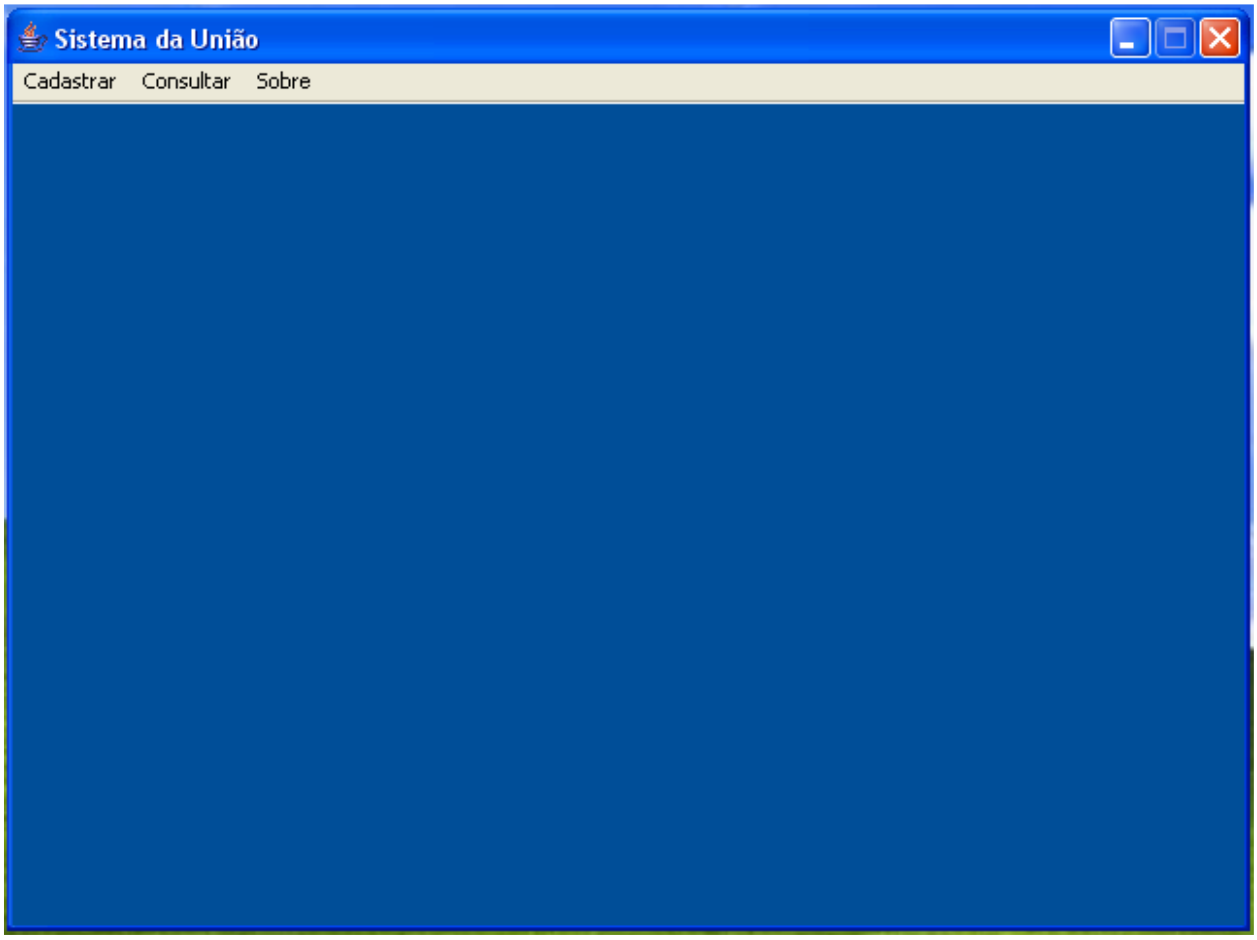

FIGURA 29: Tela principal do sistema para A União

Fonte: Pesquisa direta.

A FIGURA 30 mostra a tela de cadastro de clientes, que pos sua vez representa a modelagem do diagrama de seqüência mostrado na FIGURA 22.

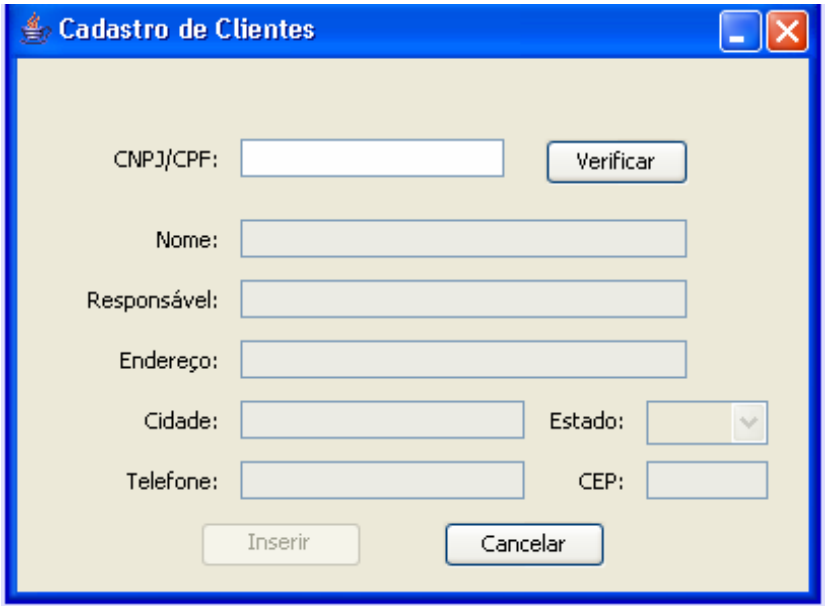

FIGURA 30: Tela de cadastro de clientes do sistema para A União.

Fonte: Pesquisa direta.

A FIGURA 31 mostra a tela de cadastro de assinatura que representa o diagrama da FIGURA 25.

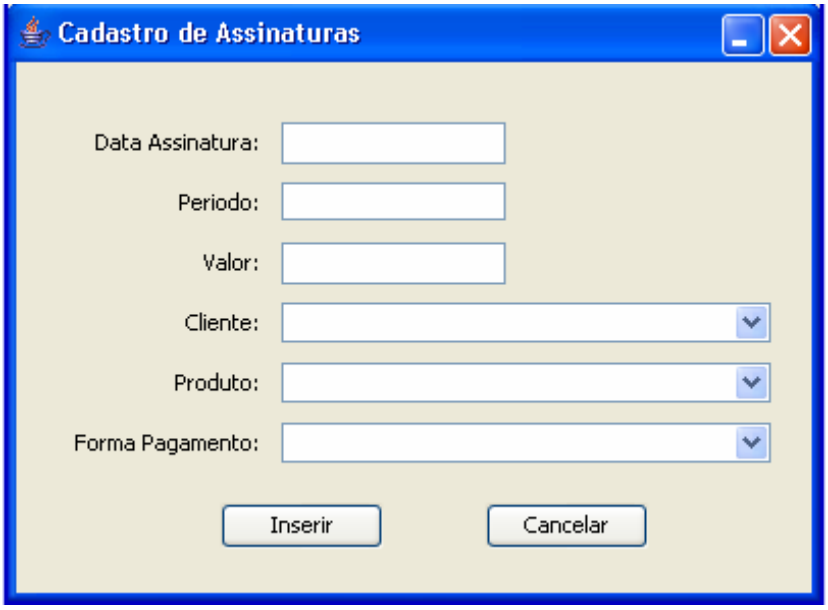

FIGURA 31: Tela de cadastro de assinaturas do sistema para A União.

FONTE: Pesquisa direta.

5.6 Considerações finais

Neste capítulo aspectos inerentes a todo o desenvolvimento de um protótipo do sistema da união foram desenvolvidos. Mostrando o que realmente foi necessário para o levantamento dos requisitos, projeto e implementação, e ainda mostrando o funcionamento do protótipo do sistema da união.

O sistema da união continua em desenvolvimento pela CODATA.

CONCLUSÃO

A elaboração deste relatório junto com a realização do estágio supervisionado necessitou muito além do que foi visto nas salas de aula durante o curso. Mas essa necessidade levou a procura de novos meios para o entendimento do que foi necessário para realizar estas duas tarefas. Além destas dificuldades o tempo e as decisões internas da CODATA foram os maiores empecilhos.

O principal objetivo deste trabalho foi à definição de uma metodologia que permitisse a criação e documentação rápida de um sistema, esta metodologia foi feita utilizando uma mescla do Processo Unificado (RUP) com uma metodologia ágil (XP). Para demonstrar essa metodologia foram mostrados dois sistemas que utiliza essa metodologia, o Sistema de Contratos e Convênios e o sistema para A União.

O Sistema de Contratos e Convênio (SC2) foi desenvolvido com o objetivo de ajudar os gerentes e diretores da CODATA, permitindo que visualizasse os dados dos contratos e convênios existentes de qualquer lugar onde houvesse um computador conectado a Internet. Esse sistema foi importante para introduzir o conhecimento de diversas tecnologias, melhorando o domínio da arte de programar, mas infelizmente por decisões internas da CODATA esse sistema não chegou a ser concluído sendo interrompido faltando pouco para ser implantada a primeira versão aberta aos usuários.

O sistema para A União foi um sistema que o jornal A União solicitou, para integrar o faturamento do jornal. Esse sistema utilizou a maioria dos conceitos aprendidos durante o desenvolvimento do SC2 tornando o desenvolvimento deste sistema um pouco mais simples. Esse sistema também não chegou a ser implantado por falta de tempo.

Conclui-se que estas atividades de elaboração do relatório e do estagio supervisionado foi de grande enriquecimento para elaboração de uma personalidade mais madura, trazendo vivência de mercado e domínio dos conceitos não só relacionado à informática.

6.1 Trabalhos futuros

Devido às decisões internas da CODATA o SC2 não foi completamente implementado, faltando os filtros para as consultas, por isso não chegando a ser implantado. Como trabalho futuro do SC2 fica apenas a implementação dos filtros.

O sistema para A União por conta do curto período de tempo este sistema ficou apenas em protótipo, faltando à implementação de algumas funcionalidades principalmente na parte do faturamento onde nada foi implementado. Como trabalho futuro do sistema para A União sugere-se que termine a implementação da parte de faturamento e dar uma melhor interface gráfica para o usuário, ou seja, tornando a interface gráfica mais bonita, pois a interface que o sistema possui atualmente foi desenvolvida apenas como protótipo faltando passar por especialistas em design.

REFERÊNCIAS

AUER, Ken; MILLER, Roy. **Extreme Programming Applied:** Playing to Win. Indianapolis: Addison Wesley, 2002.

BECK, Kent. **Extreme Programming Explained:** Embrace Change. Indianapolis: Addison Wesley, 2000.

DEITEL, Harvey; DEITEL, Paul. **Java Como Programar.** Porto Alegre: Bookman, 2003.

DEITEL, Harvey; DEITEL, Paul; NIETO, Tem; LIN, Ted; SADHU, Praveen. **XML Como Programar.** Porto Alegre: Bookman, 2003.

FOWLER, Martin; SCOTT, Kendall. **UML Essencial:** Um breve guia para a linguagem-padrão de modelagem de objetos. 2. ed. São Paulo: Bookman, 2000.

HIGHTOWER, Richard; ONSTINE, Warner; VISAN, Paul; PAYNE, Damon; GRADECKI, Joseph. **Professional Java Tools for Extreme Programming.** Indianapolis: Wiley, 2004.

HORSTMANN, Cay; CORNELL, Gary. **Core Java 2:** Volume I – Fundamentos. São Paulo: Makron Books, 2001.

LARMAN, Craig, **Utilizando UML e Padrões:** Uma introdução à análise e ao projeto orientado a objetos e ao Processo Unificado. 2. ed. São Paulo: Bookman, 2002.

OLIVEIRA, Toacy. **Tecnologias de Reutilização (Framework).** Disponível em: <http://www.inf.pucrs.br/~toacy/disciplinas/pos/msi-2005-I/aula3.pdf>. Acesso em: 17/11/2005.

PAULA FILHO, Wilson de Pádua. **Engenharia de Software:** fundamentos, métodos e padrões. Rio de Janeiro: LTC, 2001.

PENDER, Tom. **UML a Bíblia.** Rio de Janeiro: Editora Campos, 2004.

PRESSMAN, Roger S. **Engenharia de Software**. 3. ed. São Paulo: Makron Books, 1995.

SOMMERVILLE, Ian. **Engenharia de Software.** 6. ed. São Paulo: Addison Wesley, 2003.

TELES, Vinícius Manhães. **Extreme Programming:** Aprenda como encantar seus usuários desenvolvendo software cm agilidade e alta qualidade. São Paulo: Novatec, 2004.