UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO

Relatório do Estágio Integrado

Desenvolvimento e integração de sistemas de software coorporativo de notificação de eventos a partir de provisão de dados utilizando Web Services e SQL

Signove Tecnologia S/A

Romeryto Vieira Lira
Estagiário – UFCG

Hyggo Oliveira de Almeida
Orientador Acadêmico

Glauber Vinícius Ventura de Melo Ferreira
Supervisor Técnico

Campina Grande - Paraíba
Dezembro de 2010

Biblioteca Setorial do CDSA. Maio de 2021.

Sumé - PB

Desenvolvimento de sistemas de software coorporativo de notificação de eventos, integração de dados e provimento de dados utilizando *Web Services* e SQL

Aprovado em _____

Banca Examinadora

Prof. Hyggo Oliveira de Almeida
Orientador Acadêmico

Prof. Petrônio Carlos Bezerra
MEMBRO DA BANCA

Prof. Joseana Macêdo Fechine
MEMBRO DA BANCA

Sumário

Sumário de figuras

Sumário de Quadros

Agradecimentos

Agradeço, primeiramente, a ***Deus*** pela oportunidade de todo dia poder ganhar conhecimentos novos.

Em seguida, agradeço a todos que fazem parte da *Signove Tecnologia* pelo apoio durante todo esse período de grande aprendizado.

Também agradeço ao Professor *Hyggo Almeida* por ter me proporcionado a oportunidade de trabalhar no *Embedded*, trabalho este que abriu as portas da *Signove* para que assim eu pudesse executar o estágio.

E, por fim, agradeço a minha família por todo apoio que tem me dado, e a grande oportunidade de estudar no *Curso de Ciência da Computação da UFCG (Universidade Federal de Campina Grande)*, longe de minha cidade de origem.

Apresentação

Com base nas exigências da disciplina Estágio Integrado do Curso de Ciência da Computação da UFCG, apresenta-se o relatório de estágio de **Romeryto Vieira Lira**, cujo objetivo foi o desenvolvimento de dois sistemas coorporativos distintos.
Os dois sistemas são citados brevemente a seguir:

- **Sistema 1:** Sistema de integração de dados e notificação de eventos(SIDNE);
- **Sistema 2:** Sistema provedor de dados(SPD).

O Relatório tem seu conteúdo dividido da seguinte forma:

Seção 1 - Introdução;
Seção 2 - Objetivos;
Seção 3 - Informações gerais sobre o estágio;
Seção 4 - Fundamentação Teórica;
Seção 5 - Atividades Realizadas;
Seção 6 - Atividades no SIDNE;
Seção 7 - Atividades no SPD;
Seção 8 - Considerações Finais;
Seção 9 – Bibliografia;
Seção 10 – Apêndices.

As siglas SIDNE e SPD são usadas para denominar os sistemas por motivos de confidencialidade.

Resumo

O Estágio foi realizado na empresa Signove Tecnologia S/A e teve como objetivo a execução de dois projetos. O primeiro trata-se de um sistema de integração de dados e notificação de eventos. O segundo sistema é responsável por prover dados para o primeiro sistema de modo que este possa efetuar a integração de dados entre os mesmos. Os dois sistemas comunicam-se entre si a partir de acesso aos seus bancos de dados e também via *Web Services*.

1. Introdução

É vital para um estudante ter experiência o quanto antes no mercado de trabalho. Poder participar da disciplina Estágio Integrado é uma oportunidade que oferece ao aluno uma grande experiência, pois coloca o mesmo em contato com a realidade da empresa, podendo trabalhar com pessoas mais preparadas, com grande bagagem de conhecimento técnico em computação.

Também um grande aprendizado é obtido no campo de gerência de projetos, pois desempenhar atividades seguindo um processo de desenvolvimento claro e preciso ajuda ao aluno abrir seu campo de visão com relação a como se planejar e otimizar seu tempo durante a execução de tarefas.

Outro ponto fundamental do estágio é a chance de aprender novas tecnologias e a partir disso ganhar uma efetiva experiência profissional.

O estágio foi realizado na empresa Signove Tecnologia S/A. O mesmo teve como objetivo prover comunicação entre dois sistemas.

Tal comunicação foi desenvolvida via *Web Services*, que permitem a comunicação entre os dois sistemas de modo que os mesmos possam trocar informações e interagir entre si.

2. Objetivos

Foram executadas atividades relativas a dois sistemas, nas subseções seguintes detalharemos os objetivos de cada um.

2.1. Objetivos do Sistema de integração de dados e notificação de eventos (SIDNE)

Objetivo geral:

Prover um sistema se comunique com um banco de dados remoto para obter dados necessários para o suporte à funcionalidade de notificação de eventos fazendo assim uma *integração de dados* (Integrar dados é quando um sistema extrai e transforma dados de uma ou mais fontes (*podem ser sistemas*) de informação diferentes, para uso próprio) com o SPD.

Objetivos específicos:

- Aprender as tecnologias necessárias para o desenvolvimento do sistema;
- Desenvolver o sistema com a tecnologia *Grails* [GRAILS];
- Desenvolver atividades na área de integração de dados com o banco de dados remoto (do segundo sistema) via *SQL* [SQL];
- Desenvolver métodos que possibilitem a notificação de eventos para entidades do sistema;
- Desenvolver o *Web Service* [WEB SERVICE] que provê métodos para notificação de eventos;
- Desenvolver a interface web do sistema.

2.2. Objetivos do Sistema provedor de dados (SPD)

Objetivo geral:

Desenvolver um sistema que se comunica com o SIDNE promovendo, assim, a integração entre os mesmos através de provisão de dados(está ligada diretamente a disponibilização de dados). Temos como objetivo geral a construção de páginas Web para acionar as ações de notificação de eventos do primeiro sistema.

Objetivos específicos:

- Aprender as tecnologias necessárias para desenvolvimento do sistema;
- Desenvolver o sistema com a tecnologia *Grails*;
- Prover acesso ao banco de dados para que assim o primeiro sistema ter acesso aos dados;
- Desenvolver métodos consumidores para o *Web Service* do primeiro sistema;
- Construir páginas web para o sistema de modo que estas possam disparar os métodos providos pelo *Web Service* do SIDNE.

3. Informações gerais sobre o Estágio

3. 1. Informações do estagiário

Nome: Romeryto Vieira Lira

Matrícula: 20521156

Endereço residencial: Rua João Julião Martins, Nº 413, bairro Universitário

CEP: 58429-100

Cidade: Campina Grande – PB.

E-mail: romeryto.lira@signove.com

Telefones: (83) 9967-8733

(83) 9649-5554

3.2. Orientador Acadêmico

Nome: Hyggo Oliveira de Almeida

Endereço: Laboratório de Sistemas Embarcados e Computação Pervasiva, Departamento de Engenharia Elétrica, Centro de Engenharia Elétrica e Informática, Universidade Federal de Campina Grande - Av. Aprígio Veloso, 882, Bodocongó, 58109-970, Campina Grande – PB, Brasil.

Email: hyggo@dsc.ufcg.edu.br

3.3. Supervisor Técnico

Nome: Glauber Vinicius Ventura de Melo Ferreira

Endereço: Signove Tecnologia S/A. Dom Pedro II, Nº 675, bairro da Prata, 58400-565, Campina Grande, Paraíba.

Email: glauber.ferreira@signove.com

3.4. A Empresa

O estágio foi realizado na empresa Signove Tecnologia S/A, a qual está localizada na Rua Dom Pedro II, Nº 675, bairro da Prata, 58400-565, Campina Grande, Paraíba.

A Signove foi fundada por ex-alunos de mestrado e doutorado da UFCG. A maioria destes trabalhava no Laboratório de Sistemas Embarcados e Computação Pervasiva – Embedded.

A Signove é dividida atualmente em dois setores, o administrativo que é composto por Sandro Alves e Joab Pachêco. O setor de desenvolvimento é formado por sócios da empresa, funcionários contratados, estagiários e bolsistas de projetos. Também fazem parte do setor de desenvolvimento alunos do curso de Desenho Industrial da UFCG, eles são responsáveis por tudo relacionado ao design gráfico das aplicações construídas na Signove.

3.5. Ambiente de trabalho

A Signove possui um excelente ambiente de trabalho, gerando assim motivação em quem trabalha na empresa. A mesma tem um amplo laboratório de desenvolvimento e pesquisa, salas de reuniões com equipamentos para vídeo conferência, copa, recepção, sala de servidores, sala de entretenimento para os funcionários, sala de servidores e sala de diretoria. A sala de desenvolvimento é equipada com computadores, de uso individual, agrupados em ilhas para facilitar a interação dos grupos.

3.6. Unidade de estágio

O estágio foi realizado no setor de Desenvolvimento. O mesmo é composto por várias "ilhas" compostas por postos de trabalho organizados de acordo com a alocação de cada pessoa aos seus projetos específicos.

3.7. Período de estágio

O estágio teve início em 02/08/2010 e terminou no dia 07/12/2010.

3.8. Jornada Semanal/Total em horas do estágio

O estágio teve uma jornada semanal de 30hs, ao todo contabilizando 552 horas.

3.9. Equipe

A equipe do projeto foi composta pelos seguintes profissionais:

- **Glauber Ferreira:** Sócio da empresa e mestre em Ciência da Computação, é o *Product Owner* do projeto e supervisor técnico na empresa do meu estágio.
- **Kyller Gorgônio:** Sócio da empresa e mestre em Ciência da Computação. É o *SCRUM Master* do projeto.

- O estagiário trabalhou como o único desenvolvedor do projeto.

4. Fundamentação Teórica

Nesta seção, descrevemos a metodologia que foi utilizada durante a execução dos projetos no estágio. O processo de desenvolvimento utilizado foi o SCRUM [SCRUM]. Aqui será descrito como o mesmo funciona e a forma com que são delegadas e conduzidas as tarefas através deste processo.

Também nesta seção são apresentadas as tecnologias utilizadas, bem como o detalhamento de cada uma para assim poder-se ter uma idéia geral de como elas foram empregas durante as atividades.

4.1. Metodologia de desenvolvimento

A metodologia de desenvolvimento das atividades foi o SCRUM. O SCRUM apresenta-se como uma metodologia ágil e flexível, que tem por objetivo definir um processo de desenvolvimento iterativo. Esta metodologia baseia-se no desenvolvimento incremental das aplicações, centrado na equipe com ciclos de iteração curtos, cada ciclo desse é chamado de *Sprint*.

A metodologia SCRUM apenas estabelece conjuntos de regras e práticas de gestão que devem ser adotadas para garantir o sucesso de um projeto. Centra-se no trabalho em equipe, visando melhorar a comunicação e maximizar a cooperação. Englobando processos de engenharia, este método não requer nem fornece qualquer técnica ou método específico para a fase de desenvolvimento de software.

As fases de desenvolvimento SCRUM podem ser divididas basicamente em três. São elas:

- **Planejamento:** Definição de uma nova funcionalidade requerida pelo sistema baseado no conhecimento do sistema como um todo;
- **Desenvolvimento:** Desenvolvimento dessa nova funcionalidade respeitando o tempo previsto, requisitos exigidos e qualidade. Esses itens definem o fim do ciclo de desenvolvimento;
- **Encerramento:** Preparação para a entrega do produto persistindo as atividades de testes, documentação do usuário e treinamento, caso seja necessário.

No estágio o SCRUM foi utilizado de uma forma adaptada, pois o estagiário foi o único desenvolvedor do projeto. O que diferenciou foi que as reuniões eram feitas quando uma determinada funcionalidade especificada era concluída. Esta era avaliada pelo *Product Owner,* e após isso eram alocadas novas tarefas para o estagiário.

4.1.1. Atores do SCRUM

No SCRUM os três principais atores são:

***Product Owner* (Dono do produto):** É quem determina as diretrizes e requisitos junto ao cliente do projeto. É responsável por definir tudo que deve ser implementado no projeto.

Scrum Master: Responsável por gerenciar a equipe de desenvolvimento, acompanhar o andamento dos trabalhos, manter a produtividade, proporcionar um ambiente amistoso no projeto, e garantir que tudo seja entregue dentro do prazo.

Equipe: Composta pelos desenvolvedores do projeto. Os mesmos têm autonomia para organizar e distribuir atividades entre seus membros da forma mais viável.

4.1.2. Funcionamento do SCRUM

Definição do *Backlog*: Todas as funcionalidades do produto são definidas pelo *Product Owner* e descritas no *Product Backlog*. Esta lista é organizada de acordo com prioridades que são definidas a partir de aspectos relativos aos requisitos do cliente, demanda do mercado ou até mesmo à capacidade e experiência que a equipe de desenvolvimento já tem. Os itens são divididos para serem entregues em parcelas, ou seja, cada bloco dividido em cada *Sprint*.

Acompanhamento do Sprint: Durante o *Sprint* as tarefas definidas para o mesmo no *Product Backlog* vão para o *Sprint Backlog*. A partir daí a equipe

deve dividir de forma responsável as atribuições de cada membro bem como as tarefas que são mais adequadas para cada um. A equipe tem autonomia para dividir as tarefas.

Reuniões Diárias: O *Scrum Master* diariamente acompanha o andamento do Sprint. Todo dia acontece uma rápida reunião para se avaliar os seguintes aspectos:

- Saber quais são as atividades concluídas no dia anterior;
- Quais atividades serão iniciadas no dia atual;
- Quais os problemas que estão atrapalhando o andamento das tarefas.

Reuniões de Revisão e Planejamento: Ao final do *Sprint* é feita uma reunião para revisar as tarefas que foram concluídas e avaliar se as mesmas atenderam aos objetos previamente estabelecidos. Após isso é feita uma reunião de planejamento para o próximo *Sprint*.

4.2. Infra-estrutura de desenvolvimento

Durante o desenvolvimento das atividades, as ferramentas usadas foram na sua maioria *Open Source*.

Abaixo são listadas as principais ferramentas:

- Eclipse [ECLIPSE];
- Java 6 [JAVA 6];
- NetBeans IDE [NETBEANS];
- MySQL [MYSQL];
- SVN [SVN];
- Entre outras tecnologias que serão detalhadas mais a frente.

A infra-estrutura de *hardware* (computador) utilizada foi a seguinte:

- Processador Core 2 Duo E7400 2.8 GHZ;
- Memória 4 GB DDRII;
- 320 GB de HD;
- Placa de rede 10/100/1000 Mbps.

4.3. Tecnologias

4.3.1. Grails

Grails é um *framework* de desenvolvimento ágil e dinâmico, semelhante ao *Rails* [RAILS] e *Django* [DJANGO], os quais inspiraram uma nova e moderna forma de pensar e construir aplicações web.

O mesmo foi idealizado com o objetivo de reduzir drasticamente a complexidade da construção de aplicações coorporativas para web, especialmente na plataforma Java, tecnologia esta para a qual existem *frameworks* que são muito complicados que não seguem a filosofia de desenvolvimento rápido de aplicações. *Grails* é um *framework* de desenvolvimento web completo e utiliza-se de uma infra-estrutura flexível permitindo aos desenvolvedores criarem *plugins* que provêem mais funcionalidades e enriquecem a plataforma.

O ambiente *Grails* pode ser caracterizado da seguinte forma:

- Fácil de usar, pois tem uma camada com Mapeamento Objeto Relacional (*Object Relational Mapping - ORM*) construída com *Hibernate* [HIBERNATE];
- Utiliza *Groovy* [GROOVY] como Linguagem de programação;
- Utiliza-se da tecnologia expressiva para criação de páginas web chamada *GSP (Groovy Server Pages)* [GSP];
- Tem uma camada controladora construída através do framework *Spring MVC(Model-View-Controller)* [SPRING MVC];
- Possui um ambiente de linha de comando baseado na linguagem *Groovy*;

- Utiliza o *Tomcat* [TOMCAT] embutido para execução da aplicação, simulando o ambiente de produção;
- Suporte à internacionalização (*i18n*);
- Camada de Serviços para abstração da lógica de negócio.

Estes podem ser agrupados da seguinte forma:

- ***development***: Ambiente para o qual configurações são definidas especificamente para serem usadas em fase de desenvolvimento da aplicação;
- ***test:*** Ambiente para o qual configurações são definidas especificamente para serem usadas em fase de testes da aplicação;
- ***production:*** Ambiente para o qual configurações são definidas especificamente para serem usadas quando a aplicação estiver no ar em um ambiente web.

A arquitetura do *Grails* é dividida da forma vista na Figura 1.

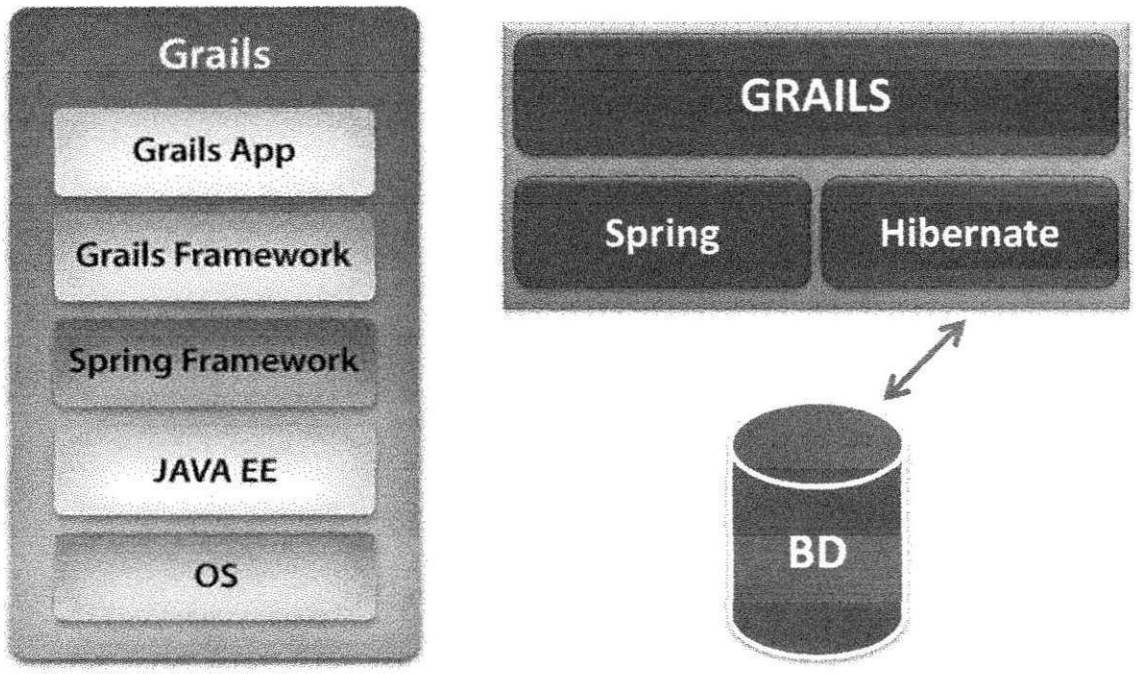


Figura 1 - Arquitetura do Grails.

A aplicação *Grails* acessa o *framework* da tecnologia, que por sua vez comunica-se com o *Spring*, e este em conseqüência comunica-se com a infra-estrutura do *JAVA EE* juntamente com o *Hibernate*, que assim conseguem fazer operações no banco de dados do sistema.

Uma aplicação *Grails* é dotada das seguintes camadas, seguindo o MVC(*Model View Controller*)[MVC]:

- ***Model*** - Camada que contem as Classes de Domínio (*Domain Class*) do sistema. Estas classes definem as entidades e objetos principais do sistema passíveis de serem persistidos, bem como definem os mapeamentos e relacionamentos entre classes;
- ***Service*** - Camada responsável por abrigar os Serviços da aplicação. Os Serviços são classes responsáveis por representar e conter códigos que estão diretamente relacionados à lógica de negócio do sistema;
- ***Controller*** - Esta camada é a porta de entrada das requisições e chamadas feitas pela camada de visualização (*View*);
- ***View*** - Camada composta pelos arquivos que representam as páginas web do sistema (*GSP*). Esta é responsável por obter os dados da tela e enviar as requisições para a Camada *Controller*.

4.3.2. Groovy

Groovy é uma linguagem de programação dinâmica para a Máquina Virtual Java (*JVM*). A mesma foi inspirada nas características das linguagens *Python* [PYTHON], *Ruby* [RUBY] e *Smalltalk* [SMALLTALK].

O objetivo da linguagem *Groovy* é tornar programação mais fácil e rápida, através de uma curva de aprendizado pequena e rápida. A sintaxe da linguagem é simples e fácil de ler, programar e manter. É baseada em linguagens no estilo de *scripts*.

Integra-se perfeitamente com todas as classes e bibliotecas do Java. A compilação gera *bytecodes* Java fazendo com que se possa usar *Groovy* em praticamente qualquer lugar onde você pode usar Java.

4.3.3. Web Services

Web Services é a principal tecnologia utilizada atualmente para integração de sistema distintos. Com esta tecnologia é possível que aplicações feitas em plataformas diferentes possam interagir entre si de forma compatível.

Web Services permite que as aplicações troquem dados entre si através de arquivos *XML* [W3C-XML], facilitando a comunicação entre as mesmas.

4.3.4. SQL

SQL (*Structered Query Language*) ou Linguagem de Consulta Estruturada, é uma linguagem de pesquisa declarativa voltada para bancos de dados relacionais, como também para banco de dados Objeto-Relacionais. A mesma define as estruturas das tabelas do banco de dados bem como provê uma maneira fácil de consulta aos dados destas tabelas, provendo através desta a definição da forma do resultado pretendido, e não como chegar até o resultado. A mesma é caracterizada principalmente pela sua simplicidade, facilidade de uso bem como reduz o ciclo de aprendizagem para aqueles que estão entrando em contato com a tecnologia pela primeira vez.

4.3.5. Hibernate

Hibernate é um *framework* escrito em linguagem Java para o mapeamento objeto-relacional. O *Grails* utiliza o *Hibernate*, convertendo o esquema de mapeamento definido em suas classes de domínio diretamente para a o formato do *Hibernate*, construindo assim as tabelas do banco de dados conforme o definido.

4.3.6. MySQL

O *MySQL* é um sistema de gerenciamento de banco de dados(*SGBD*) que utiliza a linguagem SQL para definir os esquemas dos bancos de dados.É um dos *SGBDs* mais usados no mundo por grandes corporações, a exemplo da *NASA*, *Nokia*, *Banco Bradesco*, *Dataprev*, *Google*, *Sony* entre outras.

O *MySQL* é otimizado para aplicações coorporativas web, multiplataforma e é famoso pela baixa exigência de processador comparado a outros *SGBDs* famosos no mercado.

Dentre as características mais importantes do *MySQL* estão:

- Portabilidade, pois executa praticamente em qualquer Sistema Operacional atual;
- Excelente desempenho e estabilidade;
- Compatibilidade - Pois existem várias bibliotecas *ODBC* [ODBC], *JDBC* [JDBC] e módulos de interface para diversas linguagens de programação;
- Fácil de usar;
- É distribuído com base na licença GPL [GPL];
- Dispões de várias interfaces gráficas intuitivas (*MySQL Toolkit*);
- Suporta controle transacional, *cursors*, *procedures* e *functions*;
- Oferece um sistema de gerência de usuários e senhas com segurança através de criptografia.

4.3.7. Selenium

O *Selenium* [SELENIUM] é um *framework* para testes de interface gráfica disponível para vários ambientes de desenvolvimento, inclusive o *Grails*. O *Selenium* simula bateria de testes clicando automaticamente nos links das páginas das aplicações procurando reproduzir assim a situação real do uso do sistema por um usuário real.

Dentre as vantagens do *Selenium,* pode-se destacar as seguintes:

- Oferece um retorno rápido aos desenvolvedores;
- Gera um relatório personalizado com o resultado dos testes;
- Dá suporte a metodologias ágeis de desenvolvimento;
- Possui uma boa documentação;

- Busca erros para os quais seriam de difícil detecção quando da execução manual da aplicação.

4.3.8. SVN

O *SVN,* conhecido também como *Subversion*, é um sistema open source de controle de versões de arquivos. O gerenciamento dos diretórios e arquivos é feito cronologicamente. Cada versão é guardada no repositório, sendo possível assim obter versões dos arquivos baseadas em data, em marcações, entre outros aspectos. É uma ferramenta que auxilia enormemente o trabalho dos desenvolvedores de sistemas.

5. Atividades Realizadas

Nesta seção, são abordados tópicos relativos às atividades do estágio.

Todas as atividades planejadas foram concluídas com sucesso, bem como as atividades novas, relativas à funcionalidade de notificação de eventos, que surgiram no decorrer do tempo. Estas serão detalhadas no decorrer das seções posteriores deste relatório.

5.1. Visão geral do Sistema desenvolvido

O objetivo alcançado foi o desenvolvimento de dois sistemas Web que se comunicam e compartilham dados entre si, provendo comunicação entre os dois através de *Web Services* e acesso direto ao banco de dados fazendo assim a homogeneização da interação entre os mesmos.

5.1.1. Arquitetura

Na Figura 2, é apresentada uma visão geral da arquitetura do sistema.

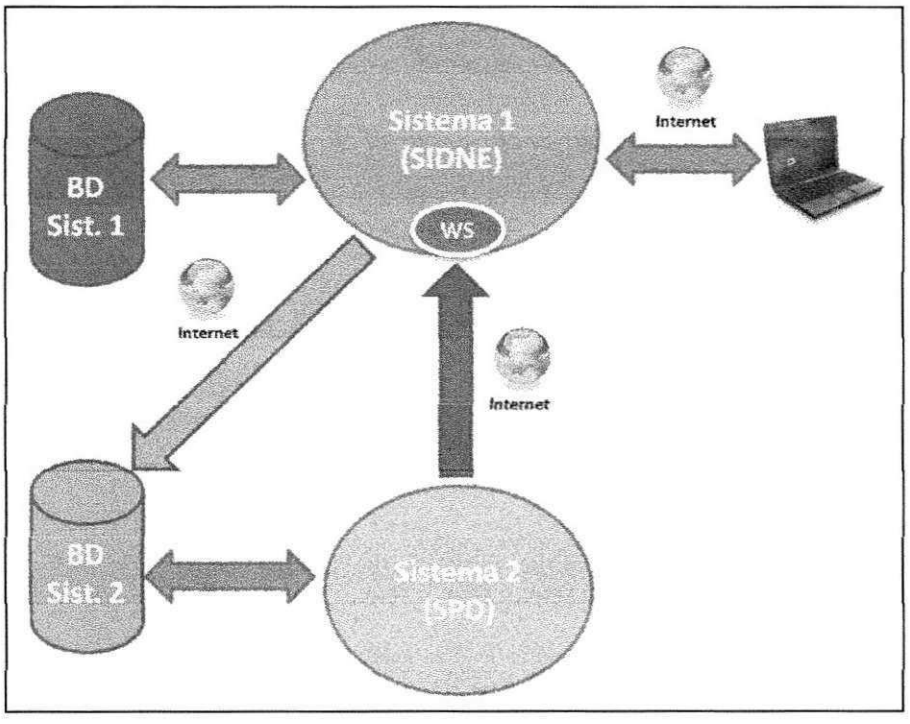


Figura 2 - Arquitetura do sistema.

Na arquitetura vemos dois sistemas web distintos.

O primeiro sistema é responsável por fazer a integração e consulta aos dados do segundo sistema (***BD Sist.2***) via conexão JDBC e consultas SQL, e desta forma usá-los na lógica de notificação de eventos. Estes métodos são disponibilizados via Web Service SOAP [SOAP].

Este sistema também possui uma base de dados própria (***BD Sist. 1***), onde são persistidos dados relativos à integração, bem como o registro e informações a respeito das chamadas ao *Web Service* provido. O primeiro sistema é chamado ***"Sistema de Integração de Dados e Notificação de Eventos"*** que será abreviado a partir deste ponto com a sigla **SIDNE**. Outra funcionalidade importante desse sistema é guardar dados relativos às chamadas ao *Web Service*, como por exemplo, quem solicitou o serviço, de que horas, qual serviço específico foi chamado e estado da execução do método do serviço.

O segundo sistema provê dados para o primeiro sistema (SIDNE) provendo assim integração e disponibilização de dados para o mesmo.

Este segundo sistema comunica-se com o primeiro através dos *Web Services* do mesmo. Através de telas, este segundo sistema aciona os *Web Services* do primeiro sistema para assim disparar as ações relativas à notificação de eventos. Este sistema chamado ***"Sistema Provedor de Dados"*** que será abreviado a partir deste ponto com a sigla **SPD**.

Os dois bancos de dados do sistema utilizam a tecnologia do MySQL como SGBD.

6. Atividades no SIDNE

Na Figura 3 é exibida a estrutura do sistema, esta reflete as atividades realizadas, detalhadas nas subseções seguintes.

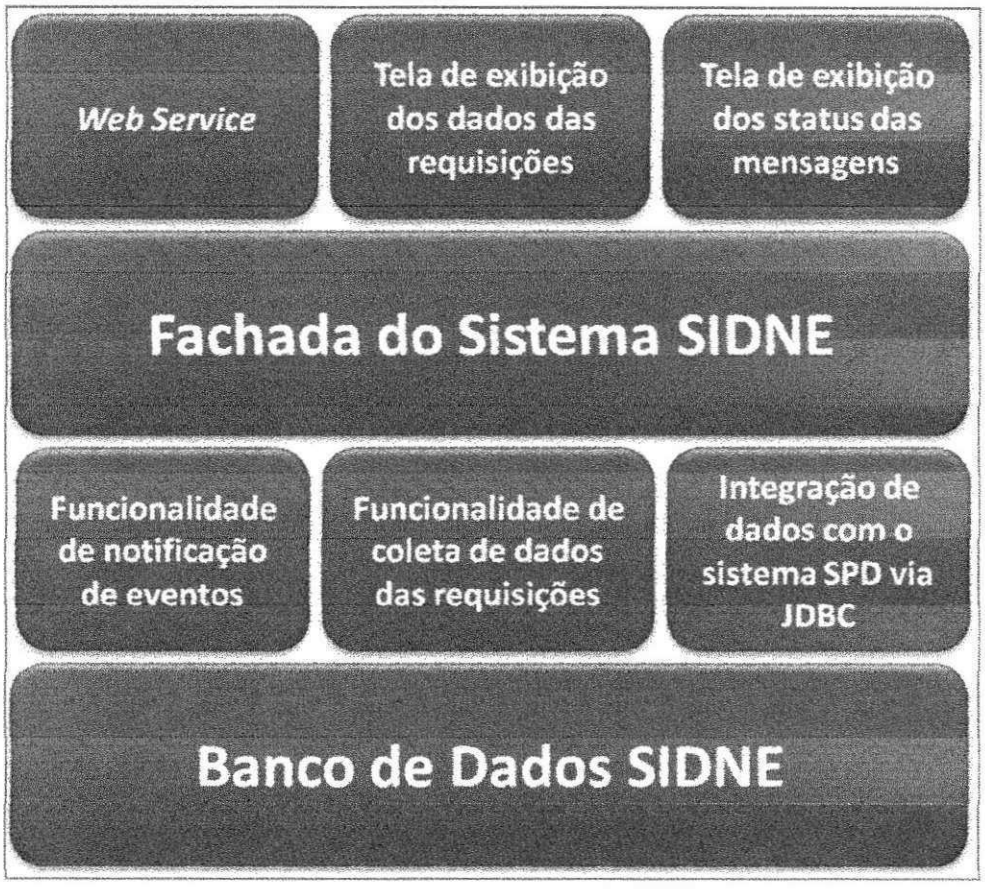


Figura 3 – Estrutura básica do SIDNE.

6.1. Avaliação e redefinição da arquitetura do sistema

Antes de iniciar a implementação das funcionalidades propriamente ditas, uma avaliação da arquitetura do sistema foi feita de forma que a mesma refletisse as necessidades do cliente e da empresa, tanto no quesito qualidade quanto no quesito financeiro associado à aplicação.

6.2. Integração entre os bancos de dados do sistema SIDNE e do sistema SPD

A primeira atividade consistiu-se em implementar a sincronização do banco de dados do sistema SIDNE com o banco de dados do sistema SPD. Para isso foi feito um método no sistema SIDNE que acessa a base de dados do sistema SPD através de consulta JDBC e verifica as mudanças ocorridas e as sincroniza através da obtenção dos novos dados.

Esta sincronização é feita sob demanda, de modo que os bancos ficam sempre sincronizados, fazendo com que o estado dos dados esteja adequado para quando precisem ser utilizados.

Na Figura 4, pode-se ver a conexão juntamente com uma consulta JDBC feita a um banco de dados MySQL na Linguagem Groovy.

```
1  import groovy.sql.Sql
2  class GroovySqlExample1{
3    static void main(String[] args) {
4      def sql =
5  Sql.newInstance("jdbc:mysql://localhost:3306/words",
6  "user1","1234", "com.mysql.jdbc.Driver")
7      sql.eachRow("select * from word"){ row ->
8         println row.word_id + " " + row.spelling + " " +
9  row.part_of_speech
10     }
11   }
12 }
```

Figura 4 - Conexão e consulta ao banco de dados MySQL via Linguagem Groovy.

6.3. Estrutura de notificação de eventos no sistema SIDNE

Para esta atividade foi desenvolvido um serviço responsável por conter as implementações dos métodos responsáveis por notificações no sistema. Além disso, foram definidas as assinaturas dos métodos juntamente com o Supervisor técnico na empresa, o qual fez sugestões e a partir de reuniões decidiu-se a estrutura final dos mesmos.

6.4. Métodos para notificação de eventos no sistema SIDNE

Nesta fase foram desenvolvidos os métodos de notificação.

A partir de determinados parâmetros passados na chamada do método do serviço, o sistema decide quais entidades do sistema serão notificadas através de uma funcionalidade de filtragem de entidades. Estas entidades podem ser objetos que representam uma entidade real, como uma pessoa, por exemplo, assim o método de notificação pode enviar uma mensagem de notificação para a pessoa, via email ou via SMS [SMS].

Cada método corresponde a um tipo de notificação diferente. Para o sistema SIDNE foi requerido que o mesmo dotasse de um módulo de notificação baseado em parâmetros passados. Com base nesses parâmetros, são filtradas entidades para as quais a notificação deve ser enviada.

Estes métodos foram implementados na camada de serviços do *Grails*. Detalhes sobre a assinatura dos mesmos serão dados quando da especificação das assinaturas dos *Web Services*, visto que os mesmo utilizam o serviço de notificação de eventos como base.

No diagrama de atividades da Figura 5 pode-se ver como é feita a notificação de eventos a partir do sistema SIDNE.

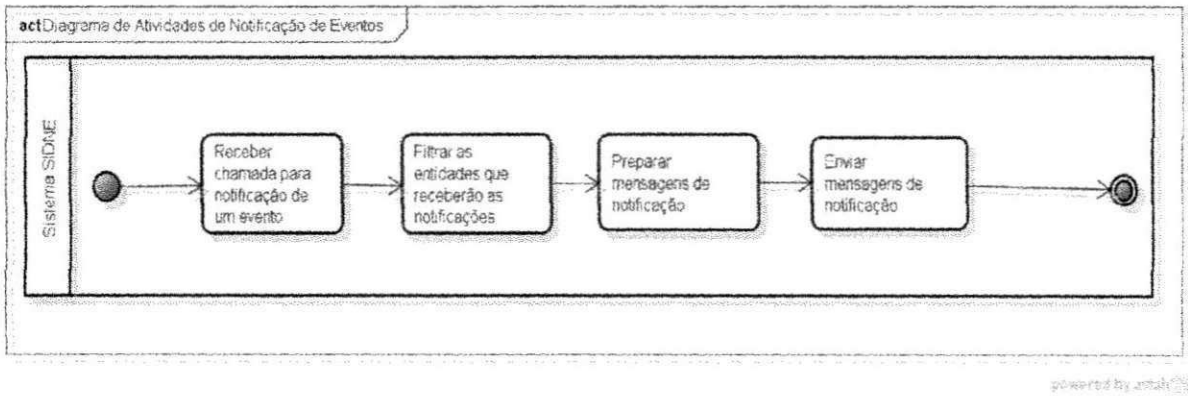


Figura 5 – Diagrama de atividades de notificação de eventos a partir do sistema SIDNE.

6.4.1. Notificação por intervalo de datas

Na notificação por intervalo da datas, a entidade é avisada através de uma mensagem a respeito de um evento que tem uma data para iniciar e uma data para finalizar. Um exemplo para ilustrar esta funcionalidade seria o caso de um Banco (como o Banco do Brasil) enviar mensagens para seus clientes informando que determinada agência do mesmo não funcionará num determinado período de tempo delimitado por um intervalo de datas.

6.4.2. Notificação em tempo determinado

Na notificação em tempo determinado, a entidade é avisada de um evento que vai ocorrer numa determinada data. Um exemplo para ilustrar esta funcionalidade seria o caso de um Banco (como o Banco do Brasil) enviar mensagens para seus clientes inadimplentes informando que a partir de uma determinada data modificar a taxa de juros no cheque especial.

6.5. Estruturação da fachada do sistema SIDNE para inserção de chamadas ao serviço de notificação de eventos

A fachada do sistema contém todas as chamadas aos serviços do sistema e neste caso houve a necessidade de adequá-la ao serviço de notificação de eventos.

Para isso foram desenvolvidos métodos nela que repassam os parâmetros de entrada para a implementação dos métodos no serviço de notificação de eventos, através de chamada aos mesmos.

6.6. Definição e Implementação do WebService para notificação de eventos

Esta atividade é das mais importantes no sistema que é a de definir as assinaturas, implementação dos métodos desses por meio de *Web Service*.

Aqui foi feita a integração do *Web Service* com a fachada para poder assim, repassar os parâmetros vindos da chamada do mesmo para os métodos de notificação de eventos.

Para cada método de notificação na fachada existe um método equivalente na chamada do *Web Service*. Além da passagem de parâmetros, é feito também tratamento e validação destes dados de entrada.

Na Figura 6, observa-se a assinaturas do método de notificação de eventos por intervalo de datas, providos pelo *Web Service*:

```
1 notificarPorIntervaloDeDatas(numeroDoProtocoloDaRequisição
2 , login, senha, listaDeParametrosParaFiltragem,
3 dataInicial, dataFinal);
```

Figura 6 - Assinatura do método de notificação de eventos por intervalo de datas.

- **Número de protocolo da requisição:** Identificador único para determinada chamada ao método;
- **Login:** Login do usuário no *Web Service*;
- **Senha:** Senha do usuário no *Web Service*;
- **Lista de parâmetros para filtragem:** É uma lista de parâmetros para filtragem das entidades que receberão as mensagens de notificação;
- **Data inicial:** Data de início do evento;
- **Data final:** Data final do evento;

Na Figura 7, observa-se a assinaturas do método de notificação de eventos em tempo determinado, providos pelo *Web Service*:

```
1 notificarEmTempoDeterminado(numeroDoProtocoloDaRequisição,
2 login, senha, listaDeParametrosParaFiltragem,
3 dataDoEvento);
```

Figura 7 - Assinatura do método de notificação de eventos em tempo determinado.

- **Número de protocolo da requisição:** Identificador único para determinada chamada ao método;
- **Login:** Login do usuário no *Web Service*;
- **Senha:** Senha do usuário no *Web Service*;
- **Lista de parâmetros para filtragem:** É uma lista de parâmetros para filtragem das entidades que receberão as mensagens de notificação;
- **Data do Evento:** Data em que o evento vai ocorrer;

O padrão utilizado para disponibilização dos métodos via WebService foi o SOAP. Para prover o *Web Service* foi utilizado o plugin XFire [XFIRE] do Grails.

Na Figura 8, tem-se um exemplo de uma pequena classe feita em *Groovy* que disponibiliza um *Web Service* via XFire no *Grails*:

```
1 class WebService {
2       static expose = ['xfire']
3       @WebMethod
4       boolean sum(number1, number2){
5             return number1 + number2;
6       }
7 }
```

Figura 8 - Classe Groovy que disponibiliza um Web Service SOAP através do plugin XFire do Grails.

6.7. Reestruturação das entidades do sistema

Durante o desenvolvimento das atividades anteriores foi necessário fazer uma reestruturação das entidades representativas do sistema. Cada entidade representa um objeto do modelo de negócio, e cada uma tem uma tabela correspondente que armazena seus dados.

Foram avaliadas as entidades juntamente com seus atributos. Desta forma, novos atributos foram adicionados, bem como novas entidades foram criadas conforme a necessidade detectada.

6.8. Comunicação com o banco de dados do sistema SPD através de conexão JDBC

Na Subseção 6.2 foi descutido o método de integração/sincronização do banco de dados do sistema SPD com o banco de dados do sistema SIDNE.

Para isto foram desenvolvidos métodos com a tecnologia JDBC que obtém dados do sistema SPD quando necessário.

6.9. Implementação da funcionalidade de coleta de dados relativos às requisições feitas ao WebService de notificação de eventos

Foi requerido que o sistema deveria gravar dados relativos às requisições feitas ao WebService de notificação de eventos. Vários dados são coletados e persistidos no Banco de Dados. Estes podem ser:

- Data da requisição;
- Hora da requisição;
- Nome da notificação;
- Número de entidades do sistema que foram notificadas;
- Login do usuário que fez a requisição;
- Número de protocolo da requisição.

Para isto foi desenvolvido um método de coleta destes dados. Este é chamado ao final de cada método do *Web Service* que faz chamada ao serviço de notificação de eventos.

Após esta fase, foram desenvolvidos métodos de acesso a estes numa controladora do *Grails* para assim posteriormente ser possível exibi-los na interface web.

6.10. Implementação da tela com dados relativos às requisições feitas ao WebService de notificação de eventos

As informações das requisições detalhadas na subseção anterior deveriam ser exibidas na interface web. Para isso foi desenvolvida uma tela em GSP para listagem destes dados, assim possibilitando a visualização dos mesmos.

6.11. Implementação da tela com dados relativos ao estado das mensagens enviadas

Nesta tela é exibido o estado da mensagem, para assim poder verificar se a mesma foi enviada corretamente ou se ouve algum erro durante o envio.

7. Atividades no Sistema SPD

Nesta seção são abordadas as atividades desempenhadas no sistema SPD. Na Figura 9, é exibida a estrutura deste sistema, que reflete as atividades realizadas detalhadas nas subseções seguintes.

Figura 9 – Estrutura básica do SPD.

7.1. Implementação de mecanismo para acesso ao WebService do sistema SIDNE via sistema SPD.

Nesta fase, a primeira atividade foi escolher a melhor forma de acessar o WebService do sistema SIDNE. Foram avaliadas alternativas de tecnologia para serem empregadas na implementação desta funcionalidade. Primeiro tentou-se o uso do GroovyWS [GROOVYWS] que é um plugin do Grails responsável por acessar o WebService SOAP disponibilizando através de métodos em Groovy as chamadas aos métodos do mesmo. No entanto, o mesmo apresentou problemas na fase de *deployment* no Tomcat, e devido a isso houve a necessidade de procurar uma tecnologia mais consistente e confiável. Depois de uma avaliação foi decidido usar o JAX-WS [JAX-WS] como tecnologia de acesso do *Web Service.*

Após esta etapa gerou-se através do JAX-WS o código Java responsável pelo acesso ao *Web Service*. Conseqüentemente foi feita a integração deste código com o código do sistema SPD, desta forma possibilitando a comunicação entre este e o sistema SIDNE.

Na Figura 10, tem-se um exemplo de código que acessa um *Web Service* SOAP via JAX-WS, serviço esse semelhante ao que foi mostrado na subseção 7.6:

```
1  public static void main(String[] args) {
2     WebService service = new WebService();
3     WebServiceDelegate delegate =
4  service.getCalculatorPort();
5     System.out.println("1. 3+7=" + delegate.sum(3, 7));
   }
```

Figura 10 - Exemplo de código em Groovy que acessa um Web Service SOAP.

7.2. Implementação de classe controladora para utilizar os métodos de consumo do WebService do sistema SIDNE via sistema SPD

Depois de integrado o código de acesso ao *Web Service* do sistema SIDNE foi desenvolvido o método de acesso a esse código para poder assim, efetivamente fazer-se chamadas aos métodos de notificação de eventos.

Este acesso foi feito a partir de uma classe controladora do Grails para posteriormente ser desenvolvida uma interface gráfica para disparar chamadas ao *Web Service*.

Também nesta classe foi feita a verificação dos parâmetros (entrada) que são enviados à chamada do *Web Service*.

7.3. Implementação da interface web para notificação de eventos

Nesta etapa foi definida em reunião como seria a interface web para notificação de eventos.

Esta interface acessa a classe controladora da subseção anterior enviando as informações dadas pelo usuário, e assim, dispara as chamadas

para os métodos de notificação de eventos do sistema SIDNE. Para cada método de notificação há uma tela diferente.

Também esta é responsável por informar ao usuário informações de ajuda caso algum dado de entrada passado pelo mesmo esteja fora dos padrões de validação.

7.4. Diagrama de atividades completo do ciclo de chamadas de notificação de eventos desde o sistema SPD até chegar ao SIDNE

Na Figura 11, pode-se ver o fluxo completo de uma notificação de eventos, desde o sistema SPD até o SIDNE.

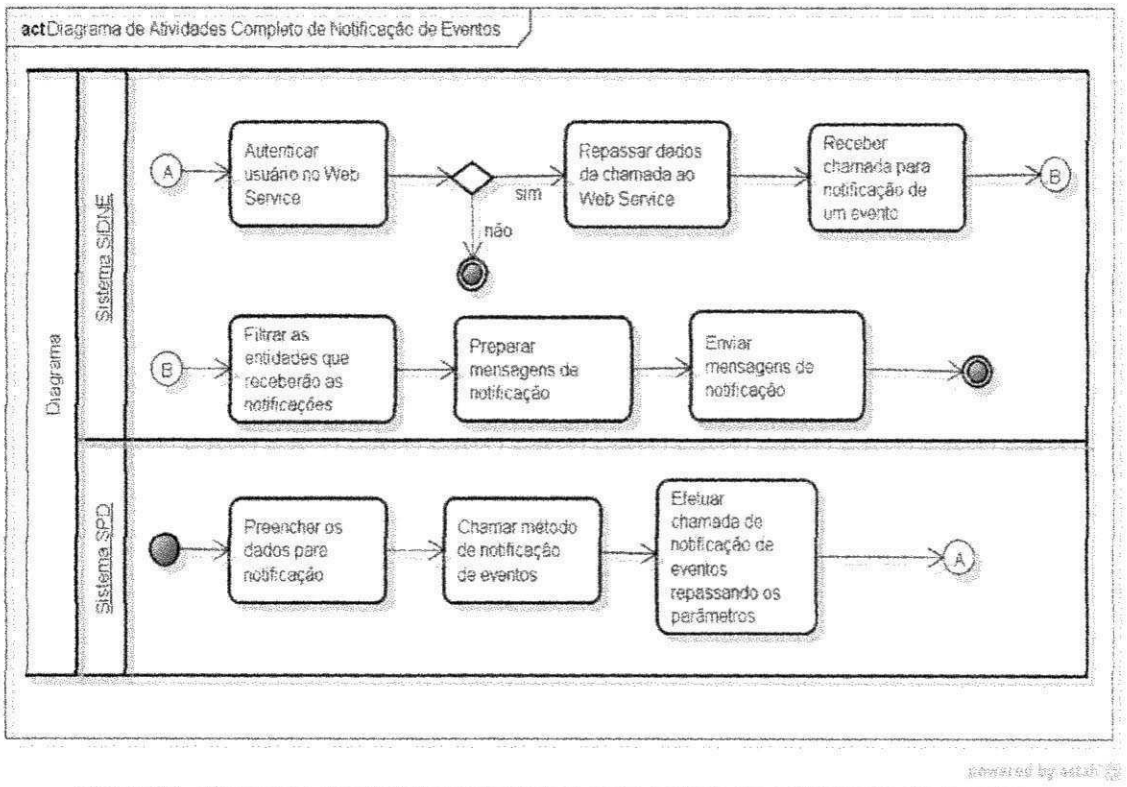


Figura 11 - Diagrama de atividades completo com o ciclo de notificação de eventos.

8. Atividades comuns aos dois sistemas

Nesta seção, são relatadas as atividades comuns aos dois sistemas.

8.1. Testes

Com o objetivo de investigar o software a fim de detectar seus principais defeitos, corrigi-los e com isso conseguir um nível de qualidade satisfatório, foram feitos vários testes nos dois sistemas, tanto no SIDNE quanto no SPD. Nas subseções seguintes detalharemos os tipos de testes que foram feitos.

8.1.1. Testes de interface gráfica

Para todas as telas (*views*) dos sistemas foram criados testes de interface gráfica. Estes visam automatizar o processo de uso do software que normalmente é feito por um usuário humano ao navegar clicando pelas diversas telas de um sistema. Para isso utilizamos o *Selenium*, *SeleniumRC Grails plugin* [SELENIUM RC] e o *Selenium IDE plugin* [SELENIUM IDE] do navegador web *Mozilla Firefox*.

8.1.2. Testes de unidade

Com o objetivo de testar os pequenas partes do sistema foram feitos testes unitários. Para isto foi utilizada a tecnologia de testes do *Grails* chamada *Grails Unit* [GRAILS UNIT TEST], que é um framework baseado em *JUnit* [JUNIT], que tem a característica principal de ser simples e ágil e aumentar a produtividade para testes unitários no *Grails*.

8.1.3. Testes de integração

Nesta fase foram desenvolvidos testes de integração nos dois sistemas. O objetivo principal foi detectar e corrigir falhas provenientes da integração dos módulos principais de cada sistema. O *Grails Unit* também fornece suporte a testes de integração, e foi a tecnologia usada para execução desta atividade.

8.1.4. Relatórios de execução dos testes

Depois de executados os testes, o *Grails Doc* [GRAILS DOC] gera um relatório com os resultados de execução dos testes. Para cada bloco de testes são apresentadas quantidades/porcentagens de testes que executaram com sucesso assim como testes com erros.

Na Figura 12, tem-se um exemplo de relatório com a saída dos testes.

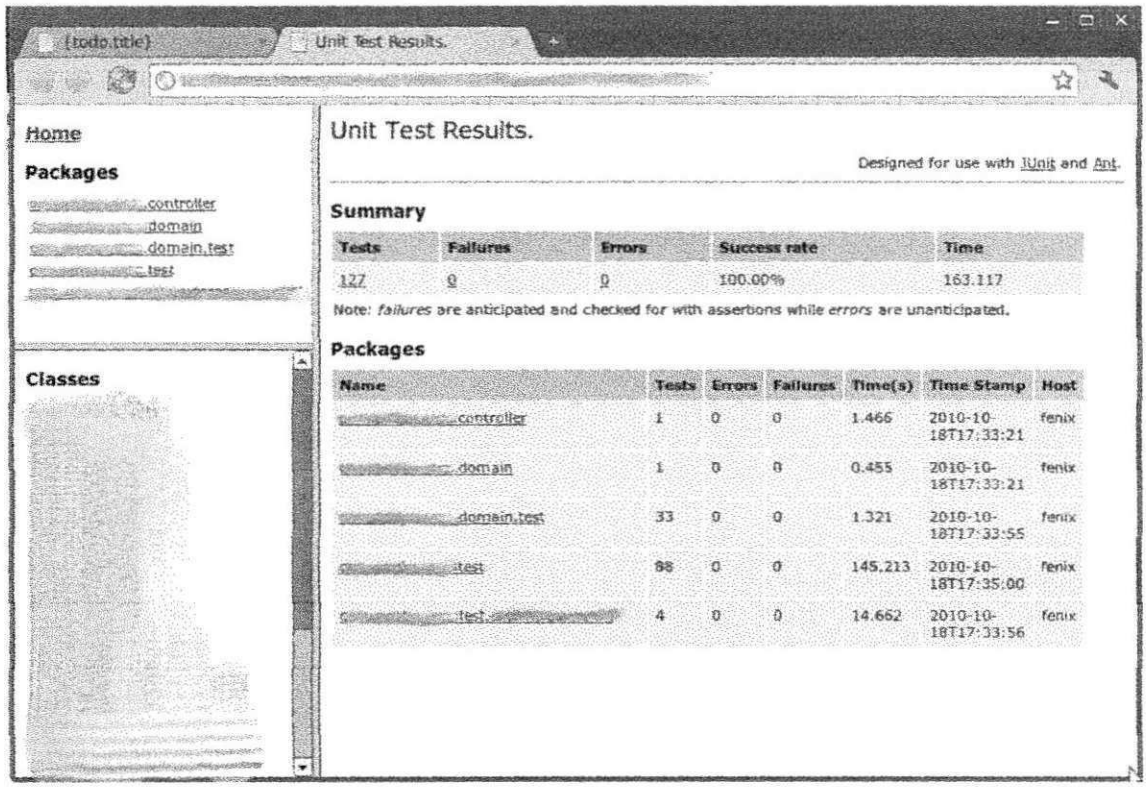


Figura 12 - Relatório com a saída dos testes.

8.2. Documentação

Para todo o código gerado durante o desenvolvimento das atividades foi feita a documentação do mesmo através da geração do *Grails Doc,* que nada mais é do que um *Java Doc* customizado para o *Grails*.

Desta forma, os desenvolvedores que tiverem que fazer a manutenção nos sistemas vão contar com mais facilidade em entender a associação entre código e funcionalidades dos sistemas.

Na Figura 13, pode-se ver um exemplo da documentação gerada.

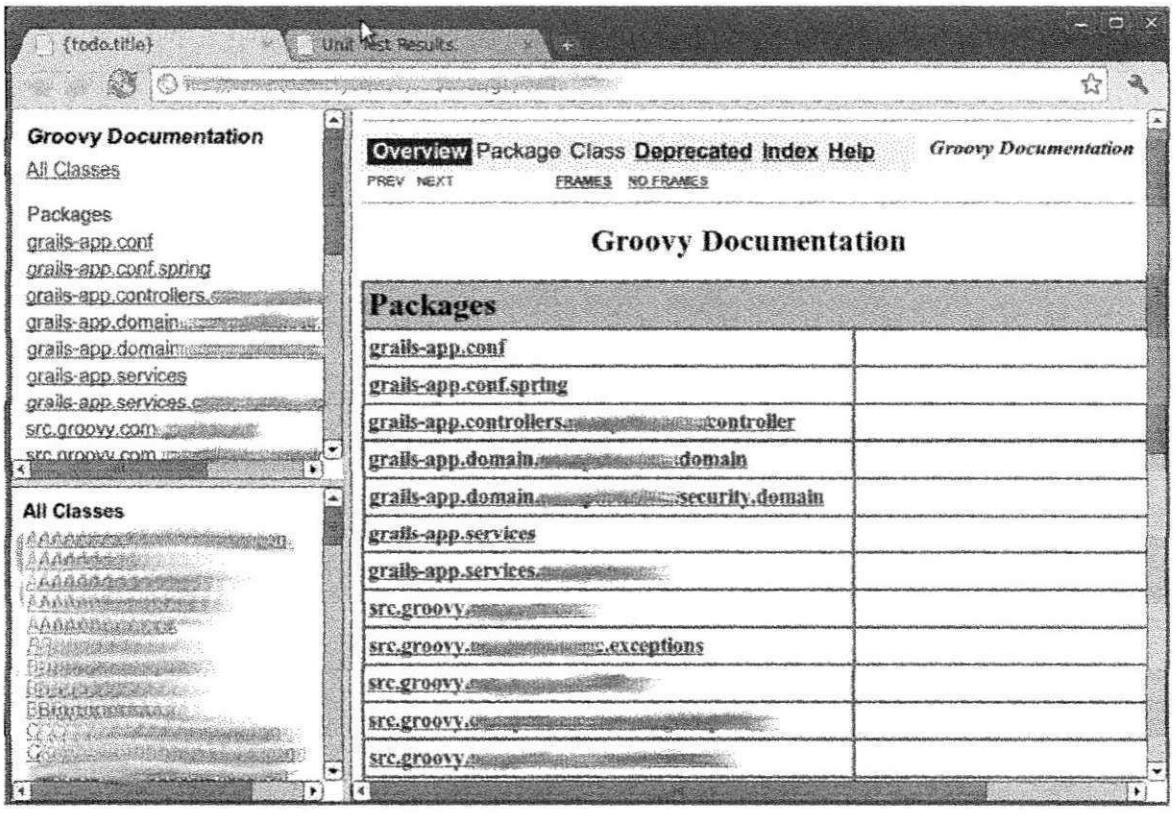


Figura 13 - Exemplo de documentação do Grails.

A partir desta documentação o desenvolvedor tem um maior detalhamento dos pacotes, classes e métodos do sistema, facilitando o entendimento das funcionalidades implementadas, bem como agilizando as atividades quando for preciso fazer alterações no código fonte.

9. Considerações finais

Neste relatório, foram descritos os resultados obtidos no estágio realizado na Empresa Signove Tecnologia S/A.

O desenvolvimento dos sistemas SIDNE e SPD proporcionou uma grande experiência e amadurecimento dos conhecimentos obtidos durante todo o curso de Ciência da Computação.

A oportunidade de poder trocar experiências com pessoas mais capacitadas no ambiente de trabalho da empresa, poder discutir idéias, resolver problemas, compartilhar conhecimentos, construir soluções de forma conjunta, fez o estágio ser dinâmico e proveitoso.

Poder desenvolver atividades seguindo uma metodologia de desenvolvimento está intrinsecamente ligado ao sucesso de qualquer projeto de *software*. Atingiu-se esse objetivo através do uso da metodologia de desenvolvimento ágil chamada SCRUM durante todo o estágio.

Os pontos chave foram poder aprender tecnologias novas, aprimorar o conhecimento obtido, ampliar a visão crítica com relação a sistemas de *software* coorporativo e principalmente ter uma visão global de todos os passos necessários para atingir o sucesso ao final de um projeto de software, tornando o mesmo um produto de qualidade.

A experiência profissional que se ganha no estágio é muito rica, pois o conhecimento obtido na universidade é colocado em prática dentro da empresa. Poder praticar o que se aprende em sala de aula é uma chance única que se recomenda para todos os alunos que buscam ser um profissional de alta qualidade.

10. Referências

[GRAILS] Grails. (s.d.). *Grails*. Disponível em: http://grails.org. Acesso em novembro de 2010.

[SQL] SQL. (s.d.). SQL. Disponível em: http://en.wikipedia.org/wiki/SQL. Acesso em novembro de 2010.

[WEB SERVICES] Web Services. (s.d.). *Web Services*. Disponível em: http://en.wikipedia.org/wiki/Web_service. Acesso em novembro de 2010.

[SCRUM] SCRUM. (s.d.). *Scrum Alliance*. Disponível em: http://www.scrumalliance.org. Acesso em novembro de 2010.

[ECLIPSE] Eclipse. (s.d.). *Eclipse IDE*. Disponível em: http://www.eclipse.org. Acesso em novembro de 2010.

[JAVA 6] Java 6. (s.d.). *Java 6*. Disponível em: http://www.oracle.com/technetwork/java/javase/overview/index-jsp-136246.html. Acesso em novembro de 2010.

[NETBEANS] NetBeans. (s.d.). *NetBeans IDE*. Disponível em: http://netbeans.org. Acesso em novembro de 2010.

[MYSQL] MySQL Database. Disponível em MySql: http://www.mysql.com. Acesso em novembro de 2010.

[SVN] SVN. Disponível em: http://en.wikipedia.org/wiki/Apache_Subversion. Acesso em novembro de 2010.

[RAILS] Rails. *Ruby on Rails*. Disponível em: http://www.rubyonrails.pro.br. Acesso em novembro de 2010.

[DJANGO] Django. *Django Project*. Disponível em: http://www.djangoproject.com. Acesso em novembro de 2010.

[HIBERNATE] Hibernate. *Hibernate*. Disponível em: http://www.hibernate.org. Acesso em novembro de 2010.

[GROOVY] Groovy. *Groovy Program Language*. Disponível em: http://groovy.codehaus.org. Acesso em novembro de 2010.

[GRAILS GSP] Grails GSP. *Grails GSP*. Disponível em: http://grails.org/doc/latest. Acesso em novembro de 2010.

[SPRING MVC] Spring MVC. *Spring MVC*. Disponível em: http://www.springsource.org. Acesso em novembro de 2010.

[TOMCAT] Tomcat. *Apache Tomcat*. Disponível em: http://tomcat.apache.org. Acesso em novembro de 2010.

[MVC] MVC. *Model View Controller*. Disponível em: http://en.wikipedia.org/wiki/Model%E2%80%93View%E2%80%93Controller. Acesso em novembro de 2010.

[PYTHON] Python. *Python Program Language*. Disponível em: http://www.python.org. Acesso em novembro de 2010.

[RUBY] Ruby. *Ruby Program Languaage*. Disponível em: http://www.rubyonrails.pro.br. Acesso em novembro de 2010.

[SMALLTALK] SmallTalk. *SmallTalk Program Language*. Disponível em: http://www.smalltalk.org. Acesso em novembro de 2010.

[W3C-XML] W3C - XML. *W3C-XML*. Disponível em: http://www.w3.org/XML. Acesso em novembro de 2010.

[ODBC] ODBC. *ODBC*. Disponível em: http://en.wikipedia.org/wiki/Open_Database_Connectivity. Acesso em novembro de 2010.

[JDBC] JDBC. *JDBC*. Disponível em: http://www.oracle.com/technetwork/java/overview-141217.html. Acesso em novembro de 2010.

[GPL] GPL. *GPL*. Disponível em: http://www.gnu.org/licenses/gpl.html. Acesso em novembro de 2010.

[SELENIUM] Selenium. *Selenium*. Disponível em: http://seleniumhq.org. Acesso em novembro de 2010.

[SOAP] SOAP. *SOAP*. Disponível em: http://www.w3.org/TR/soap. Acesso em novembro de 2010.

[SMS] SMS. *SMS*. Disponível em: http://en.wikipedia.org/wiki/SMS. Acesso em novembro de 2010.

[XFIRE] Xfire Grails Plugin. *Xfire Grails Plugin*. Disponível em: http://www.grails.org/XFire+plugin. Acesso em novembro de 2010.

[GROOVYWS] GroovyWS. *Groovy Web Service*. Disponível em: http://groovy.codehaus.org/GroovyWS. Acesso em novembro de 2010.

[JAX-WS] Jax–WS. *JAX-WS*. Disponível em: https://jax-ws.dev.java.net. Acesso em novembro de 2010.

[SELENIUM RC] Selenium RC Grails Plugin. *Selenium RC Grails Plugin*. Disponível em: http://www.grails.org/plugin/selenium-rc. Acesso em novembro de 2010.

[SELENIUM IDE] Selenium IDE. *Selenium IDE.* Disponível em: http://seleniumhq.org/projects/ide. Acesso em novembro de 2010.

[GRAILS UNIT TEST] Grails Unit Test. *Grails Unit Test.* Disponível em: http://www.grails.org/doc/1.2.2/guide/9.%20Testing.html, Acesso em novembro de 2010.

[JUNIT] JUnit. *JUnit.* Disponível em: http://www.junit.org. Acesso em novembro de 2010.

[GRAILS DOC] Grails doc. *Grails doc.* Disponível em: http://www.grails.org/doc/1.3.x/ref/Command%20Line/doc.html. Acesso em novembro de 2010.

11. Apêndices

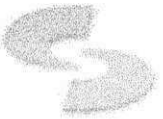
11.1. Apêndice A - Cronograma de Estágio Integrado

Meses	**Agosto**	**Setembro**	**Outubro**	**Novembro**	**Dezembro**
Atividades	**Atividades no Sistema SIDNE**				
5.1					
6.1					
6.2					
6.3					
6.4					
6.5					
6.6					
6.7					
6.8					
6.9					
6.10					
6.11					
Atividades no Sistema SPD					
7.1					
7.2					
7.3					
Atividades comuns aos dois sistemas					
8.1					
8.2					

Quadro 1 – Cronograma seguido no estágio.

11.2. Apêndice B - Plano de Estágio Integrado

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO



Plano de Estágio Integrado

Integração de sistemas de software coorporativo utilizando Web Services e SQL

Signove Tecnologia S/A

Romeryto Vieira Lira
Estagiário – UFCG

Glauber Vinícius Ventura de Melo Ferreira
Supervisor Técnico

Hyggo Oliveira de Almeida
Supervisor Acadêmico

Agosto 2010

1. Informações Pessoais

Nome: Romeryto Vieira Lira
Matrícula: 20521156
Endereço residencial: Rua João Julião Martins, Nº 413, bairro Universitário
CEP: 58429-100, Campina Grande – PB.
E-mail: romeryto.lira@signove.com
Telefones: (83) 9967-8733
(83) 8839-4118

2. Ambiente de Estágio

O estágio será realizado na empresa Signove Tecnologia S/A, a qual está localizada na rua Dom Pedro II, Nº 675, bairro da Prata, 58400-565, Campina Grande, Paraíba.

A Signove, fundada por ex-alunos de mestrado e doutorado da UFCG, possui um amplo laboratório de desenvolvimento e pesquisa, salas de reuniões, copa, recepção, sala de servidores, etc. A sala de desenvolvimento é equipada com computadores em rede de uso individual e agrupados em ilhas para facilitar a interação dos grupos.

3. Supervisão

Supervisor Acadêmico

Nome: Hyggo Oliveira de Almeida

Endereço: Laboratório de Sistemas Embarcados e Computação Pervasiva, Departamento de Engenharia Elétrica, Centro de Engenharia Elétrica e Informática, Universidade Federal de Campina Grande - Av. Aprígio Veloso, 882, Bodocongó, 58109-970, Campina Grande – PB, Brasil.

Email: hyggo@dsc.ufcg.edu.br

Supervisor Técnico

Nome: Glauber Vinicius Ventura de Melo Ferreira
Endereço: Signove Tecnologia S/A. Dom Pedro II, Nº 675, bairro da Prata, 58400-565, Campina Grande, Paraíba.
Email: glauber.ferreira@signove.com

4. Resumo

Desenvolver dois sistemas Web que se comunicam e compartilham dados entre si. Essa comunicação será efetivada através de *Web Services*[1] e acesso direto ao banco de dados, homogeneizando assim a forma através da qual os dois sistemas se iteragem.

5. Objetivos

Trabalhar no desenvolvimento de dois sistemas web com bases de dados distintas e que interagem entre si. Os dois sistemas serão desenvolvidos utilizando o framework Grails[2], os dois terão bancos de dados que serão criados com o SGBD MySQL[3]. A comunicação entre os dois sistemas web será feita através de Web Services(sistema 2 acessando os Web Services do sistema 1) e acesso direto ao banco de dados (sistema 1 acessando o banco de dados do sistema 2). Os padrões usados para os Web Services serão XML[4] e SOAP[5]. A tecnologia para acesso direto ao banco de dados é JDBC[6].

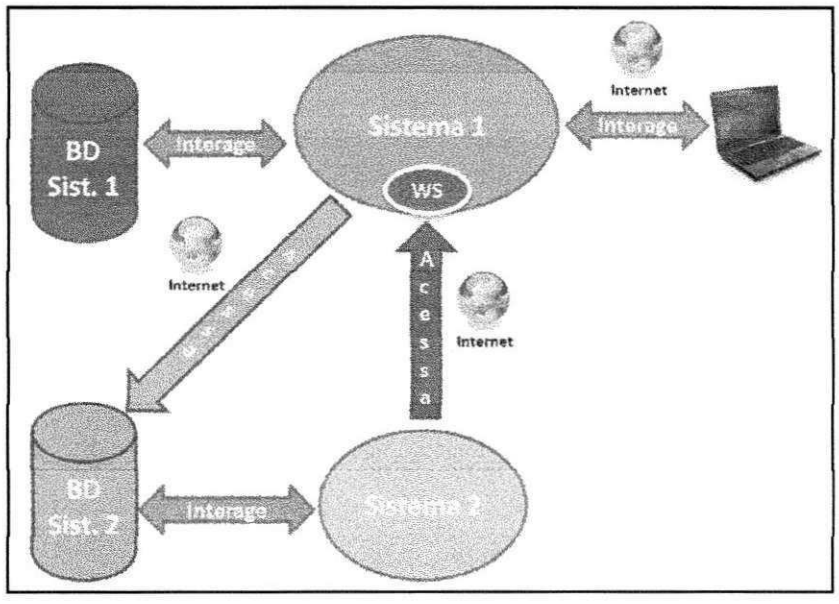


Figura 14- Arquitetura do sistema.

Entre os objetivos específicos, pode-se citar:

1. Implementar funcionalidades no sistema 1 (utilizando Grails);
2. Implementar/alterar banco de dados utilizando o SGBD MySQL para o sistema 1;
3. Implementar funcionalidades no sistema 2 (utilizando Grails);
4. Implementar/alterar banco de dados utilizando o SGBD MySQL para o sistema 2;
5. Implementar os Web Services utilizando os padrões XML e SOAP no Sistema 1 ;
6. Implementar acesso via JDBC ao Sistema 2;

6. Resultados Esperados

Implementação das novas funcionalidades bem como a comunicação completa entre dois sistemas web.

7. Metodologia

A metodologia de desenvolvimento será o SCRUM. O SCRUM apresenta-se como uma metodologia ágil e flexível, que tem por objetivo definir um processo de desenvolvimento iterativo. Esta metodologia baseia-se no desenvolvimento incremental das aplicações, centrado na equipe com ciclos de iteração curtos.

A metodologia SCRUM apenas estabelece conjuntos de regras e práticas de gestão que devem ser adotadas para garantir o sucesso de um projeto. Centra-se no trabalho em equipe, visando melhorar a comunicação e maximizar a cooperação. Englobando processos de engenharia, este método não requer nem fornece qualquer técnica ou método específico para a fase de desenvolvimento de software.

As fases de desenvolvimento SCRUM podem ser divididas basicamente em três. São elas:

- Planejamento: Definição de uma nova funcionalidade requerida pelo sistema baseado no conhecimento do sistema como um todo;
- Desenvolvimento: Desenvolvimento dessa nova funcionalidade respeitando o tempo previsto, requisitos exigidos e qualidade. Esses itens definem o fim do ciclo de desenvolvimento;
- Encerramento: Preparação para a entrega do produto persistindo as atividades de testes, documentação do usuário e treinamento, caso seja necessário.

8. Atividades Planejadas

O estágio integrado terá uma carga horária de 30 horas semanais. Seguem abaixo as atividades planejadas:

Atividades	Horas Estimadas
1. Estudo das tecnologias associadas	80
2. Desenvolvimento do sistema 1	80
3. Criação/alteração do banco de dados relacional para o sistema web I	20
4. Desenvolvimento do sistema 2	80
5. Criação/alteração do banco de dados relacional para o sistema 2	20
6. Desenvolvimento dos Web Services para sistemas 1.	40
7. Desenvolvimento do acesso via JDBC ao sistema 2.	40
8. Testes	60
9. Documentação	60
10. Elaboração do relatório final	70

9. Cronograma

Atividade	AGO	SET	OUT	NOV	DEZ
Atividade 1					
Atividade 2					
Atividade 3					
Atividade 4					
Atividade 5					
Atividade 6					
Atividade 7					
Atividade 8					
Atividade 9					
Atividade 10					

10. Bibliografia

[1] Web Services. (s.d.). *Grails*. Acesso em Agosto de 2010, disponível em: http://en.wikipedia.org/wiki/Web_service

[2] SpringSource. (s.d.). *Grails*. Acesso em Agosto de 2010, disponível em: http://grails.org/

[3] MySql Database. Acesso em Agosto de 2010, disponível em MySql: http://www.mysql.com/

[4] W3C - XML. Acesso em Agosto de 2010, disponível em: http://www.w3.org/XML/

[5] W3C - SOAP. Acesso em Agosto de 2010, disponível em: http://www.w3.org/TR/soap/

[6] JDBC. Acesso em Agosto de 2010, disponível em: http://www.oracle.com/technetwork/java/overview-141217.html