ANEXO I

Documento de Padrões Arquiteturais

Histórico de Revisão			
Data	Versão	Descrição	Autor
29/05/2013	1.0	Criação do artefato.	Amélia Pessoa
23/10/2013	1.1	Atualização do documento acrescentando informações dos frameworks usados, padrão de codificação e padrão de exceções.	Amélia Pessoa
14/11/2013	1.2	Boas práticas de codificação adotadas e observação sobre estrutura de pacotes.	Amélia Pessoa
19/02/2014	1.3	Atualização do JSF para a versão 2.2.5 e do Primefaces para a versão 4.0	Amélia Pessoa
22/04/2014	1.4	Inclusão do Framework Apache POI	Amélia Pessoa
22/04/2014	1.5	Inclusão do Framework HTML Unit	Mariana Victor
18/06/2014	1.6	Inclusão do Framework iText	Amélia Pessoa
08/07/2014	1.7	Inclusão do PostgreSQL para o módulo de Acessos à Internet	Mariana Victor
31/07/2014	1.8	Inclusão do Framework Quartz	Mariana Victor
11/12/2014	1.9	Alteração do JDK para 1.8	Tonicley Araújo
06/01/2015	1.10	Inclusão da ferramenta SQL Power Architect	Amélia Pessoa

Introdução

Este documento tem como finalidade apresentar um esboço da arquitetura dos projetos **AsaSigaLib** e **SigaNet**. Tal esboço contempla o modelo de contexto, tecnologias usadas, elementos da arquitetura e as decisões arquiteturais para os referidos projetos. Todas as especificações descritas são aplicadas aos dois projetos a não ser que se diga o contrário.

O projeto **AsaSigaLib** tem como propósito servir de framework para os projetos desenvolvidos pela ASA. Dessa forma ele proverá interfaces para todas as APIs de terceiros (como outros frameworks) fazendo com que os projetos da ASA não tenham nenhum acoplamento com estas APIs.

Ambiente/Plataforma de Desenvolvimento

Os projetos serão desenvolvidos com uso da linguagem de programação **Java**, usando a JDK (Java Development Kit) versão 1.8 (http://www.oracle.com/technetwork/pt/java/javase/downloads/index.html).

A IDE (Integrated Development Environment) de desenvolvimento utilizada será o **Asa Development Studio** descrito no documento **Padroes Arquiteturais_Asa Development Studio.**

O container web utilizado será o **Tomcat** versão 7.0 (http://tomcat.apache.org/).

Para o módulo de 'Acessos à Internet', como IDE para gerenciamento de banco de dados, utilizamos o **pgAdmin III** (http://www.pgadmin.org/) versão 1.18.1. Para os demais módulos, utilizamos o **Sql Developer** (http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html) versão 4.0.2.15.

CNPJ: 05.080.329/0001-23
E-mail: asa@asabrasil.org.br – Fone/Fax: (81) 2121.7666
Rua Nicarágua, nº 111, Espinheiro, CEP: 52.020-190 - Recife/PE

Para o design e modelagem de banco de dados, é usada a ferramenta **SQL Power Architect** (http://www.sqlpower.ca/page/architect) versão 1.0.7.

Jaspersoft studio (http://community.jaspersoft.com/project/jaspersoft-studio) software usado no design e prototipação de relatórios. Versão 6.0.0.

SonarSource (http://www.sonarsource.com/), software usado para gerenciamento e inspeção da qualidade do código.

Os passos para configurar o ambiente de desenvolvimento estão descritos no documento **Configuração do Ambiente de Desenvolvimento.**

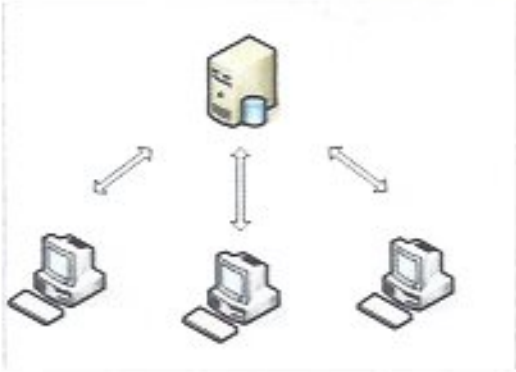
Decisões Arquiteturais

Esta seção apresenta os principais elementos relacionados às decisões arquiteturais dos projetos, os estilos arquiteturais escolhidos e os padrões de projeto que deverão guiar as atividades de projeto e implementação.

Estilos Arquiteturais

a) Arquitetura Centrada em Dados:

Os projetos irão utilizar um *repositório central de dados passivo*, isto é, o software cliente tem acesso a dados independentemente de quaisquer modificações nos dados ou nas ações dos outros softwares clientes. A arquitetura centrada em dados promove integrabilidade. Componentes existentes podem ser modificados e novos componentes clientes podem ser adicionados à arquitetura sem preocupação com outros clientes (porque os componentes clientes operam independentemente).



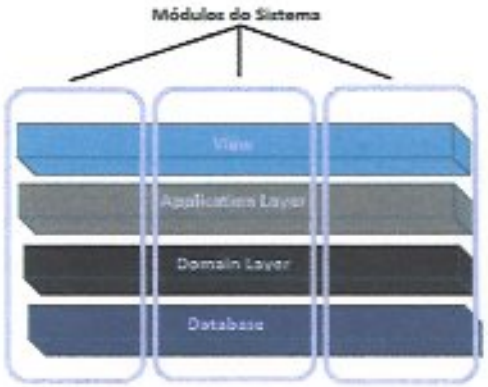
b) Arquitetura Orientada a Objetos:

Os projetos irão utilizar a arquitetura orientanda a objetos para aproximar o mundo real do mundo virtual, no qual esses objetos irão se relacionar com trocas de mensagens entre eles.

c) Arquitetura em Camadas:

CNPJ: 05.080.329/0001-23
E-mail: asa@asabrasil.org.br – Fone/Fax: (81) 2121.7666

Os projetos irão utilizar a arquitetura em camadas que visa a separar a lógica de negócio da lógica de apresentação, permitindo o desenvolvimento, teste e manutenção isolados de cada componente.

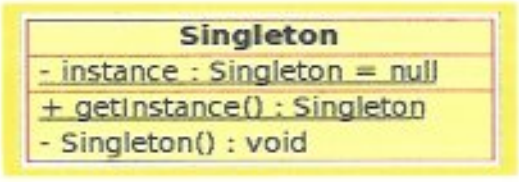


Padrões de Projeto GoF

Soluções genéricas para os problemas mais comuns encontrados em software orientado a objetos. Conjunto de 23 padrões de projetos fornecidos por Erich Gamma, John Vlissides, Ralph Jonhsone e Richard Helm no livro clássico "Design Patterns" (Gamma et al, 1994).

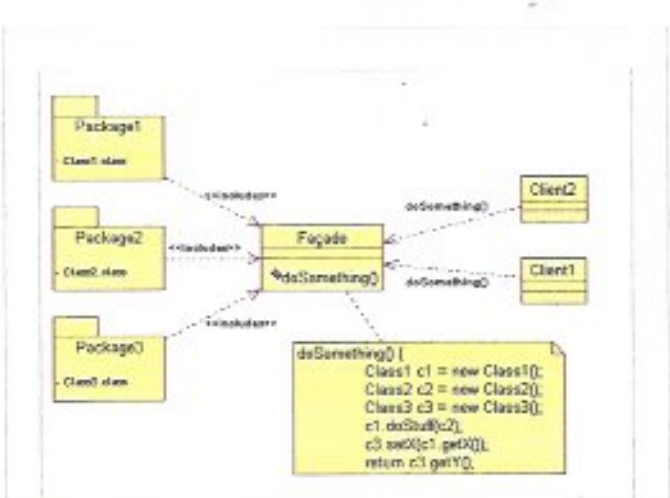
a) Singleton:

Este padrão garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto.



b) Facade:

Os projetos irão utilizar a fachada para instanciar todas as classes do pacote de negócios, aumentando a flexibilidade no desenvolvimento do projeto. A fachada implementará ainda o padrão Singleton.



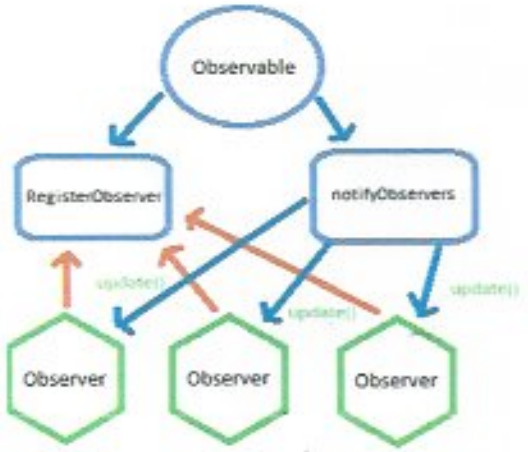
CNPJ: 05.080.329/0001-23
E-mail: asa@asabrasil.org.br – Fone/Fax: (81) 2121.7666

c) DAO:

Data Access Object é um padrão de projeto que abstrai e encapsula os mecanismos de acesso a dados escondendo os detalhes da execução da origem dos dados. Numa aplicação que utilize a arquitetura MVC, todas as funcionalidades de bancos de dados, tais como obter as conexões, mapear objetos Java para tipos de dados SQL ou executar comandos SQL, devem ser feitas por classes de DAO.

d) Observer:

O Observer é um padrão de projeto de software que define uma dependência um-para-muitos entre objetos de modo que quando um objeto muda o estado, todos seus dependentes são notificados e atualizados automaticamente. Permite que objetos interessados sejam avisados da mudança de estado ou outros eventos ocorrendo num outro objeto. O padrão Observer é também chamado de Publisher-Subscriber, Event Generator e Dependents.

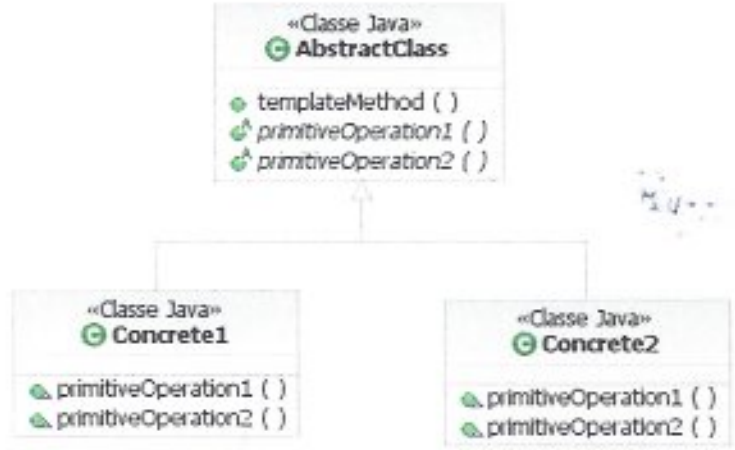


e) Template Method:

Um Template Method auxilia na definição de um algoritmo com partes do mesmo definidos por Método abstratos. As subclasses devem se responsabilizar por estas partes abstratas, deste algoritmo, que serão implementadas, possivelmente de várias formas, ou seja, cada subclasse irá implementar à sua necessidade e oferecer um comportamento concreto construindo todo o algoritmo.

O Template Method fornece uma estrutura fixa, de um algoritmo, esta parte fixa deve estar presente na superclasse, sendo obrigatório uma classeAbstrata que possa conter um método concreto, pois em uma interface só é possível conter métodos abstratos que definem um comportamento, esta é a vantagem de ser uma Classe Abstrata porque também irá fornecer métodos abstratos às suas subclasses, que por sua vez herdam este método, por Herança (programação), e devem implementar os métodos abstratos fornecendo um comportamento concreto aos métodos que foram definidos como abstratos. Com isso certas partes do algoritmo serão preenchidos por implementações que irão variar, ou seja, implementar um algoritmo em um método, postergando a definição de alguns passos do algoritmo, para que outras classes possam redefiní-los.

CNPJ: 05.080.329/0001-23
E-mail: asa@asabrasil.org.br – Fone/Fax: (81) 2121.7666
Rua Nicarágua, nº 111, Espinheiro, CEP: 52.020-190 - Recife/PE



Padrões de Projeto GRASP

GRASP: General Responsability and Assignment Software Patterns. Foca em atribuição de responsabilidades a objetos. Foram apresentados por Craig Larman no livro "Applying UML and Patterns" (Larman, 2002).

a) Alta Coesão:

Medida de quão focadas estão as responsabilidades de uma classe. As classes devem ter responsabilidades bem definidas.

b) Baixo Acoplamento:

Acoplamento ou dependência é o grau em que um módulo depende de outros módulos de programação. Isso deve ser sempre evitado no intuito de termos um acoplamento tão pequeno quanto possível.

CNPJ: 05.080.329/0001-23
E-mail: asa@asabrasil.org.br – Fone/Fax: (81) 2121.7666
Rua Nicarágua, nº 111, Espinheiro, CEP: 52.020-190 - Recife/PE

c) Polimorfismo:

Na programação orientada a objetos, o polimorfismo permite que referências de tipos de classes mais abstratas representem o comportamento das classes concretas que referenciam. Assim, é possível tratar vários tipos de maneira homogênea (através da interface do tipo mais abstrato).

d) Variações Protegidas:

Protege elementos das variações de outros elementos (objetos, sistemas, subsistemas) envolvendo o foco de instabilidade com uma interface e utilizando polimorfismo para criar várias implementações desta interface.

Métodos getters	Métodos setters

public String getNome() {
return nome;
}
 	
public double getSalario() {
return salario;
}
	public void setNome(String nome) {
	this.nome = nome;
	}
	
	public void setSalario(double salario) {
	this.salario = salario;
	}

CNPJ: 05.080.329/0001-23
E-mail: asa@asabrasil.org.br – Fone/Fax: (81) 2121.7686
Rua Nicarágua, nº 111, Espinheiro, CEP: 52.020-190 - Recife/PE

Frameworks Utilizados

Estamos fazendo uso dos frameworks descritos abaixo. Lembrando sempre que apenas o projeto **AsaSigaLib** fará referência direta aos mesmos. Sendo este responsável por fornecer ao **SigaNet** interfaces para as funções que o mesmo necessitar.

a) JSF:

JavaServer Faces (JSF) é um framework MVC de aplicações Web baseado em Java que se destina a simplificar o desenvolvimento de interfaces de usuário baseadas em web. O projeto utiliza a versão 2.2.5 do JSF e suas dependências, a saber:

JavaServer Pages 2.2, incluindo Expression Language 2.2 (JSR-245);

Expression Language 3.0 (JSR-341);

Servlet 3.0 (JSR-315);

Java Standard Edition, versão 6 (JSR-270);

Java Enterprise Edition, versão 6 (JSR-316);

Java Beans 1.0.1;

JavaServer Pages Standard Tag Library (JSTL) 1.2.

O padrão adotado para a interface gráfica do projeto é o XHTML. Este é o formato padrão adotado pelo Facelets. O Facelets é uma linguagem de descrição de páginas (PDL – Page Description Language) criada especificamente para JSF. Ele estabelece uma linguagem de templates que suporta a criação da árvore de componentes das telas JSF, o que permite o reuso de padrões de telas e a composição de componentes JSF para formar novos componentes. Ele, quando compilado no servidor e o usuário recebe a interface em puro HTML.

b) Primefaces:

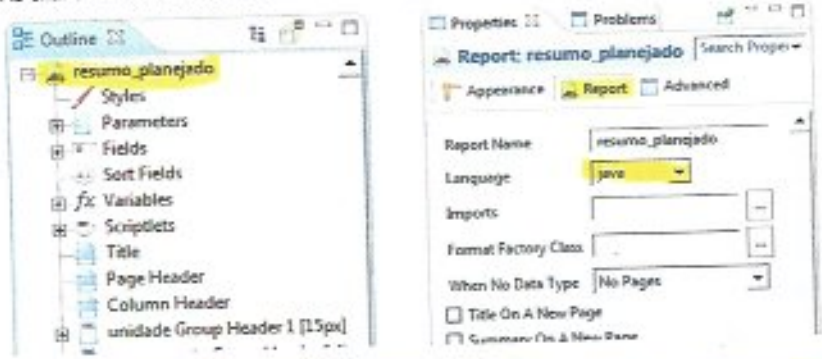
O PrimeFaces é um framework que oferece um conjunto de componentes ricos para o JavaServer Faces. Seus componentes foram construídos para trabalhar com AJAX por "default", isto é, não é necessário nenhum esforço extra por parte do desenvolvedor para realização de chamadas assíncronas ao servidor. Além disso, o PrimeFaces dá suporte à criação de funcionalidades que fazem uso de Ajax Push e permite a aplicação de temas (skins) com o objetivo de mudar a aparência dos componentes de forma simples. O projeto utiliza a versão 4.0 do Primefaces. Utiliza ainda o módulo Extensions do mesmo na versão 1.2.1.

c) Hibernate:

A persistência ao banco de dados é feito través do framework Hibernate. Ele permite a persistência dos objetos Java sem precisar (na maioria dos casos) criar os comandos SQL, bastando utilizar o objeto Session. O projeto utiliza a versão 4.2.2 do Hibernate.

d) JasperReports:

Os relatórios do projeto são feitos usando a biblioteca JasperReports. A ferramenta usada para criar o layout é o Jaspersoft Studio. Todo relatório tem uma classe que é responsável por processar os dados para a apresentação e uma classe que serve como o repositório para esses dados. O projeto utiliza a versão 6.0.0 do Jaspersoft Studio e a versão 5.0.0 do JasperReport. Ao criar um relatório é necessário alterar a linguagem de groove para java, como mostra a figura abaixo:



CNPJ: 05.080.329/0001-23
E-mail: asa@asabrasil.org.br – Fone/Fax: (81) 2121.7666

e) Log4J:

O log4j é um software livre de código aberto desenvolvido pela Apache Software Foundation. Ele fornece uma API para que o desenvolvedor de software possa fazer log de dados na sua aplicação. O projeto utiliza a versão 1.2.17.

f) Apache POI:

O Apache POI é uma biblioteca de código aberto desenvolvido pela Apache Software Foundation. Ele permite a integração Java com documentos no formato do pacote Office da Microsoft. O projeto utiliza a versão 3.10.

g) HTML Unit:

O HTML Unit é uma biblioteca de código aberto desenvolvido originalmente pela Gargoyle Software. Ele permite a manipulação de alto nível de sites no código Java, como o preenchimento e envio de formulários. O projeto utiliza a versão 2.14.

h) iText:

O iText é uma biblioteca de código aberto para gerar e manipular documentos formato PDF, usada no módulo de relatórios genéricos do sistema. O projeto utiliza a versão 2.1.7.

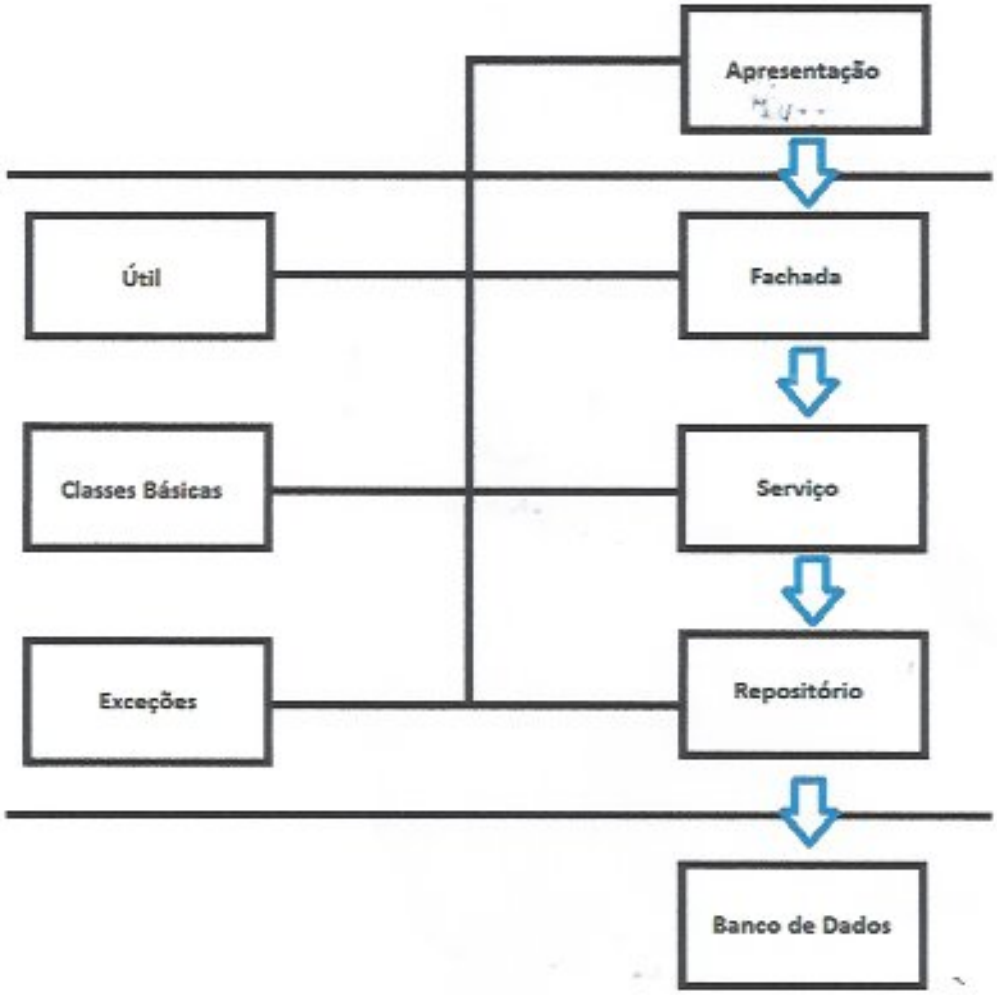
i) Quartz:

O Quartz é uma biblioteca de código aberto para o agendamento de tarefas em períodos de tempo pré-definidos, usada no módulo de Acessos à Internet. O projeto utiliza a versão 2.2.1.

CNPJ: 05.080.329/0001-23
E-mail: asa@asabrasil.org.br – Fone/Fax: (81) 2121.7666

Elementos da Arquitetura

A arquitetura dos projetos é composta pelos seguintes elementos:



Apresentação

Pacote 'visão' nos projetos. Reúne os recursos usados na camada de apresentação de dados para o usuário.

No projeto **AsaSigaLib** terá classes que encapsulam os acoplamentos com os frameworks usados na visão: JSF e Primefaces. Todo o acoplamento com estes frameworks devem se limitar as classes desse pacote. Incluem *'Listenevrs'*, *'Handlers'*, *'Converters'* etc.

Esta camada pode interagir com classes básicas, útil e exceções e pode requisitar dados da fachada.

Fachada

A fachada é um ponto de comunicação entre as camadas de apresentação e a camada de negocio e utiliza o padrão Singleton, onde se tem apenas uma instancia desse objeto nos projetos.

A principal função da fachada é centralizar todas as chamadas de métodos da camada de negócio para que outras aplicações ou outras camadas superiores possam usá-las.

CNPJ: 05.080.329/0001-23
E-mail: asa@asabrasil.org.br – Fone/Fax: (81) 2121.7666
Rua Nicarágua, nº 111, Espinheiro, CEP: 52.020-190 - Recife/PE

Esta camada pode interagir com classes básicas, útil e exceções e pode requisitar dados da camada de negócios.

Serviços

As classes de Serviço são responsáveis por fazer toda a regra de negócio do projeto. Esta camada também trata o processamento de relatórios do sistema. No projeto **AsaSigaLib** encapsula as iterações com o framework JasperReports.

Esta camada pode interagir com classes básicas, útil e exceções e pode requisitar dados da camada de repositório.

Repositórios

O Repositório é uma classe que trata todo relacionamento com o banco de dados. No projeto **AsaSigaLib** usamos o framework Hibernate, esta camada também encapsula o mesmo. Cada método do Repositório deve apenas conter comandos que criem os comandos de manipulação do objeto ao sistema. Não podendo haver validações ou qualquer tipo de regra de negócio.

Esta camada pode interagir com classes básicas, útil e exceções e pode requisitar dados direto ao banco de dados.

Banco de Dados

Bancos de dados (ou bases de dados) é um elemento na arquitetura do projeto com estrutura regular que organiza informações. Sendo essas estruturas em forma de tabela, onde cada tabela é composta por linhas, colunas e seus relacionamentos. Para o módulo de "Acessos à Internet" estamos usando o Sistema Gerenciador de Banco de Dados PostgreSQL 9.3. Para o restante do sistema estamos usando o Sistema Gerenciador de Banco de Dados Oracle versão 11g.

Útil

Pacote de classes utilitárias interagindo com os pacotes de apresentação, fachada, serviço e repositório. Contém classes para manipulação de datas, strings, números, funções de entrada e saída, criptografia, log de erros, constantes do sistema etc.

Classes Básicas

Contêm as entidades básicas do projeto, mapeadas com os campos do banco de dados. Interage com os pacotes de apresentação, fachada, serviço e repositório.

Exceções

Contém as exceções personalizadas do projeto. Interage com os pacotes de apresentação, fachada, serviço e repositório.

Padrão de Exceções

Teremos exceções personalizadas para lidar com erros específicos do sistema. Os pacotes **Repositório**, **Serviço** a **Apresentação** terão tratamento diferenciado com elas, a saber:

Repositório

Neste pacote deverá ser usada a exceção **RepositorioException**. Seu intuito é encapsular as exceções por ventura ocorrentes no banco de dados. Para tanto, todos os métodos que fazem acesso ao banco de dados deverão capturar toda e qualquer exceção ocorrida e lançar uma exceção deste tipo.

Serviço

O pacote **Serviço** usará a exceção **ServicoException**. Esta exceção será a 'mãe' da exceção **RepositorioException**. Ela será usada pelos desenvolvedores para lançar na tela exceções nas regras de negócio do sistema. Os métodos deste pacote não precisam capturar exceção alguma. Apenas as lançarão caso necessitem.
A exceção **ServiçoException** deve ser do tipo não checada (unchecked) para evitar que todos os métodos que a chamem precisem trata-la, facilitando assim o caso comum.

CNPJ: 05.080.329/0001-23
E-mail: asa@asabrasil.org.br – Fone/Fax: (81) 2121.7666

Apresentação

De forma centralizada, utilizando os recursos do framework **JSF** (Listener que centraliza requisições ao servidor), todos os métodos da visão terão suas exceções capturadas. Os métodos da visão não precisarão fazer o tratamento das exceções, já que haverá no projeto **AsaSigaLib** um tratamento centralizado para isso.

Este tratamento tomará as seguintes medidas:

a) Para exceções de Negócio:

Ao capturar uma exceção de negócio o sistema simplesmente exibirá na tela a mensagem vinculada à exceção.

b) Para as demais exceções:

Ao capturar qualquer outra exceção o sistema tomará todas as seguintes medidas:

i. Geração de log de erro:

Com a ajuda do framework **Log4J** o sistema deve gravar todos os dados possíveis da exceção capturada como mensagem e pilha de execução; print da tela no momento da exceção; usuário logado e sua unidade responsável; nome e versão do navegador utilizado. O sistema gravará tudo isso em uma pasta chamada *"erro + [data e hora da ocorrência]"* que ficará dentro de *temp/erro* localizado no diretório raiz do sistema operacional.

ii. Apresentação de mensagem de erro ao usuário:

O sistema apresentará ao usuário mensagens de erro que deem alguma informação ao usuário mas sem necessariamente expor o erro ocorrido quando não houver necessidade disso. Devemos genericamente dar uma informação sobre o erro, pedir que o usuário tente novamente e caso o erro persista, contatar o suporte do sistema.

iii. Envio de e-mail aos administradores do sistema:

O sistema considerará o atributo *javax.faces.PROJECT_STAGE* localizado no arquivo *web.xml*. Se o atributo indicar que o projeto está em produção, um e-mail será enviado ao suporte do sistema. Caso contrário será encaminhado ao usuário da máquina. O sistema enviará e-mail informando que um erro aconteceu e que todas as informações para depuração do mesmo encontram-se no servidor na pasta *temp/erro/erro + [data e hora da ocorrência]*.

Outros

Quando a exceção acontecer em outros pacotes como, por exemplo, classes que encapsulam libs de terceiros, as mesmas devem capturar a exceção e em seguida lançar uma exceção do tipo **AsaSigaLibException**, passando para esta a exceção originalmente ocorrida e uma mensagem que será futuramente exibida ao usuário.

Páginas de Erro Personalizadas

A cada comunicação realizada entre o servidor e o browser do cliente, que usa o protocolo HTTP, um código de status da operação é retornado. Estes códigos nos indicam se houve sucesso na requisição (códigos 2xx) e no caso da ocorrência de erros, nos dão informações sobre que erro ocorreu. Estes erros podem acontecer na máquina do cliente (código 4xx) ou no servidor (código 5xx).

O sistema deve apresentar página personalizada para estes erros no intuito de fornecer sempre uma interface amigável ao usuário. Além disso, os procedimentos de geração de log de erro e envio de e-mail aos administradores também devem ser acionados.

Seguem abaixo os códigos de erros mais comuns que devem ser tratados:

400 Bad Request - A solicitação não foi atendida devido a sintaxe incorreta da solicitação.

401 Unauthorized - A autenticação para realização do pedido não foi possível ou não foi fornecida.

404 Not Found - O recurso solicitado não foi encontrado, mas pode estar disponível novamente no futuro.

407 Proxy Authentication Required - Para atender a solicitação, é necessária uma autenticação no proxy.

408 Request Timeout - O servidor esperou tempo demais para processar a resposta, e finalizou o processamento para evitar que o recurso fique preso.

499 Client Closed Request - O cliente, de alguma forma, cancelou a solicitação/conexão e o servidor não consegue entregar o retorno.

CNPJ: 05.080.329/0001-23
E-mail: asa@asabrasil.org.br – Fone/Fax: (81) 2121.7666
Rua Nicarágua, nº 111, Espinheiro, CEP: 52.020-190 - Recife/PE

500 Internal Server Error - Aconteceu um erro qualquer no processamento da solicitação, porém o servidor não sabe especificar qual foi o erro.

503 Service Unavaliable - O servidor está indisponível no momento, nesse caso o servidor está ativo, mas não consegue mais atender a novas solicitações até finalizar seus processamentos.

Padrão de Codificação

Frameworks Utilizados

O projeto **AsaSigaLib** deve prover uma interface de acesso a todos os recursos dos frameworks usados de tal forma que o projeto **SigaNet** não tenha nenhum acesso direto a funcionalidades de terceiros, sempre que possível.

Classes Básicas

As classes básicas que representam entidades do sistema deverão estar localizadas no projeto **AsaSigaLib**, partindo do princípio de que as classes que são comuns a todos o projetos deverão estar localizas nesta *library*.
As classes básicas serão herdeiras diretas da classe **ObjetoBasicoSequence** e com isso herdarão os atributos que esta provêm: 'sysDate', do tipo 'Date', armazena momento da última modificação no registro; 'sysUser', do tipo 'Pessoa', armazena usuário responsável pela última modificação do registro; 'sysInactive', do tipo 'Byte', indica se o registro está ativo (Não = 0; Sim = 1); e 'id', do tipo 'Long', chave primária do registro.
Por padrão, todas as classes básicas devem ter uma sequence mapeada como sua chave primária, seguindo o seguinte padrão de nomenclatura: SEQ_NOME_TABELA. Para as classes antigas que não tinham sequences, temporariamente herdarão da classe **ObjetoBasico**.

Tratamento de Exceções

O tratamento de exceções do sistema estará centralizado no projeto **AsaSigaLib** conforme descrito na seção **Padrão de Exceções**.

Camada da Visão

O sistema é composto por diversos padrões de tela conforme descrito no documento **Padrões de Interface**. Cada padrão de tela tem suas classes básicas que devem ser herdadas. Tudo isso é discriminado no documento.

Este camada deve ter apenas as iterações feitas com as telas, deixando todas as regras de negócio no pacote **Serviço**. Quando a regra de negócio envolver algum tipo de interação com o usuário, como por exemplo, uma mensagem de confirmação ou exibição de uma mensagem específica, deve ser usada a classe **RetornoView**. O método da classe **Serviço** retorna esse objeto e os métodos da visão se encarregam da exibição.

Exemplo:
Método de negócio

```
public RetornoView salvarConfiguracao(ConfiguracaoPlanejamento configPlan) {

        RetornoView retorno = null;

        //Regras de negócio, validações, salvando registros

        retorno = RetornoView.SALVO_SUCESSO;

        return retorno;
}
```

Método da visão

```
public void salvar(){
        //Chama salvar sem exibir a mensagem
        RetornoView retorno = Fachada.getInstancia().salvarConfiguracao();

        //Manda exibir retorno
        exibirDialogo(retorno);
}
```


CNPJ: 05.080.329/0001-23
E-mail: asa@asabrasil.org.br – Fone/Fax: (81) 2121.7666
Rua Nicarágua, nº 111, Espinheiro, CEP: 52.020-190 - Recife/PE

Comentários de Código

Uma boa prática na codificação que vem a facilitar a manutenção do software, principalmente quando temos vários desenvolvedores trabalhando em conjunto é manter o código sempre comentado de forma a facilitar o entendimento com relação ao comportamento daquele código.
Neste projeto padronizamos os comentários de classe e os de método. Não excluindo comentários interiores, mas sim certificando que haja ao menos estes.

Os comentários devem seguir os padrões abaixo:

Comentários de classe

```
/**
 * Breve definição das tarefas da classe
 * @author Criador da mesma
 *
 */
```

Comentários de método

```
/**
 * Descrição do comportamento do método
 *
 * @param Descrição dos parâmetros
 * @return Descrição do retorno do método
 */
```

Sessões Web

As sessões web gerenciadas pelo **Tomcat** iniciam-se automaticamente quando a página inicial do sistema é aberta. A partir do momento que o usuário faz login, um usuário é vinculado àquela sessão.

Enquanto houver um usuário vinculado àquela sessão, o encerramento desta será transparente ao usuário, pois o **AsaSigaLib** trata isso para que a sessão seja recuperada sem prejuízo dos dados digitados pelo usuário. Para ele o sistema nunca expirará a sessão sem que ele saia do sistema ou feche o navegador.
Este tratamento encontra-se na classe **HttpSessionListener**.

Encode Padrão

A codificação padronizada no projeto é UTF-8 (8-bit Unicode Transformation Format). Esta é um tipo de codificação Unicode de comprimento variável. Pode representar qualquer carácter universal padrão do Unicode, sendo também compatível com o ASCII. Por esta razão, está lentamente a ser adotado como tipo de codificação padrão para email, páginas web, e outros locais onde os caracteres são armazenados.

Constantes do Sistema

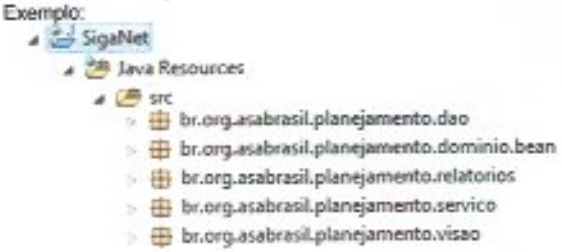
Os valores constantes no sistema serão listados na classe **ConstantesSistema**. Nesta classe deverão estar apenas valores que não correm o risco de serem mudados, como por exemplo, o nome de páginas .xhtml.
Para os valores parametrizáveis usamos o armazenamento em banco de dados através da tabela **PARAMETRO_SISTEMA**. A tabela segue modelo conforme figura abaixo.

Exemplos de uso:
TEMPO_EXPIRAR_SESSAO_USUARIO
TEMPO_EXPIRAR_SENHA_USUARIO
SERVER_SMTP_EMAIL

CNPJ: 05.080.329\0001-23
E-mail: asa@asabrasil.org.br – Fone/Fax: (81) 2121.7666

Estrutura de Pacotes

Teremos no sistema o conceito de *módulos*. Os módulos serão conjuntos de funcionalidades interligadas. Como exemplo disso podemos citar o módulo de planejamento, módulo de segurança etc.
Daí, além da estrutura listada no tópico **Elementos da Arquitetura**, teremos pacotes personalizados para cada módulo do sistema, assim o módulo planejamento terá um pacote com este mesmo nome, também subdividido pelos pacotes citados nos **Elementos da Arquitetura**.
Exemplo:



Boas Práticas Adotadas

a) Funções e Procedimentos:

A principal diferença entre uma função e um procedimento está no fato de que uma função obrigatoriamente retorna um valor, enquanto que um procedimento não retorna valor algum, ou seja, o procedimento apenas executa uma ação, podendo em algumas situações alterar os parâmetros recebidos. Por isso, no desenvolvimento de procedimentos devemos detalhar no comentário do método como e qual alteração será feita no parâmetro recebido.

b) Duplicação de código:

A duplicação de código do sistema será controlada pelo Sonar. Logo, o índice de duplicação apontado pelo mesmo deve estar sempre zerado com exceção das classes do pacote **domínio**. Serão desconsideradas ainda as duplicações apontadas no pacote **dao** quando estas ocorrerem por conta da utilização das mesmas tabelas.

c) Complexidade ciclomática:

Este item também será controlado pelo Sonar. Devendo seu índice estar sempre em zero.

d) Controle de qualidade do código fonte:

Para maior controle na qualidade do código escrito, a cada nova funcionalidade/melhoria adicionada, deve fazer uma análise do código com o software sonar de forma que o mesmo não acuse **nenhuma** violação. Ao fim do desenvolvimento deve-se realizar o checklist de codificação disponível no link https://docs.google.com/forms/d/1KodprWziLiul_PsIwyuDKlwsgae6E9lRu5bL9T9JKJg/viewform e encaminhar a RM para auditoria de outro desenvolvedor.
As respostas enviadas no checklist estão disponíveis no link https://docs.google.com/spreadsheet/ccc?key=0AtlZHMKCFZIHdDhDUlNFLTVVbWI90Ny11N2JUOVdCa3c#gid=0.

CNPJ: 05.080.329/0001-23
E-mail: asa@asabrasil.org.br – Fone/Fax: (81) 2121.7666
Rua Nicarágua, nº 111, Espinheiro, CEP: 52.020-190 - Recife/PE

Mensagens do Sistema

O projeto SigaNet, fazendo uso dos recursos do JSF, centralizará todas as mensagens trocadas com o usuário no arquivo "messages_pt_BR.properties" permitindo assim uma futura internacionalização do sistema.

Padrões de Nomenclatura

A escrita de código passa inevitavelmente por criar nomes. Nomes cuidadosamente escolhidos tornam o código mais legível.
O objetivo de uma política de nomenclatura é escolher certas regras que se seguidas, não importa por quem, produzem um nome legível, com significado e com a mesma estrutura.

Estilos de Escrita

Todos os nomes devem seguir o padrão **camelCase** em que a primeira letra é escrita minúscula e as palavras são aglutinadas sem espaços ou sinais não alfabéticos. As palavras são diferenciadas colocando a primeira letra em caixa alta. No caso de constantes as mesmas devem ser nomeadas com caixa alta e com suas palavras separadas por *underline*, exemplo: MINHA_CONSTANTE.
Siglas como HTTP, URL e AJAX (entre outros) devem seguir o estilo de caixa normal como se fossem palavras, por exemplo um método que recupere um URL seria getUrl() e não getURL.

Nomeando Classes

Os nomes das classes devem ser substantivos que designam o conceito ou a responsabilidade que a classe ocupa no sistema. Os nomes devem ser específicos o suficiente para estarem relacionados ao problema ou à solução. Nomes de classes devem sempre começar com letra maiúscula e podem ter prefixos ou sufixos que tentam orientar o programador sobre o nível que a classe ocupa em uma hierarquia ou no design do sistema, por exemplo: **GraficoBean**, onde o sufixo *Bean* denota a função da classe.

Nomeando Interfaces

As regras de nomenclatura de interfaces devem seguir os mesmos princípios das regras para classes. A única diferença é que podemos escolher não apenas substantivos, mas também adjetivos e advérbios.

Nomeando Métodos

Nomes de métodos devem ser verbos no infinitivo com a possibilidade de serem compostos com advérbios ou substantivos relativos ao papel dos argumentos (enviarEmail, gerarRelatorio etc). Nomes de métodos devem expressar a ação que será realizada ou a mudança de estado que se espera que aconteça.
Não deve ser incluído no nome do método nomes de tipos de variáveis que ele receba ou retorne, pois uma pequena mudança pode fazer necessária a mudança também do nome do método.
Para os métodos de CRUD de dados, deve ser usado o padrão *selecionar*, *atualizar* (servindo para inserção e atualizações) e *deletar*.

Modelos De Nomenclatura

Algumas classes com utilidade específica têm formas específicas de serem nomeadas. A seguir são listados alguns modelos de nomenclatura para estes conjuntos de classes, baseados nas regras anteriores.

a) **Repositório:**

A nomenclatura do padrão Repositório segue o modelo: [Entidade]Repositorio.
Esta nomenclatura segue o princípio de Preferência de Domínio Técnico deixando o nome do padrão no fim do nome. O nome é precedido pelo nome da entidade a que este repositório se refere.

b) **Serviço:**

O padrão Service segue o modelo: [Entidade]Servico ou para classes de regra de negócio sem entidade específica vinculada: [Propósito]Servico.

xxxxxxxxxxxxxxxxxxxx

CNPJ: 05.080.329/0001-23
E-mail: asa@asabrasil.org.br – Fone/Fax: (81) 2121.7666