Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Programa de Pós-Graduação em Ciência da Computação

Dissertação de Mestrado

Documentação e Validação de Arquiteturas de Software: Uma Proposta Concreta

Leandro José Ventura Silva

Campina Grande – Paraíba – Brasil
Junho de 2013

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Programa de Pós-Graduação em Ciência da Computação

Documentação e Validação de Arquiteturas de Software: Uma Proposta Concreta

Leandro José Ventura Silva

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande – Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de Software

Jacques Philippe Sauvé
(Orientador)

Campina Grande – Paraíba – Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

S586d Silva, Leandro José Ventura.

Documentação e validação de arquiteturas de software : uma proposta concreta / Leandro José Ventura Silva. – Campina Grande, 2013.

239 f. : il.

Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2013.

"Orientação: Prof. Dr. Jacques Philippe Sauvé".

Referências.

1. Arquitetura de Software. 2. Documentação. 3. Validação. I. Sauvé, Jacques Philippe. II. Título.

CDU 004.273(043)

"DOCUMENTAÇÃO E VALIDAÇÃO DE ARQUITETURAS DE SOFTWARE: UMA PROPOSTA CONCRETA"

LEANDRO JOSÉ VENTURA SILVA

DISSERTAÇÃO APROVADA EM 26/06/2013

JACQUES PHILIPPE SAUVÉ, Ph.D, UFCG
Orientador(a)

DALTON DARIO SEREY GUERRERO, D.Sc, UFCG
Examinador(a)

CHRISTINA VON FLACH GARCIA CHAVEZ, Drª, UFBA
Examinador(a)

CAMPINA GRANDE - PB

Resumo

Arquiteturas de software fornecem o contexto do sistema, guiam o processo de desenvolvimento, auxiliam na comunicação entre as partes interessadas, aumentam a integração com sistemas legados e softwares de terceiros e reduzem custos de manutenção e evolução; todavia, criar, documentar e validar arquiteturas são atividades complexas, ainda mais para aqueles com pouca experiência. A escassez de exemplos e de métodos diretos para auxiliar na documentação e validação também dificulta a realização dessas atividades. Neste trabalho, apresentamos um modelo de documento e um método para validação de arquiteturas de software. Nele, foram desenvolvidas arquiteturas de software para dois sistemas reais e um deles foi validado com o uso do método em questão. Os resultados obtidos indicam que o modelo de documento agiliza o processo de documentação e guia arquitetos iniciantes por conter as seções necessárias a serem preenchidas; evita o esquecimento de tópicos importantes e leva o arquiteto a seguir uma metodologia para documentar a arquitetura. Já o uso do método de validação aumentou a confiança na arquitetura desenvolvida. Seguir o processo foi relativamente simples devido à existência de um fluxo bem definido; os artefatos produzidos garantiram que a validação fosse bem documentada, mantendo uma memória do que foi feito.

Abstract

Software architectures provide the system context, guide the development process, assist the communication among stakeholders, increase reuse and help integrate with legacy and third party software and reduce maintenance and evolution costs; however, creating, documenting and validating architectures are not trivial tasks. The lack of examples and straightforward methods also makes accomplishing these activities more difficult. We present a document template and a method for validating software architectures. Software architectures for two different real systems were designed and one of them was validated using the proposed method. The results indicate that the document template accelerates the documentation process and guides junior architects by containing the necessary sections to be filled out; the template also ensures that important topics are mentioned and forces the architect to follow a methodology to document the architecture. Results show that the validation method increased the confidence in the designed architecture. The process was relatively simple due to the existence of a well-defined flow, and the resulting artifacts ensured that the validation was well documented by keeping a memory of what was done.

Agradecimentos

Agradeço primeiramente a Deus, por me dar força de vontade, garra, inteligência e sabedoria para trilhar os meus caminhos.

Agradeço também a meus pais e meu irmão, que sempre me apoiaram e me deram forças para não desistir dessa jornada.

Também sou muito grato a minha amada esposa e ao meu gordinho do coração, Abraão, que vieram para iluminar minha vida; esses dois também sempre me fizeram continuar firme nessa conquista.

A Jacques, um professor excelente que soube como me suportar e também quando puxar minha orelha.

A Eloi, por sempre me ajudar na *Smartiks* e me aliviar um pouco no trabalho para que eu pudesse dar continuidade às atividades.

Aos companheiros da *Smartiks* e do STI, por acreditarem no meu trabalho e sempre quebrar galhos para mim quando necessário.

A meus amigos André, Antônio Júnior, Artur, Camilla, Danilo, Diego, Gilliard, Isaac, Lorena, Marcondes, Mikaela, Renato, Rodrigo, Romeryto, Thiago e Zé Gildo; obrigado a todos pela torcida. Eu provavelmente esqueci de alguém, mas não fique com raiva, alguém, não foi por mal!

Por fim, ao pessoal da COPIN, Aninha, Rebeka e Vera, que sempre me ajudaram quando necessário.

Conteúdo

Lista de Acrônimos

Acrônimo	Descrição
AJAX	*Asynchronous Javascript and XML*
ALSPM	*Architecture Level Prediction of Software Maintenance*
ATAM	*Architecture Trade-Off Analysis Method*
Chesf	Companhia Hidro Elétrica do São Francisco
Equest	Questionário do Plano diretor de TI
ESAAMI	*Extending SAAM by Integration in the Domain*
GUI	*Graphical User Interface*
GWT	*Google Web Toolkit*
HTTP	*Hypertext Transfer Protocol*
IP	*Internet Protocol*
JAR	*Java Archive*
JVM	*Java Virtual Machine*
JOSSO	*Java Open Single Sign-On*
LDAP	*Lightweight Directory Access Protocol*
MVAS	Modelo de Validação de Arquiteturas de Software
PSI	Portal de Sistemas Integrados
REST	*Representational State Transfer*
RF	Requisito Funcional
RMI	*Remote Method Invocation*
RNF	Requisito Não-funcional
SAAM	*Scenario-Based Architecture Analysis Method*
SAAMER	*Software Architecture Analysis Method for Evolution and Reusability*
SAEM	*Software Architecture Evaluation Model*

SAGE	Sistema Aberto de Gerenciamento de Energia
SBAR	*Scenario-Based Architecture Reengineering*
SFTP	*Secure File Transfer Protocol*
SGBD	Sistema de Gerenciamento de Banco de Dados
SICOB	Sistema de Controle de Bolsas
SIRPE	Sistema de Relatórios para Pontos Eletrônicos
SSH	*Secure Shell*
SSL	*Secure Sockets Layer*
SSO	*Single Sign-On*
STI	Serviço de Tecnologia da Informação
SVG	*Scalable Vector Graphics*
UFCG	Universidade Federal de Campina Grande
URL	*Uniform Resource Locator*
WAR	*Web Application Archive*
WYSIWIG	*What You See Is What You Get*
XML	*Extensible Markup Language*

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O termo "arquitetura de software"foi utilizado pela primeira vez no final da década de 1970 em uma conferência sobre técnicas de Engenharia de Software organizada pelo *NATO Science Commitee* [1, 2]; o seu conceito como disciplina, porém, foi enfatizado somente nos anos 90 com a publicação dos primeiros livros [3–8], artigos [9–11] e métodos para criação e validação de arquiteturas [12–14].

Embora não haja um consenso para a sua definição [2], uma arquitetura de software é, de forma simples, o conjunto de informações e decisões cruciais para auxiliar no desenvolvimento, implantação, manutenção e evolução de um sistema de software.

A documentação das arquiteturas é uma parte muito importante no processo de desenvolvimento de softwares, pois esse documento permite o registro dos porquês de cada decisão importante tomada. Além disso, o entendimento do sistema é facilitado para novos envolvidos no processo, como, por exemplo, desenvolvedores e arquitetos. Ainda, a documentação acelera a fase de implementação, pois, assumindo que decisões complexas já foram tomadas e devam estar na arquitetura, os desenvolvedores podem continuar seu trabalho sem muitas consultas ao arquiteto. Por fim, a partir das restrições impostas pela arquitetura, os desenvolvedores podem tomar decisões para problemas mais simples, que não necessariamente estarão na arquitetura.

Assim como documentar as arquiteturas, validá-las é essencial para garantir que as decisões tomadas satisfazem todos os requisitos impostos pelo cliente. E, para sistemas que executam tarefas de segurança crítica, a validação deve garantir que nada além do que foi especificado venha a acontecer [15].

1.1 Dificuldades na Produção de Arquiteturas de Software

Não é rara a existência de softwares desenvolvidos com problemas de desempenho, escalabilidade, segurança, portabilidade, disponibilidade e outros comportamentos inesperados. Tais problemas podem ter sido causados pela falta de uma arquitetura ou falha na produção da mesma, pois é exatamente nela onde essas preocupações devem ser abordadas [15].

A criação de arquiteturas de software não é um trabalho simples, e se torna ainda mais difícil porque o ensino para os não-praticantes é defasado [16]. Ademais, é preciso saber como documentá-las corretamente, porém existem poucos modelos e exemplos bem escritos de documentos arquiteturais [17–23], dificultando o entendimento do que e como deve ser ou não descrito nos mesmos.

Como mencionado, validar é essencial no processo de produção de arquiteturas, porque é custoso alterá-las quando o sistema já está em desenvolvimento. Infelizmente, essa etapa não é trivial porque nem sempre é factível esperar que o sistema seja implementado para validá-lo. Complementando, essa etapa também partilha do mesmo problema da documentação, ou seja, há poucos exemplos na literatura [13, 24].

1.2 Objetivos

O objetivo principal deste trabalho é amenizar os problemas citados anteriormente com a contribuição de exemplos reais de documentos de arquiteturas de software. Para isso, foi criado um modelo de documento, que, por sua vez, foi utilizado para documentar as arquiteturas de dois sistemas. Por último, um método para validação foi desenvolvido e usado para validar uma das arquiteturas produzidas.

O modelo de documento está especificado no capítulo 3 e o método de validação, no capítulo 4. Já os exemplos produzidos podem ser lidos nos apêndices B, C, D.

1.3 Relevância

Como citado, o ensino de arquiteturas de software para não-praticantes é defasado. Visto isso, a existência de novos exemplos completos de documentos para especificação e validação de arquiteturas para sistemas reais pode reduzir essa defasagem. Além do mais, esses

exemplos também podem ser utilizados como referência em ambientes profissionais.

O diferencial do que foi produzido é que os documentos foram preparados para serem bem didáticos, pois houve a preocupação de definir até os termos mais comuns na área de Engenharia de Software a fim de garantir que o entendimento seja satisfatório para diferentes tipos de leitores.

Outro ponto que torna o trabalho relevante é que o modelo de documento e o método de validação foram utilizados e aperfeiçoados a partir da utilização dos mesmos em ambientes reais.

1.4 Estudo Realizado

No estudo em questão foi produzido um modelo de documento para arquiteturas de software seguindo recomendações do padrão ISO/IEC/IEEE 42010 [25] e de livros na área [15, 26, 27]. Esse modelo é composto por uma breve descrição do sistema; definições de termos utilizados; partes interessadas; requisitos; decisões-chave; um mapeamento de requisitos para decisões-chave; pontos de vista e visões.

Em seguida, uma arquitetura de software foi projetada e documentada para o sistema SmartSwitch [28] a partir de reuniões com as partes interessadas. Durante o desenvolvimento, essa arquitetura sofreu ajustes porque houve certas mudanças nos requisitos do sistema.

Após a criação da primeira arquitetura, o Método para Validação de Arquiteturas de Software (MVAS) foi desenvolvido com base em métodos já existentes [13, 24, 29, 30]. Esse método exige o uso de diferentes técnicas de medição dependendo do tipo do requisito arquitetural; seu resultado final é um relatório de validação arquitetural, resumindo o que ocorreu durante todo o processo.

Depois do desenvolvimento do MVAS, uma nova arquitetura de software foi produzida para o Portal de Sistemas Integrados (PSI), que será utilizado na Universidade Federal de Campina Grande (UFCG). Assim como no caso do SmartSwitch, foram realizadas reuniões com as partes interessadas, que ocorreram no Serviço de Tecnologia da Informação (STI) da UFCG. Por fim, essa arquitetura foi validada utilizando o método proposto nesse trabalho.

Com esse estudo, verificou-se que o tempo gasto e a dificuldade no processo de docu-

mentação e validação foram menores. Fora isso, os artefatos do MVAS ajudaram a manter registro do que foi feito. Por outro lado, existem partes do modelo que estão mais detalhadas por terem sido mais utilizadas na documentação dos sistemas já expostos. Outro ponto negativo é que foi difícil manter a documentação da arquitetura atualizada com relação ao código. Em suma, o modelo e o método produzidos são promissores porém não são maduros o suficiente porque foram utilizados em somente dois sistemas.

Capítulo 2

Fundamentação Teórica

2.1 Arquiteturas de Software

Não existe uma definição *de facto* para arquiteturas de software [16]; na verdade, existem dezenas delas [15]. Seguem abaixo algumas bastante utilizadas na literatura:

> *System fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution.* [25]
>
> *The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time.* [31]
>
> *The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relations among them.* [26]

Embora existam várias definições diferentes, elas são compatíveis, pois podem ser resumidas em elementos, suas relações e o resultado dessas relações [16]. No tocante à sua utilidade, arquiteturas auxiliam em [32]:

- **Comunicação:** Além dos desenvolvedores, gerentes de desenvolvimento e arquitetos; partes interessadas como gerentes de projeto; testadores; documentadores e até mesmo clientes precisam entender o sistema, logo, modelá-lo em alto nível facilita a comunicação;

- **Fornecimento do contexto do sistema:** Os desenvolvedores e futuros mantenedores precisam entender o sistema como um todo. Em grandes sistemas, é difícil para os desenvolvedores compreenderem todos os seus os detalhes de forma eficiente. Eles precisam de uma compreensão detalhada dos elementos em que estão trabalhando, todavia, a falta do entendimento de estruturas relacionadas pode reduzir a qualidade do sistema em termos de desempenho, disponibilidade e de outros atributos de qualidade;

- **Alocação de trabalho:** É possível particionar o trabalho dos desenvolvedores de forma eficiente a partir da decomposição do sistema em estruturas relativamente independentes, com responsabilidades claras e que se comunicam entre si com o uso de interfaces bem definidas;

- **Redução dos custos de manutenção e evolução:** Arquiteturas podem minimizar esses custos com a antecipação de possíveis mudanças no sistema, garantindo que o projeto do mesmo facilite tais mudanças e documentando como elas podem afetar os elementos arquiteturais envolvidos;

- **Reuso e integração com sistemas legados e softwares de terceiros:** Uma arquitetura pode ser projetada para permitir e facilitar o reuso de componentes, arcabouços, bibliotecas, sistemas legados e softwares de terceiros.

2.2 Partes Interessadas

Uma parte interessada é um indivíduo, time, organização, ou classes dos anteriores que possuem interesse em um sistema [25]. Seguem abaixo os tipos mais comuns de partes interessadas:

- **Usuários finais:** indivíduos que irão de fato utilizar o sistema. Dependendo do projeto, o cliente também pode ser considerado um usuário final;

- **Clientes:** parte interessada fora das fronteiras de uma determinada organização ou unidade organizacional. Eles fazem uso dos produtos ou serviços produzidos pela organização e podem possuir direitos morais ou contratuais que a organização é obrigada a atender [33];

- **Desenvolvedores:** indivíduos que irão desenvolver o sistema a partir de restrições impostas pela arquitetura e por outras partes interessadas;
- **Testadores:** indivíduos que realizarão testes no sistema a fim de verificar o correto comportamento do mesmo;
- **Implantadores:** responsáveis por implantar o sistema e torná-lo operacional;
- **Gerentes de desenvolvimento:** responsáveis pela coordenação da equipe de desenvolvimento de software, escolha das tecnologias, orientação da equipe de programadores e controle dos processos e tarefas [34];
- **Gerentes de projeto:** responsáveis pelo cronograma e atribuição de recursos a fim de garantir uma solução que atenda à necessidade do negócio;
- **Arquitetos:** indivíduos responsáveis pela definição da arquitetura e, mais formalmente, pela produção da descrição arquitetural [35]. Os arquitetos devem garantir que todas as outras partes interessadas estejam satisfeitas e precisam realizar negociações a fim de resolver problemas causados por requisitos concorrentes.

2.3 Requisitos

Um requisito pode ser definido como [33]:

- Uma condição ou capacidade necessária para uma parte interessada resolver um problema ou atingir um objetivo;
- Uma condição ou capacidade que deve ser alcançada ou possuída por uma solução, ou componente de solução, para satisfazer um contrato, padrão, especificação ou outros documentos formalmente impostos;
- Uma representação documentada de uma condição ou capacidade como em (1) ou (2).

No contexto deste trabalho será utilizada a classificação de requisitos contida em [33]. Seguem abaixo os tipos de requisitos:

- **Requisitos do negócio:** declarações das metas, objetivos e necessidades da empresa;

- **Requisitos de partes interessadas:** declaracões das necessidades de partes interessadas em particular e como essas partes irão interagir com a solução;
- **Requisitos da solução:** descrevem as características de uma solução que atende aos requisitos do negócio e aos requisitos das partes interessadas;
  - **Requisitos funcionais:** descrevem o comportamento e a informação que a solução irá gerenciar;
  - **Requisitos não-funcionais:** capturam condições que não se relacionam diretamente ao comportamento e funcionalidade da solução, mas descrevem condições ambientais sob as quais a solução deve permanecer efetiva, ou qualidades que os sistemas precisam possuir;
- **Requisitos de transição:** descrevem capacidades que a solução deve possuir com o objetivo de facilitar a transição do estado atual da organização para um estado futuro desejado.

Além dos tipos acima, existem **os requisitos arquiteturais**, que são um subconjunto dos **requisitos de solução**. Um requisito pode ser considerado arquitetural quando ele captura funcionalidades essenciais para o sistema; atinge vários elementos arquiteturais; possui exigências rigorosas à arquitetura, como por exemplo, requisitos de desempenho, disponibilidade e outros; ou envolve comunicação e sincronização com sistemas externos [36].

2.4 Decisões Arquiteturais

Uma decisão pode ser considerada arquitetural quando a mesma se propõe a alcançar um ou mais atributos de qualidade do sistema, por meio de estruturas ou regras que ela envolve ou define [15, 16].

Não existe uma regra decisiva para determinar se uma decisão é arquitetural ou não, embora deve-se investigar se tal decisão é relevante para atingir os requisitos gerais e se ela impacta diferentes partes do sistema [37]; caso positivo, ela é arquitetural.

2.5 Pontos de Vista e Visões Arquiteturais

Um ponto de vista é um arcabouço conceitual que define elementos, conexões e técnicas que compõem uma visão arquitetural, além de especificar seu propósito de acordo com seus interessados [16]. Já as visões expressam a arquitetura de um sistema ou de parte dele a partir da perspectiva de um conjunto de interesses relacionados [25].

Para aperfeiçoar o entendimento entre a relação de ponto de vista e visão, pode-se considerar linguagens de programação e programas: uma linguagem de programação pode ser utilizada para desenvolver diversos programas; assim como um ponto de vista, para elaborar diversas visões arquiteturais.

Com respeito a sua utilidade, o uso de visões facilita os processos de *design* e documentação da arquitetura, pois o arquiteto pode abstrair detalhes desnecessários à visão trabalhada no momento.

Capítulo 3

Modelo de Documento Arquitetural

Como foi mencionado no capítulo 1, documentar arquiteturas é importante para manter um registro dos motivos para decisões realizadas, facilitar o entendimento do sistema e agilizar a implementação. Não obstante, muitas vezes a documentação não é realizada ou, se realizada, não é preparada corretamente porque não é uma tarefa trivial, já que ela reflete a complexidade da arquitetura; reflete o tamanho da arquitetura e é difícil de se manter consistente com o sistema descrito [16].

O modelo proposto, que pode ser encontrado no apêndice A, pretende facilitar a documentação de arquiteturas por:

- Ser um modelo de documento simples, porém com as seções necessárias para retratar uma arquitetura de software;
- Ser um guia de como preparar arquiteturas de software: cada lacuna a ser preenchida é um item que deve ser pensado para a arquitetura em questão;
- Ter definições básicas para que os iniciantes na área de Arquitetura de Software e até mesmo os não-praticantes consigam capturar a essência do documento;
- Ser baseado nas recomendações de livros e especificações na área [15, 25, 26].

3.1 Requisitos do Modelo de Documento

Na produção do modelo de documento, os seguintes requisitos gerais foram considerados:

RG1 O modelo deve estar de acordo com as recomendações existentes no padrão ISO/IEC/IEEE 42010 [25];

RG2 Toda lacuna do modelo deve possuir uma breve explicação sempre que o seu conteúdo não for intuitivo;

RG3 O documento deve ser escrito de forma clara para que todas as partes interessadas tenham ao menos uma noção geral da arquitetura do sistema;

RG4 Deve ser genérico a fim de possibilitar a documentação de diferentes tipos de sistemas de software.

Houve também uma preocupação com a estrutura do documento. Os requisitos estruturais estão a seguir:

RE1 Possuir versão, autores e datas de modificação;

RE2 Possuir informações sobre o sistema em questão, o contexto em que ele está e será inserido, seus propósitos e quaisquer outras informações determinadas pelas partes interessadas;

RE3 Possuir definições de termos utilizados no documento;

RE4 Possuir listagem das partes interessadas;

RE5 Possuir listagem dos requisitos para o sistema;

RE6 Possuir listagem dos riscos identificados;

RE7 Todo requisito, risco e decisão-chave deve possuir identificador; descrição; requisitos ou riscos relacionados e partes interessadas;

RE8 Cada requisito funcional pode ser associado a casos de uso;

RE9 Cada requisito não-funcional deve possuir atributos de qualidade;

RE10 Cada risco deve possuir providências a serem tomadas;

RE11 Possuir pelo menos uma visão arquitetural;

RE12 Possuir um ponto de vista para cada visão arquitetural;

RE13 Cada ponto de vista deve possuir identificador; descrição; interesses, ou seja, detalhar o principal foco da visão associada; partes interessadas e tipos de modelo[1] aplicáveis;

RE14 Cada visão deve possuir um identificador; descrição; termos e definições utilizados no detalhamento da visão; decisões-chave e modelos desenvolvidos;

RE15 Cada decisão-chave deve ter suas vantagens e desvantagens elencadas.

3.2 Estrutura do Documento

O modelo de documento realizado é composto por:

1. **Capa:** contém o título do projeto; organização envolvida; local; mês e ano;
2. **Revisões:** possui informações sobre os autores envolvidos; alterações realizadas; número da versão e data da revisão;
3. **Índice[2]:** acesso rápido às seções do documento;
4. **Introdução:** provê informações sobre as referências utilizadas na preparação do documento e traz um resumo das seções existentes;
5. **Termos e definições:** contém definições para diversos termos utilizados: arquiteturas de software; partes interessadas; interesses; decisões-chave e outros;
6. **Sistema de interesse:** trata do contexto em que o sistema se encontra; seu escopo; propósitos; decisões-chave; partes interessadas e seus interesses;
7. **Pontos de vista:** cita os pontos de vista que serão utilizados como base na definição das visões. Para cada ponto de vista são informados seus interesses; as partes interessadas e os tipos de modelos que podem ser utilizados. O modelo traz os seguintes pontos de vista para referência: lógico; desenvolvimento; processos; físico e cenários;

[1] Os tipos de modelo podem ser UML [38], SysML [39], redes de Petri [40] e outros.

[2] Para os exemplos de documentos de arquitetura nos apêndices B e C, essa seção está compartilhada com o documento principal.

8. **Visões:** possui informações sobre as visões arquiteturais. O conjunto dessas visões define a arquitetura do sistema;

9. **Bibliografia[2]:** contém o registro de documentos utilizados na elaboração do documento.

3.3 Forma de Utilização

O documento é composto por várias lacunas, que são delimitadas pelos símbolos < e >. Os textos que não estão delimitados por esses símbolos podem ser alterados ou removidos quando não são aplicáveis à arquitetura do sistema em questão.

O modelo de documento pode ser utilizado para guiar o processo de criação da arquitetura. As informações foram dispostas para esse propósito. Os passos a serem seguidos para o preenchimento do modelo de documento são:

1. Definir termos específicos para o projeto na seção A.2.2 do documento;

2. Descrever o sistema, seu contexto, propósitos e outros aspectos na seção A.3;

3. Definir as partes interessadas envolvidas na seção A.3.1;

4. Definir requisitos funcionais, não-funcionais e riscos em A.3.2. Outros tipos de requisitos também podem ser documentados;

5. Criar novos pontos de vista; isso deve ser feito somente se necessário, pois o modelo de visões arquiteturais [41] escolhido é bastante utilizado e cobre bem as necessidades da documentação arquitetural. Além disso, é um arcabouço arquitetural válido com relação ao padrão ISO/IEC/IEEE 42010 [25];

6. Revisar as seções A.3 e A.4 para verificar se existe algum termo específico que necessite ser definido em A.2.2;

7. Para cada visão:

   (a) Dar uma breve descrição do que será tratado nessa visão;

(b) Definir termos necessários para o entendimento dos modelos, decisões-chave e outros no contexto da visão corrente;

(c) Preencher decisões-chave;

(d) Incluir modelos associados à visão e detalhá-los;

(e) Preencher **interfaces** para a visão lógica;

(f) Preencher **nodos x processos** para a visão física;

(g) Revisar a visão para verificar se existe algum termo específico que necessite ser definido.

3.3.1 O uso do modelo em processos de desenvolvimento

Utilizar o modelo de documento em processos existentes não é uma tarefa complexa. Sugere-se que a documentação seja escrita aos poucos, de forma iterativa e incremental. Para que o uso do modelo seja ideal é necessária a identificação das seguintes fases no processo a ser utilizado:

- **Entendimento geral sobre o sistema em questão:** Nessa fase, as seções A.2.2 e A.3 devem ser preenchidas;
- **Análise de requisitos:** Os requisitos coletados devem ser preenchidos em A.3.2. Outros tipos de requisitos também podem ser acrescentados nessa seção. Ademais, deve-se tentar identificar possíveis **riscos** para o projeto;
- **Implementação do sistema:** Antes de começar o desenvolvimento do sistema é importante que **decisões-chave** sejam tomadas, **interfaces** sejam definidas e **diagramas** sejam produzidos. É interessante também ter uma ideia geral dos tipos de **nodos de *hardware*** em que o sistema será instalado e quais serão os **processos** em execução nos mesmos;
- **Implantação do sistema:** Os **nodos de *hardware*** e **processos em execução** devem ser revisados para garantir que a documentação da arquitetura esteja consistente com a realidade.

Por fim, a arquitetura deve ser revisada durante cada fase identificada acima, para que a quantidade de mudanças na documentação seja menor e a mesma se mantenha atualizada.

3.3.2 Versão inicial da arquitetura

Uma versão inicial da arquitetura pode ser desenvolvida para auxiliar a negociação do escopo e valores do sistema com o cliente. Essa versão do documento arquitetural pode conter:

- Descrição breve do sistema de interesse;
- Partes interessadas do sistema;
- Especificação dos requisitos funcionais, não-funcionais e riscos, incluindo identificadores, descrições e justificativas;
- Definição de decisões-chave importantes, incluindo identificadores, descrições e justificativas;
- Criação de um diagrama com a visão geral do sistema e uma breve descrição dos principais componentes envolvidos.

Capítulo 4

Método de Validação de Arquiteturas de Software

A validação é uma etapa vital no processo de criação de sistemas computacionais. Essa atividade ajuda a responder a pergunta "Estamos construindo o produto certo?". Para isso, deve-se considerar tanto requisitos funcionais quanto não-funcionais. O custo necessário para realizar mudanças devido a especificações incorretas é menor quando a validação for realizada o quanto antes.

A arquitetura é o produto da fase de *design* e o seu efeito sobre o sistema e o projeto é profundo, ou seja, uma arquitetura inadequada pode inviabilizar a satisfação de requisitos de desempenho, segurança, disponibilidade e outros. Visto isso, o propósito deste capítulo é descrever um método para validar se uma arquitetura atende aos requisitos arquiteturais definidos.

4.1 Artefatos Envolvidos

Artefato: Requisitos arquiteturais

Descrição: Listagem dos requisitos arquiteturais de forma que cada requisito arquitetural contenha (i) identificador; (ii) descrição; (iii) interesses relacionados; (iv) partes interessadas; (v) casos de uso (opcional); (vi) modelos (opcional).

Propósito: Os requisitos, ou parte deles, listados nesse artefato serão confrontados com as decisões-chave relacionadas a fim de validar se a arquitetura os atendem ou não.

Artefato: Decisões-chave

Descrição: Listagem das decisões-chave de forma que cada uma contenha: (i) identificador; (ii) descrição; (iii) justificativa (opcional); (iv) alternativas existentes (opcional); (v) vantagens; (vi) desvantagens; (vii) modelos.

Propósito: As decisões-chave identificam o que deve ser feito para satisfazer um ou mais requisitos e serão utilizadas na etapa de obtenção de resultados a fim de coletar métricas e, por sua vez, checar se estão dentro de um limite aceitável ou não.

Artefato: Tabela de requisitos arquiteturais x decisões-chave

Descrição: Consiste em uma tabela de duas colunas em que os valores da primeira coluna são os identificadores dos requisitos arquiteturais e os valores da segunda são as decisões-chave associadas.

Propósito: A obtenção de resultados será feita para cada requisito arquitetural e, com o auxílio da tabela, todas as decisões-chave relacionadas a esse requisito serão avaliadas.

Artefato: Matriz de relacionamento entre requisitos arquiteturais

Descrição: Matriz M quadrada de tamanho n cujo elemento M_{ij} deve ser preenchido com:

- X se existir relacionamento entre os requisitos i e j;
- Vazio se não existir.

Propósito: A matriz de relacionamento é utilizada caso seja necessário realizar novamente a etapa de obtenção dos resultados para alguns requisitos, pois os requisitos relacionados também devem ser incluídos.

Artefato: Requisitos arquiteturais priorizados pelo grau de importância

Descrição: Consiste em uma tabela de duas colunas em que os valores da primeira coluna são os identificadores dos requisitos arquiteturais e os valores da segunda coluna são os graus de importância. Os requisitos de maior grau deverão aparecer primeiro.

Propósito: Esse artefato auxilia a escolha de melhorar ou não as decisões-chave associadas a um ou mais requisitos caso as métricas obitidas não atinjam o limiar estabelecido.

Artefato: Lista de métricas, limites e técnicas x requisito arquitetural

Descrição: Consiste em uma tabela de quatro colunas: (i) requisito arquitetural; (ii) métricas; (iii) limites; (iv) técnicas.

Propósito: Essa listagem guia a etapa de obtenção de resultados, identificando quais técnicas serão utilizadas para avaliar cada requisito arquitetural; quais as métricas a serem obtidas e os limites aceitáveis.

Artefato:	Documento de preparação da técnica de medição
Descrição:	Esse artefato possui os dados necessários para descrever em detalhes como a técnica de medição será utilizada para validar o requisito arquitetural associado. Deve existir um documento para cada requisito arquitetural e o seu conteúdo varia dependendo da técnica escolhida.
Propósito:	Juntamente com o artefato anterior, guia a obtenção dos resultados para cada requisito arquitetural.

Artefato:	Lista resumida de requisito arquitetural x técnicas x resultados
Descrição:	A tabela é composta por três colunas: (i) requisito arquitetural; (ii) técnica; (iii) resultados. Nesse documento deve existir uma tabela para cada iteração realizada.
Propósito:	O artefato em questão é gerado na etapa de obtenção de resultados e ajuda na elaboração do relatório de validação arquitetural.

Artefato: Relatório de validação arquitetural

Descrição: O relatório deve conter as seguintes seções:

1. Sistema de interesse;
2. Requisitos arquiteturais: cada requisito deve possuir pelo menos o identificador, a descrição o grau de importância e os requisitos relacionados;
3. Decisões-chave: pelo menos o identificador e a descrição para cada uma;
4. Validação: (i) métricas; (ii) limites; (iii) técnicas; (iv) resultados obtidos; (v) conclusões;
5. Conclusão final.

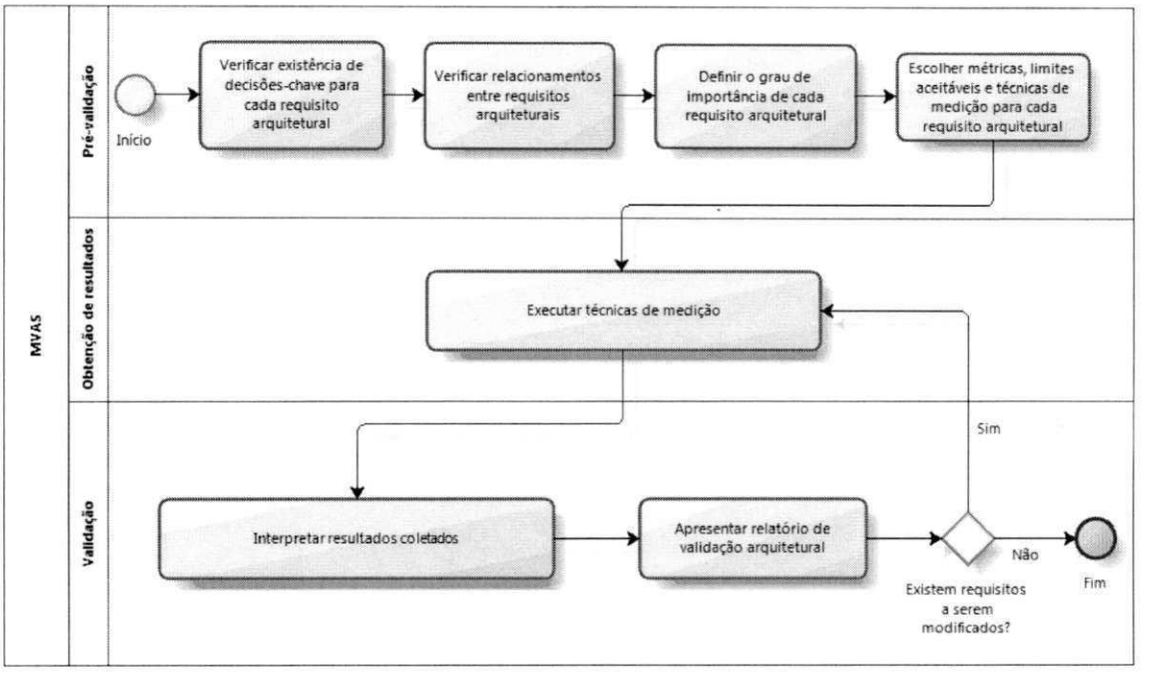
Propósito: O relatório documenta todo o processo de validação e é apresentado aos clientes a fim de gerar uma discussão sobre o que deve ser melhorado ou não na arquitetura, com a possibilidade de uma nova obtenção de resultados.

4.2 Fases do Método

O MVAS é dividido em três fases: pré-validação, obtenção de resultados e validação. A primeira fase é constituída pela verificação da existência de decisões-chave para os requisitos arquiteturais; definição dos relacionamentos entre eles; decisão do grau de importância dos mesmos e escolha das métricas; limites aceitáveis e técnicas de medição. A segunda fase é definida pela execução das técnicas escolhidas para cada requisito arquitetural. A terceira fase trata da checagem da satisfação ou não das métricas coletadas com relação aos limites definidos e da apresentação do relatório de validação arquitetural.

As duas últimas fases são iterativas e devem ser executadas novamente caso existam requisitos que não atinjam os limites mínimos exigidos e a decisão tomada seja melhorar a arquitetura. Segue abaixo uma figura que representa o processo a ser seguido:

Figura 4.1: Resumo do MVAS



As etapas do método serão explicadas nas próximas subseções. As entradas de uma etapa são os artefatos necessários para iniciá-la, já as saídas são os artefatos gerados a partir da sua execução.

4.2.1 Pré-validação

Atividade: Verificar existência de decisões-chave para cada requisito arquitetural

Passos: Checar se existe pelo menos uma decisão-chave para cada requisito arquitetural:

- Se existir, passar para o requisito arquitetural seguinte;
- Se não existir, deve-se verificar se: (i) o requisito é realmente arquitetural; (ii) uma decisão-chave não foi tomada.

Entradas: Requisitos arquiteturais; decisões-chave.

Saídas: Tabela de requisitos arquiteturais x decisões-chave.

Atividade: Verificar relacionamentos entre requisitos arquiteturais

Passos:

1. Criar uma matriz quadrada M de tamanho n para relacionar os requisitos arquiteturais;

2. Para cada par de requisitos arquiteturais, verificar se existe algum relacionamento entre eles: (i) se existir, preencher o elemento equivalente da matriz com X; (ii) caso contrário, deixar o elemento equivalente da matriz vazio.

Entradas: Requisitos arquiteturais.

Saídas: Matriz de relacionamento entre requisitos arquiteturais.

Atividade: Definir o grau de importância de cada requisito arquitetural

Passos: Associar um grau de importância a cada requisito:

- **Alto:** caso o requisito seja crucial para o funcionamento do sistema;
- **Médio:** caso o requisito seja importante mas talvez não valha a pena melhorar as decisões-chave caso os limites aceitáveis não sejam atingidos;
- **Baixo:** caso o requisito não seja tão crucial para o funcionamento do sistema.

Entradas: Requisitos arquiteturais.

Saídas: Requisitos arquiteturais priorizados pelo grau de importância.

Atividade: Escolher métricas, limites aceitáveis e técnicas de medição para cada requisito arquitetural

Passos: Para cada requisito arquitetural:

1. Escolher as métricas;
2. Definir limites aceitáveis;
3. Escolher técnicas para coleta das métricas;
4. Elaborar um documento de preparação para a técnica de medição escolhida.

Entradas: Requisitos arquiteturais.

Saídas: Lista de métricas, limites e técnicas x requisito arquitetural; documentos de preparação das técnicas de medição.

4.2.2 Obtenção de Resultados

Atividade: Executar técnicas de medição

Passos: Para cada requisito arquitetural:

1. Executar as técnicas escolhidas;
2. Coletar os resultados.

Entradas: Requisitos arquiteturais; decisões-chave; tabela de requisitos arquiteturais x decisões-chave; lista de métricas, limites e técnicas x requisito arquitetural; matriz de relacionamento entre requisitos arquiteturais; documentos de preparação das técnicas de medição.

Saídas: Lista resumida de requisito arquitetural x técnicas x resultados.

4.2.3 Validação

Atividade: Interpretar resultados coletados

Passos: Para cada requisito arquitetural, verificar se os valores obtidos atingem os limites aceitáveis definidos:

- Se atingirem, prosseguir com os próximos requisitos;
- Se não atingirem deve-se checar o grau de importância do requisito: (i) se for alto, é aconselhável tentar melhorar as decisões; (ii) se for médio, é aconselhável avaliar se vale a pena efetuar mudanças nas decisões-chave; (iii) se for baixo, é aconselhável aceitar o resultado e prosseguir para o próximo requisito arquitetural.

Entradas: Lista resumida de requisito arquitetural x técnicas x resultados; lista de métricas, limites e técnicas x requisito arquitetural; requisitos arquiteturais priorizados pelo grau de importância.

Saídas: Relatório de validação arquitetural.

Atividade: Apresentar relatório de validação arquitetural

Passos:

1. Apresentar os resultados obtidos utilizando o relatório de validação arquitetural;
2. Para cada requisito que não atingiu o limite aceitável é necessário discutir se os requisitos ou se as decisões-chave serão modificadas, considerando as recomendações no relatório;
3. Deve-se executar a fase de validação novamente para os requisitos arquiteturais cujas decisões-chave foram alteradas. Os requisitos relacionados também serão considerados na nova validação.

Entradas: Relatório de validação arquitetural; requisitos arquiteturais priorizados pelo grau de importância.

Saídas: Relatório de validação arquitetural.

4.3 Técnicas de Medição

Nas seguintes subseções serão descritas três técnicas que podem ser utilizadas para obter métricas de diferentes tipos de requisitos arquiteturais.

4.3.1 Cenários

Cenários[1] são um conjunto de usos ou modificações a um sistema. As modificações podem ser na forma em que um ou mais componentes realizam alguma ação; na adição de um componente; na adição de uma conexão entre dois componentes existentes ou na combinação de uma dessas situações. Ao criar e organizar cenários é importante que todos os papéis relevantes ao sistema sejam considerados, já que decisões de *design* podem ter sido feitas para permiti-los. Diferentes papéis representam diferentes partes interessadas do sistema. Tais partes podem ser usuários finais, responsáveis pela execução do software; administradores do sistema, responsáveis pelo gerenciamento dos repositórios de dados utilizados pelo

[1] Os cenários podem ser gerados para requisitos de segurança; proteção; confiabilidade; manutenibilidade e reusabilidade.

sistema; desenvolvedores, responsável por modificar funções de execução do sistema ou a própria organização, responsável por aprovar novos requisitos [42].

O uso de cenários é vantajoso pela simplicidade e rapidez na sua elaboração e execução; pela compatibilidade com diversos requisitos arquiteturais e por tentar possibilitar o "uso"de um sistema que provavelmente ainda não existe. Ao documentarmos tal uso em formas de cenários, é possível extrair informações que serão úteis no futuro para embasar decisões arquiteturais realizadas.

Por outro lado, é difícil estimar a quantidade ideal de cenários preparados para validar os requisitos. Sugere-se que os cenários elaborados contemplem casos usuais e também casos críticos. Se os cenários forem executados corretamente em ambos os casos, a chance de problemas posteriores será possivelmente menor. Além disso, na execução dos cenários é recomendado citar os elementos arquiteturais que estão envolvidos em cada passo dos mesmos, para facilitar o entendimento e aumentar a confiança nessa execução.

Métrica 1

Seja n o número de cenários preparados, r_i o ranking obtido ao executar o cenário i, $1 \leq i \leq n$. Com essa métrica é possível ter uma ideia do grau de satisfacao que o sistema dará. A métrica m é definida por:

$$m = e \sum_{i=1}^{n} \frac{r_i}{n} \tag{4.1}$$

O valor de r_i está no intervalo $[0, k], k \geq 1$. Para o caso de $k = 1$, considera-se que $r_i = 0$ significa que o cenário não foi executado com sucesso e $r_i = 1$ significa que houve sucesso. Para $k > 1$, definições qualitativas associadas a cada valor podem ser consideradas, por exemplo: (i) 0: falha fatal na execução; (ii) 1: erro aceitável na execução; (iii) 2: sucesso.

O valor de e pode ser 0, caso exista algum cenário crucial que falhou na execução, ou 1 caso nenhum cenário crucial tenha falhado. Com o uso dessa variável, pode-se garantir que o limite aceitável para essa métrica não seja atingido caso existam cenários cuja execução não possa falhar.

Métrica 2

Com base em [43], pode-se considerar como métrica o número de mudanças necessárias no sistema para que um cenário seja executado com sucesso. Essa métrica é interessante para tentar prever o custo de possíveis alterações no produto a ser desenvolvido. A métrica m é definida por:

$$m = \sum_{i=1}^{n} d_i \tag{4.2}$$

n representa o número de mudanças necessárias para os cenários e d_i é o nível de dificuldade da alteração i, devendo ser maior ou igual a 1. Definições qualitativas podem ser associadas a essa variável, por exemplo: (i) 1: alteração simples; (ii) 2: alteração complexa.

O valor de m está no intervalo $[0, d_{max}n]$, $1 \leq i \leq n$. d_{max} representa o nível máximo de dificuldade e, assim como d_i, deve ser maior ou igual a 1.

4.3.2 Simulações

Os componentes principais de uma arquitetura são implementados e outros componentes são simulados resultando em um sistema executável. O contexto, em que o sistema deve ser executado, também pode ser simulado em um nível de abstração aceitável. Esta implementação pode então ser utilizada para simular o comportamento da aplicação diante de várias circunstâncias [30].

Simuladores[2] são vantajosos quando se deseja validar a arquitetura principalmente com relação ao seu desempenho, viabilizando a descoberta de possíveis gargalos do sistema. Todavia, deve-se atentar à dificuldade de estimar os parâmetros de entrada do mesmo, por exemplo:

- Como determinar a distribuição de probabilidade que representa o acesso de clientes?
- Como determinar a distribuição de probabilidade que representa o tempo de execução de uma determinada tarefa?
- O que eu posso abstrair do meu sistema?

[2] O uso de simuladores é recomendado para validar requisitos de desempenho e de tolerância a falhas.

O uso de dados históricos facilita a resposta das duas primeiras perguntas anteriores, mas não é aplicável se o sistema ou uma versão legada do mesmo não existir. Com isso, faz-se necessário supor tais distribuições, diminuindo a confiança dos resultados obtidos na simulações.

Diversas métricas de desempenho podem ser utilizadas, como tempo de resposta e vazão, ideais para mensurar sistemas que exigem baixos tempos de resposta diante de muitos acessos simultâneos, como sites de grande acesso; quantidade de memória utilizada e percentual de CPU utilizado, interessantes para dimensionar a quantidade de recursos que devem ser utilizados para suportar as cargas do sistema, além de ajudar na identificação de gargalos de processamento e gastos excessivos de memória.

4.3.3 *Checklists*

Um *checklist*[3] é uma lista de itens a serem verificados para garantir consistência e completude ao realizar atividades complexas. Pode ser usado também na checagem da satisfação de requisitos arquiteturais a partir de decisões-chave. Cada item dessa lista pode ser uma pergunta a ser respondida, de forma que uma resposta positiva seja a esperada.

Assim como cenários, *checklists* podem ser elaborados e respondidos rapidamente. Têm como vantagem a necessidade de conhecimento do domínio por parte do elaborador, o que o força a estudar mais para que a lista de itens a ser criada seja significativa. Todavia, *checklists* também compartilham desvantagens com os cenários, ou seja, é difícil determinar as quantidades ideais de itens dos mesmos.

Métricas

Seja n o número de questões. Cada questão q_i pode ter uma pontuação máxima $pmax_i$, uma pontuação p_i, de forma que $pmax_i > 0, 0 \leq pi \leq pmax_i, 1 \leq i \leq n$. A métrica m pode ser calculada como:

$$m = \sum_{i=1}^{n} p_i \tag{4.3}$$

[3] *Checklists* são aplicáveis a requisitos de portabilidade; testabilidade; auditabilidade; integração com outros produtos; segurança; proteção; confiabilidade; manutenibilidade; reusabilidade e de tolerância a falhas.

O valor de m estará no intervalo $[0, \sum_{i=1}^{n} pmax_i]$. Caso um valor entre $[0, 1]$ seja melhor ao aplicar a técnica, pode-se utilizar:

$$m = \left(\sum_{i=1}^{n} \frac{p_i}{pmax_i} \right) / n \tag{4.4}$$

Capítulo 5

Validação da Solução

A validação do trabalho foi dividida em quatro etapas. A primeira etapa foi garantir que os requisitos definidos para o modelo de documento foram atingidos. A segunda foi utilizar o modelo para documentar a arquitetura do sistema SmartSwitch. A seguir, documentou-se a arquitetura do sistema PSI com o uso do modelo. Por último, a arquitetura gerada para o PSI foi validada com o método de validação proposto no trabalho.

5.1 Checagem dos Requisitos para o Modelo de Documento

Para garantir que os requisitos gerais e estruturais do modelo foram atingidos, cada um deles foi avaliado. Seguem abaixo as informações levantadas nessa atividade:

RG1 As principais recomendações existentes no padrão ISO/IEC/IEEE 42010 [25] foram atendidas, porém não serão listadas aqui por questões de direitos autorais;

RG2 As lacunas, delimitadas por < e >, possuem explicações quando necessário. Por exemplo, a seção de requisitos possui informações mais extensas sobre como preenchê-la, ao contrário da seção de partes interessadas, pois seu conteúdo é mais simples;

RG3 O modelo foi escrito com uma linguagem simples e existem definições para termos que não são de conhecimento geral. Além disso, o uso de visões evita que os leitores

precisem ler todo o documento de arquitetura para entender aspectos que os interessem;

RG4 Não há seções no documento que sejam aplicáveis a tipos de softwares específicos. E, no caso de sistemas que necessitem de explicações específicas, o modelo pode ser estendido com a adição de novas seções;

RE1 A versão, autores e datas de modificação do documento podem ser encontradas na seção dedicada para revisões do documento;

RE2 As informações gerais sobre o sistema estão em A.3;

RE3 As definições de termos estão na seção A.2 e, para cada visão, existe uma seção específica para tais definições;

RE4 As partes interessadas estão descritas em A.3.1;

RE5 A listagem dos requisitos é definida na seção A.3.2;

RE6 Os riscos identificados podem ser lidos na seção A.3.2;

RE7, RE8 Um requisito funcional possui identificador; descrição; interesses relacionados; partes interessadas e casos de uso;

RE7, RE9 Um requisito não-funcional possui identificador; atributos de qualidade; descrição; interesses relacionados e partes interessadas;

RE7, RE10 Um risco possui identificador; descrição; interesses relacionados; providências a serem tomadas e partes interessadas;

RE11 O modelo de documento possui seções para especificar várias visões arquiteturais;

RE12 O modelo de documento descreve cinco pontos de vista;

RE13 As seções A.4.1 a A.4.5 são relativas aos pontos de vista;

RE14 As visões estão detalhadas em A.5;

RE7, RE15 Cada decisão-chave possui identificador; interesses relacionados; descrição; vantagens e desvantagens.

5.2 Arquitetura do SmartSwitch

O SmartSwitch é um sistema para geração de manobras no setor eletroenergético; sejam elas geradas de forma automática ou manual, por meio da interação com a interface gráfica. Além da geração, o sistema também é responsável pela auditoria de manobras. Trata-se de um sistema de interface *web*, o que facilita o acesso por parte dos operadores e viabiliza o aprendizado dos mesmos [28].

Com base em uma reunião realizada entre o autor e as partes interessadas e em um documento preliminar da arquitetura preparado por Camilla Falconi, uma versão inicial do documento arquitetural foi escrita. Seguindo a primeira versão, mais seis foram realizadas. Por fim, o documento – localizado no apêndice B – foi liberado para a Companhia Hidro Elétrica do São Francisco (Chesf) como parte dos relatórios do projeto a serem entregues.

5.3 Arquitetura do PSI

O Portal de Sistemas Integrados (PSI) é um arcabouço, composto por várias bibliotecas de software, desenvolvido para auxiliar na produção de novos sistemas institucionais para a Universidade Federal de Campina Grande. O seu objetivo principal é ser um ponto único de acesso que permitirá o uso de diversos sistemas pelos alunos e servidores da instituição, de forma que a autenticação seja feita somente uma vez.

Os requisitos do PSI foram capturados a partir de reuniões com os analistas de TI do Serviço de Tecnologia de Informação e da experiência do autor como aluno e funcionário da instituição. Com isso, duas versões do documento arquitetural foram geradas e o mesmo se encontra no apêndice C.

5.4 Validação da Arquitetura do PSI

A validação do PSI foi realizada após a finalização da primeira versão do documento arquitetural do mesmo. Foram escolhidos nove requisitos arquiteturais para serem validados. Em seguida, constatou-se que todos os requisitos escolhidos possuem decisões-chave associadas a eles. A partir disso, quatro requisitos foram definidos com grau **alto** de importância, três com grau **médio** de importância e os dois restantes com grau **baixo**.

Após a definição dos graus de importância, as técnicas de medição, métricas e os limites aceitáveis foram escolhidos. O uso de cenários, *checklists* e simuladores foi determinado para, respectivamente, dois, cinco e dois requisitos arquiteturais. Com isso, a primeira iteração relativa à captura das métricas foi realizada.

Constatou-se que três requisitos não atingiram os limites mínimos. Visto isso, esses requisitos e suas decisões-chave associadas foram alterados a fim de obter valores melhores na segunda iteração. Na nova iteração, cinco requisitos foram aferidos, devido à necessidade de aferir também os requisitos associados aos três anteriores.

Ao final da segunda iteração, um requisito ainda não atingiu o limite mínimo, mas ele não foi alterado por ter um grau baixo de importância, não valendo a pena o esforço de um novo conjunto de medições. Para finalizar a validação, o relatório de validação arquitetural foi produzido e pode ser lido no apêndice D.

5.5 Melhorias Realizadas

Com o conhecimento obtido por meio das atividades relatadas nas seções anteriores, o modelo de documento e o MVAS foram aperfeiçoados. As alterações realizadas foram:

- No modelo de documento:
  - Adição de **providências a serem tomadas** na parte dos riscos identificados;
  - Adição de um modelo para **interfaces** no caso da visão lógica;
  - Adição de uma tabela de relacionamento entre interesses e decisões-chave. Essa tabela facilita a visualização dos interesses e das decisões-chave; além de ser uma forma preliminar de validar a arquitetura por checar se existem decisões para cada interesse;
  - Adição de um modelo para **nodos x processos** no caso da visão de processos;
- No método de validação:
  - Adição de uma figura com a visão geral do processo para facilitar a sua execução;
  - Criação de modelos de documento para todos os artefatos envolvidos no processo;

- Criação de um novo artefato para detalhar como uma técnica de medição será utilizada para validar o requisito arquitetural associado: documento de preparação da técnica de medição;
- Adição de novas métricas para as técnicas de medição;
- Melhoria na descrição dessas técnicas.

Capítulo 6

Discussão Geral

Durante o processo de validação do trabalho, foram encontrados resultados positivos, negativos e algumas limitações da solução. As seções a seguir tratam dessas informações.

6.1 Resultados

Ao utilizar o modelo de documento, houve uma redução no tempo demandado para decidir sobre o que deve possuir um documento arquitetural, pois o modelo já possui as seções necessárias a serem preenchidas. O tempo gasto para explicar aspectos do sistema aos participantes do projeto também foi menor, porque muitas informações necessárias já se encontravam no documento arquitetural.

A documentação se tornou mais confiável porque o modelo funcionou como um *checklist*, ou seja, preenchendo todas as lacunas exigidas foi possível reduzir a chance de que itens importantes fossem deixados de lado. Por fim, a definição de vários termos básicos facilitou a leitura dos participantes do projeto ao evitar: (i) o uso de termos desconhecidos por leitores menos experientes nas áreas de Engenharia e Arquitetura de Software; (ii) falhas no entendimento por não explicar termos que possuem várias significados na literatura.

Com relação ao MVAS, os artefatos produzidos durante a validação serviram como registro de memória sobre como o sistema foi validado e o que foi melhorado no sistema, já que todas as etapas do processo envolvem a produção de pequenos documentos. Além disso, o uso de várias técnicas de medição facilitou a validação de diferentes tipos de requisitos arquiteturais, pois existem requisitos que são mais simples para validar dependendo da técnica

escolhida.

Utilizar *checklists*, uma das técnicas de medição propostas, foi simples, confiável e de baixo custo por garantir que detalhes importantes não fossem deixados de lado ao tomar decisões-chave para satisfazer um requisito. Para preparar checklists foi necessário o entendimento maior sobre o assunto tratado, com isso, houve maiores chances de encontrar vantagens, desvantagens e riscos para as decisões-chave. Ainda, é possível reusar, com poucos ajustes, um *checklist* para o mesmo tipo de requisito em diferentes sistemas.

Por outro lado, alguns resultados negativos também foram encontrados. A visão lógica nos documentos produzidos ficou mais detalhada, sendo assim, usuários da solução proposta que necessitem de um foco maior em outras visões precisam pesquisar mais sobre como preenchê-las, o que irá demandar mais tempo dos mesmos. Na validação do PSI, o uso de um simulador foi custoso e as taxas de entrada não foram determinadas a partir de dados reais, o que tornou os dados obtidos menos confiáveis.

6.2 Limitações

A maior limitação do estudo é que tanto o modelo de documento quanto o método de validação foram utilizados, em grande parte, somente pelo autor. Porém, espera-se que esses artefatos sejam usados por mais pessoas e que os resultados obtidos por elas possam gerar sugestões sobre como e em que a solução proposta deve ser ajustada e aperfeiçoada.

No modelo de documento não há uma seção para especificar o relacionamento entre elementos arquiteturais de diferentes visões; por isso, o modelo de documento não está totalmente de acordo com as exigências do padrão ISO/IEC/IEEE 42010 [25]. Já no MVAS não há um método preciso para estimar a quantidade de itens no *checklist* e nos cenários a serem gerados para um certo requisito.

Capítulo 7

Trabalhos Relacionados

Com o levantamento bibliográfico na área de Arquiteturas de Software, foi possível encontrar trabalhos relacionados ao que foi produzido nesse estudo. Nas seções 7.1, 7.2 e 7.3 são descritos, respectivamente, modelos de documentos arquiteturais, documentos arquiteturais e métodos de validação.

7.1 Modelos de Documentos Arquiteturais

Em [17] se encontra um modelo de documento bem completo e detalhado. Nele, sugere-se a explicação de cada diagrama exibido, com exemplos interessantes no mesmo. Além disso, existe um apêndice mostrando a sua conformidade com uma versão preliminar [44] do que se tornou o padrão ISO/IEC/IEEE 42010 [25]. Embora date de 2000, foi o modelo de documento mais completo encontrado e, com ajustes mínimos, estaria em conformidade com a versão mais recente desse padrão. Não obstante, o documento é disponibilizado em PDF, o que dificulta ou até mesmo inviabiliza a sua utilização; não há uma seção específica para listar as decisões-chave para a arquitetura e não há delimitadores para especificar o que deve ser preenchido ou não.

O modelo [18] foi desenvolvido seguindo as diretrizes do livro [15]. É uma versão disponibilizada em páginas *wiki* de [45], o que facilita o acesso à versão mais recente da arquitetura e ainda mantém versões anteriores das páginas. Assim como o modelo produzido neste trabalho, as lacunas preenchidas estão delimitadas por < e > e, existem explicações para aquelas cujo preenchimento não seja intuitivo. Por outro lado, a definição de termos co-

muns utilizados no documento é curta, o que pode dificultar o entendimento para iniciantes que desejam documentar arquiteturas de software. Essas explicações breves se repetem nas seções, o que força o arquiteto a possuir um bom conhecimento prévio antes de começar a utilizar esse modelo.

No levantamento, um modelo em português [19] foi encontrado, que por sua vez é uma tradução de [20]. Os conteúdos a serem preenchidos no documento estão delimitados por [e] e possuem explicações sobre como devem ser escritos. Por fim, é um modelo bem simples e pode ser melhor utilizado por arquitetos experientes, porque o mesmo só possui as seções a serem preenchidas e pequenas explicações, não contendo definições de termos importantes; exemplos de como as seções devem ser preenchidas e nem mesmo uma seção para referências do estudo.

Na tabela 7.1 é apresentado um comparativo entre os modelos I [17], II [18], III [20] e IV, proposto nesse trabalho. A comparação foi realizada a partir de treze itens e os mesmos podem ser lidos nessa tabela.

7.2 Exemplos de Documentos Arquiteturais

Existem muitos exemplos de documentos na literatura, mas nem todos são bons guias de como realizar uma documentação, portanto, somente uma amostra será tratada aqui.

A arquitetura em [21] é bem descrita; as seções estão bem separadas e vários aspectos importantes são tratados, como: (i) ideia geral do sistema, assim como seu escopo; (ii) requisitos funcionais e não-funcionais; (iii) casos de uso; (iv) múltiplas visões arquiteturais; (v) preocupações com desempenho e segurança; (vi) definições de termos técnicos. Por outro lado, nos diagramas existentes não há explicações mais específicas do que é tratado por eles, dificultando o entendimento.

Já o documento [22] não é tão bem dividido quanto o primeiro, pois não há uma boa separação entre as várias preocupações arquiteturais, além de não descrever o sistema em visões. Em suma, o sistema em questão está bastante detalhado, mas a leitura do documento é um pouco complicada por não haver seções específicas para tratar de requisitos, decisões-chave, visões arquiteturais e outras informações.

A descrição arquitetural localizada em [23] possui uma divisão melhor com relação ao

Tabela 7.1: Tabela comparativa entre os modelos de documento

Item	**Modelo I**	**Modelo II**	**Modelo III**	**Modelo IV**
Revisões do documento	Não	Sim	Sim	Sim
Índice	Sim	Sim	Sim	Sim
Lacunas para preenchimento bem delimitadas	Não	Parcialmente	Sim	Sim
Lacunas para preenchimento com explicações	Sim	Sim	Sim	Sim
Facilmente reutilizável	Não	Sim	Sim	Sim
Seção dedicada à explicação do sistema proposto	Sim	Sim	Sim	Sim
Definições de termos utilizados	Sim	Parcialmente	Não	Sim
Listagem e definição dos pontos de vista	Sim	Não	Não	Sim
Modelo de acordo com o padrão ISO/IEC/IEEE 42010 [25]	Parcialmente	Não	Não	Parcialmente
Definição dos requisitos com detalhes sobre o que deve ser descrito	Sim	Não	Não	Sim
Definição dos riscos	Sim	Não	Não	Sim
Definição das decisões-chave	Sim	Sim	Não	Sim
Divisão em visões arquiteturais	Sim	Sim	Sim	Sim

segundo exemplo, porém não são descritos os requisitos relacionados a cada decisão-chave. Além disso, não há uma seção específica contendo as explicações para cada termo importante utilizado no documento.

A partir da leitura dos exemplos acima e outros, nota-se que padrões para documentação de arquitetura não são seguidos, já que cada exemplo lido possui uma forma bem distinta para documentá-las. Esse problema pode ser decorrente da inexistência ou inexperiência de arquitetos propriamente ditos[1] para os sistemas ou até mesmo a própria defasagem no ensino de arquiteturas de software.

7.3 Métodos de Validação

Os trabalhos que tratam de métodos de validação para arquiteturas são *Architecture Trade-Off Analysis Method* (ATAM) [24], *Scenario-Based Architecture Reengineering* (SBAR) [30], *Architecture Level Prediction of Software Maintenance* (ALPSM) [46], *Software Architecture Evaluation Model* (SAEM) [47], *Scenario-Based Architecture Analysis Method* (SAAM) [13], *SAAM Founded on Complex Scenarios* (SAAMCS) [48], *Extending SAAM by Integration in the Domain* (ESAAMI) [49], *Software Architecture Analysis Method for Evolution and Reusability* (SAAMER) [50]. Os métodos que mais se assemelham ao proposto no trabalho são SAAM, SBAR e ATAM.

O SAAM foi desenvolvido originalmente para comparar diferentes soluções arquiteturais a fim de determinar qual arquitetura atende às necessidades de uma organização. Considera-se que o atributo de qualidade avaliado pelo SAAM é a modificabilidade, ou seja, o quão fácil é realizar alterações no sistema. Esse método já foi aplicado em sistemas de tráfego aéreo, ambientes de desenvolvimento para interfaces de usuários, sistemas de informação *web* e outros [42]. A desvantagem desse método é que o mesmo não se aplica a qualquer tipo de requisito arquitetural.

Com o uso de SBAR é possível submeter os requisitos a várias técnicas, como cenários; modelagem matemática; simuladores e outros dependendo do atributo de qualidade. O foco do SBAR é estimar o potencial da arquitetura projetada para atingir os seus requisitos de

[1] Utilizou-se essa expressão com a finalidade de referenciar casos em que os próprios desenvolvedores são incubidos de documentar a arquitetura do sistema.

qualidade. Nesse método, o envolvimento de muitas partes interessadas não é obrigatório [42]. O MVAS possui uma abordagem semelhante ao SBAR por permitir o uso de várias técnicas de medição e por checar se requisitos são ou não atendidos pela arquitetura; mas as explicações do último não são bastante claras para casos em que cenários não são exercidos.

O método ATAM deve ser empregado quando se deseja identificar as consequências causadas por decisões arquiteturais em função das necessidades de atributos de qualidade [24]. Assim como o SBAR e o MVAS, permite o uso de várias técnicas de medição, resultando na validação de vários tipos de requisitos, contudo, foi inicialmente proposto para medir atributos de qualidade, segurança, desempenho e disponibilidade. Ao contrário do SBAR, o ATAM é melhor descrito, e em [24] é apresentado um exemplo de seu emprego em um sistema chamado *Battlefield Control System (BCS)*, utilizado por batalhões do exército para controle estratégico das tropas em tempo real nos campos de batalha.

Para finalizar, na tabela 7.2 é feita uma comparação entre alguns métodos de validação na literatura e o MVAS. Os critérios e os resultados são baseados em [42]: (i) técnicas para avaliação utilizada; (ii) atributos de qualidade; (iii) envolvimento das partes interessadas; (iv) fase em que o método é aplicado; (v) quando parar a geração de cenários.

Tabela 7.2: Tabela comparativa entre os métodos de validação

Critério	**SAAM**	**SBAR**	**ATAM**	**MVAS**
Técnicas para avaliação utilizada	Cenários	Cenários; modelagem matemática; simuladores ou raciocínio objetivo	Integra técnicas existentes de questionamento e medição	Cenários; *checklists* ou simuladores
Atributos de qualidade	Modificabilidade	Vários	Vários	Vários
Envolvimento das partes interessadas	Todos	Arquiteto	Todos ou somente o arquiteto	Todos ou somente o arquiteto
Fase em que o método é aplicado	Após a versão final da arquitetura de software	Combinado com o *design* da arquitetura	Após a versão final da arquitetura de software ou combinado com o *design* da arquitetura	Após a versão final da arquitetura de software
Quando parar a geração de cenários	Quando a adição de um novo cenário não causar mudanças no projeto	Sugere o uso de um conjunto completo ou representativo dos cenários	Utiliza um conjunto padrão de questões específicas para atributos de qualidade	Sugere o uso de um conjunto que contemple casos triviais e também casos críticos

Capítulo 8

Conclusão

Neste estudo foi produzido um conjunto de artefatos para guiar a documentação e validação arquiteturas de software. Esse conjunto é composto por um modelo de documentação, um método para validação, exemplos de documentos arquiteturais e tudo o que foi produzido para a validação de uma arquitetura. Todos esses artefatos podem ser encontrados em `https://code.google.com/p/exemplosarquiteturais/` sob a licença *Creative Commons 3.0* [51], permitindo a cópia, distribuição, adaptação e extensão do conteúdo por parte de novos autores.

Esse modelo de documento foi desenvolvido com a finalidade de ser simples, porém completo e de fácil entendimento para todas as partes interessadas no processo de definição da arquitetura. O método de validação foi escrito de forma bem detalhada para garantir que não existam maiores dúvidas em sua aplicação. Além disso, o método permite diversas técnicas de medição, garantindo que diferentes tipos de requisitos arquiteturais sejam validados. Ademais, os documentos foram produzidos a partir de problemas reais e utilizados na Chesf e na UFCG.

A validação do modelo de documento foi realizada a partir da construção e documentação de duas arquiteturas. O MVAS, por sua vez, foi validado com a sua utilização na arquitetura de software do Portal de Sistemas Integrados. Finalmente, foram realizadas melhorias em todos os artefatos a partir da experiência coletada na validação e de comentários feitos pelos envolvidos em todo o processo.

Em suma, este trabalho traz uma contribuição na área de arquiteturas de software com as abordagens desenvolvidas e exemplos detalhados. Os artefatos produzidos vão guiar o arqui-

teto na documentação e validação da arquitetura, todavia, não é garantido que a mesma não terá problemas, afinal, o modelo de documento foi realizado para guiar sua documentação e não para impor tecnologias a serem utilizadas e decisões a serem tomadas. Já o método de validação é utilizado para tentar garantir que as decisões arquiteturais satisfazem os requisitos definidos; mas há a possibilidade da existência de requisitos incorretos ou esquecidos, decisões-chave não tomadas ou de inconsistências entre a arquitetura e o código produzido.

8.1 Trabalhos Futuros

Com o conhecimento obtido durante todo o estudo, foi possível identificar os pontos fortes e fracos das abordagens desenvolvidas. Por isso, há a necessidade de elencar algumas sugestões de melhoria no estudo:

- O modelo de documento pode ser alterado para seguir todas as recomendações do padrão ISO/IEC/IEEE 42010 [25];
- Ainda no modelo, podem ser inclusas seções relativas a casos de uso e itens específicos de visões; além de um maior detalhamento nas lacunas a serem preenchidas, com exemplos no próprio modelo;
- Novas técnicas de medição e métricas associadas podem ser adicionadas ao MVAS;
- Os artefatos produzidos devem ser utilizados em novos sistemas por pessoas diferentes a fim de aumentar a confiabilidade dos mesmos.

Além de melhorias no que foi desenvolvido, propõe-se uma melhoria em todo o processo de definição arquitetural, pois manter a documentação arquitetural sincronizada com o código não é simples. Visto isso, o desenvolvimento de algumas ferramentas de software poderia auxiliar nessa tarefa:

- Uma ferramenta para criação e exportação de documentos arquiteturais, de forma que partes interessadas; interesses; decisões-chave; pontos de vista; visões e quaisquer elementos arquiteturais sejam gerenciados na mesma. Essa ferramenta poderia garantir a consistência no uso de termos, checar se existem decisões-chave para os requisitos, dentre outras formas de validação;

- Uma ferramenta para associar trechos de código a elementos arquiteturais, verificar se interfaces definidas na documentação estão de acordo com o que foi especificado na arquitetura e outras possibilidades.

Referências Bibliográficas

[1] BUXTON, J. N.; RANDELL, B. *Software Engineering Techniques: Report on a Conference Sponsored by The NATO Science Committee, Rome, Italy, 27th to 31st October 1969*. [S.l.]: NATO Science Committee, 1970.

[2] KRUCHTEN, P.; OBBINK, H.; STAFFORD, J. The past, present, and future for software architecture. *Software, IEEE*, 2006, IEEE, v. 23, n. 2, p. 22–30, 2006.

[3] WITT, B. I.; BAKER, F. T.; MERRITT, E. W. *Software Architecture and Design: Principles, Models, and Methods*. [S.l.]: John Wiley & Sons, Inc., 1993.

[4] VLISSIDES, J. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. [S.l.: s.n.], 1995.

[5] SHAW, M.; GARLAN, D. *Software Architecture: Perspectives on an Emerging Discipline*. [S.l.]: Prentice Hall, 1996.

[6] RECHTIN, E.; MAIER, M. *The Art of Systems Architecting*. [S.l.]: CRC Books, 1997.

[7] BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. 1. ed. [S.l.]: Addison-Wesley Professional, 1997.

[8] HOFMEISTER, C.; NORD, R.; SONI, D. *Applied Software Architecture*. [S.l.]: Addison-Wesley, 1999.

[9] PERRY, D. E.; WOLF, A. L. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 1992, ACM, v. 17, n. 4, p. 40–52, 1992.

[10] RAYMOND, K. Reference model of open distributed processing (RM-ODP): Introduction. In: BRISBANE, AUSTRALIA, CHAPMAN AND HALL. *IFIP TC6 International Conference on Open Distributed Processing*. [S.l.], 1995. p. 3–14.

[11] SHAW, M.; CLEMENTS, P. A field guide to boxology: Preliminary classification of architectural styles for software systems. In: IEEE. *Computer Software and Applications Conference, 1997. COMPSAC'97. Proceedings., The Twenty-First Annual International*. [S.l.], 1997. p. 6–13.

[12] LINDEN, F. J. V. D.; MULLER, J. K. Creating architectures with building blocks. *Software, IEEE*, 1995, IEEE, v. 12, n. 6, p. 51–60, 1995.

[13] KAZMAN, R. et al. Saam: A method for analyzing the properties of software architectures. In: IEEE COMPUTER SOCIETY PRESS. *Proceedings of the 16th international conference on Software engineering*. [S.l.], 1994. p. 81–90.

[14] KAZMAN, R. et al. The architecture tradeoff analysis method. In: IEEE. *Engineering of Complex Computer Systems, 1998. ICECCS'98. Proceedings. Fourth IEEE International Conference on*. [S.l.], 1998. p. 68–78.

[15] CLEMENTS, P. et al. *Documenting Software Architectures: Views and Beyond*. 2. ed. [S.l.]: Addison-Wesley Professional, 2010.

[16] BARBOSA, G. M. G. *Um Livro-texto para o Ensino de Projeto de Arquitetura de Software*. Dissertação (Mestrado) — Universidade Federal de Campina Grande, 2009.

[17] OGUSH, M. A.; COLEMAN, D.; BERINGER, D. A template for documenting software and firmware architectures. 2000, 15 mar. 2000. Disponível em: <http://www.iso-architecture.org/ieee-1471/applying-the-standard.html>. Acesso em: 17 mai. 2013.

[18] MERSON, P. Software architecture documentation template. 2006, 28 jul. 2006. Disponível em: <https://wiki.sei.cmu.edu/sad/>. Acesso em: 17 mai. 2013.

[19] SOFTWARE Architecture Document (SAD). Disponível em: <http://www.dca.ufrn.br/ anderson/FTP/dca0120/modelodocarquiteturasoftware.doc>. Acesso em: 17 mai. 2013.

[20] SCRUMUP. Software architecture document. Disponível em: <http://www.scrumup.com/downloads/Software%20Architecture%20Document%20Template%20-%20ScrumUP%20(v1.00).doc.>. Acesso em: 17 mai. 2013.

[21] MCDOWELL, A. Software architecture documentfor the sola development snapshot. 2011, 15 ago. 2011. Disponível em: <http://flossola.org/sites/default/files/fao_floss_sola_software_architecture_document_-_development_snapshot_v1.1.pdf>. Acesso em: 17 mai. 2013.

[22] INTEL CORPORATION. Linux system software for theinfiniband architecture. 2002, 01 ago. 2002. Disponível em: <http://infiniband.sourceforge.net/LinuxSAS.1.0.1.pdf>. Acesso em: 17 mai. 2013.

[23] SOUTHWEST FLORIDA WATER MANAGEMENT DISTRICT. Comprehensive watershed managementwater use tracking project: Software architecture document. 2007, 13 fev. 2007. Disponível em: <http://www.plato-consulting.com/Software%20Architecture%20Document.pdf>. Acesso em: 17 mai. 2013.

[24] KAZMAN, R.; KLEIN, M.; CLEMENTS, P. *ATAM: Method for architecture evaluation*. [S.l.], 2000.

[25] ISO/IEC/IEEE. *Systems and software engineering — Architecture description*. 2011.

[26] BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. 2. ed. [S.l.]: Addison-Wesley Professional, 2003.

[27] EELES, P.; CRIPPS, P. *The process of software architecting*. [S.l.]: Addison-Wesley Professional, 2009.

[28] CRISPIM, C. F. *Geração Automática de Manobras para Sistemas Eletroenergéticos*. Dissertação (Mestrado) — Universidade Federal de Campina Grande, 2013.

[29] MICHAEL, J. B.; RIEHLE, R.; SHING, M.-T. The verification and validation of software architecture for systems of systems. In: IEEE. *System of Systems Engineering, 2009. SoSE 2009. IEEE International Conference on*. [S.l.], 2009. p. 1–6.

[30] BENGTSSON, P.; BOSCH, J. Scenario-based software architecture reengineering. In: IEEE. *Software Reuse, 1998. Proceedings. Fifth International Conference on*. [S.l.], 1998. p. 308–317.

[31] GARLAN, D.; PERRY, D. E. Introduction to the special issue on software architecture. *IEEE Transactions on software engineering*, 1995, Citeseer, v. 21, n. 4, p. 269–274, 1995.

[32] BREDEMEYER CONSULTING. Motivating software architecture. 2012, Bloomington, 11 jul. 2012. Disponível em: <http://www.bredemeyer.com/why.htm>. Acesso em: 11 mai. 2013.

[33] IIBA. *A guide to the Business Analysis Body of Knowledge (BABOK guide)*. 2. ed. Toronto: International Institute of Business Analysis, 2009.

[34] TI MASTER. Gerente de desenvolvimento - TI Master. Disponível em: <http://www.timaster.com.br/revista/raiox/raiox.asp?prof=4&entrevista=nao>. Acesso em: 27 abr. 2013.

[35] EMERY, D. Applying IEEE 1471: Thoughts on an architecting process. 2011, 13 out. 2011. Disponível em: <http://www.iso-architecture.org/ieee-1471/applying-the-standard.html>. Acesso em: 11 mai. 2013.

[36] BREDEMEYER CONSULTING. Architectural requirements in the visual architecting process. 2013, Bloomington, 26 abr. 2013. Disponível em: <http://www.bredemeyer.com/ArchitectingProcess/ArchitecturalRequirements.htm>. Acesso em: 11 mai. 2013.

[37] MALAN, R.; BREDEMEYER, D. Software architecture: Central concerns, key decisions. 2005, Bloomington, 28 abr. 2005. Disponível em: <http://www.bredemeyer.com/pdf_files/ArchitectureDefinition.PDF>. Acesso em: 11 mai. 2013.

[38] OBJECT MANAGEMENT GROUP. Introduction To OMG's Unified Modeling Language™ (UML®). 2013, 14 fev. 2013. Disponível em: <http://www.omg.org/gettingstarted/what_is_uml.htm>. Acesso em: 18 mai. 2013.

[39] OBJECT MANAGEMENT GROUP. OMG SysML. Disponível em: <http://www.omgsysml.org/>. Acesso em: 18 mai. 2013.

[40] MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 1989, IEEE, v. 77, n. 4, p. 541–580, 1989.

[41] KRUCHTEN, P. B. The 4+1 view model of architecture. *Software, IEEE*, 1995, IEEE, v. 12, n. 6, p. 42–50, 1995.

[42] DOBRICA, L.; NIEMELA, E. A survey on software architecture analysis methods. *Software Engineering, IEEE Transactions on*, 2002, IEEE, v. 28, n. 7, p. 638–653, 2002.

[43] KAZMAN, R. et al. Scenario-based analysis of software architecture. *Software, IEEE*, 1996, IEEE, v. 13, n. 6, p. 47–55, 1996.

[44] IEEE. *IEEE P1471/D5.1 Draft Recommended Practice for Architectural Description*. 1999.

[45] SOFTWARE ENGINEERING INSTITUTE. Software architecture document (sad). 2005, 05 fev. 2005. Disponível em: <http://www.sei.cmu.edu/architecture/SAD_template2.dot>. Acesso em: 17 mai. 2013.

[46] BENGTSSON, P.; BOSCH, J. Architecture level prediction of software maintenance. In: IEEE. *Software Maintenance and Reengineering, 1999. Proceedings of the Third European Conference on*. [S.l.], 1999. p. 139–147.

[47] DUEÑAS, J. C.; OLIVEIRA, W. L. de; JUAN, A. A software architecture evaluation model. In: *Development and Evolution of Software Architectures for Product Families*. [S.l.]: Springer, 1998. p. 148–157.

[48] LASSING, N.; RIJSENBRIJ, D.; VLIET, H. van. On software architecture analysis of flexibility, complexity of changes: Size isn't everything. In: *Proc. Second Nordic Software Architecture Workshop NOSA*. [S.l.: s.n.], 1999. v. 99, p. 1103–1581.

[49] MOLTER, G. Integrating saam in domain-centric and reuse-based development processes. In: CITESEER. *Proceedings of the 2nd Nordic Workshop on Software Architecture, Ronneby*. [S.l.], 1999. p. 1–10.

[50] LUNG, C.-H. et al. An approach to software architecture analysis for evolution and reusability. In: IBM PRESS. *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*. [S.l.], 1997. p. 15.

[51] CREATIVE COMMONS. Creative Commons — Attribution 3.0 Unported — CC BY 3.0. Disponível em: <http://creativecommons.org/licenses/by/3.0/>. Acesso em: 16 mai. 2013.

[52] ISO/IEC. *Systems and software engineering — System life cycle processes*. 2008.

[53] THOMPSON, S. Systems designer: Job description and activities. Disponível em: <http://ww2.prospects.ac.uk/p/types_of_job/systems_designer_job_description.jsp>. Acesso em: 21 mai. 2013.

[54] TRAORE, I. Part 3.3: Implementation, process, and deployment views. Disponível em: <http://www.ece.uvic.ca/%7Eitraore/seng422-06/notes/arch06-3-3.pdf>. Acesso em: 21 mai. 2013.

[55] SAUVÉ, J. P. Projeto Arquitetural do Sistema SIEx.

[56] HEYWOOD, R. Uml use case diagrams: Tips and faq. 1999, 21 abr. 1999. Disponível em: <http://www.andrew.cmu.edu/course/90-754/umlucdfaq.html>. Acesso em: 21 mai. 2013.

[57] ARAUJO, A. S. et al. SisRTM - Sistema de Roteiro de Manobras.

[58] DHAKAL, P. Computer aided design of substation switching schemes. 2000, 2000.

[59] PEREIRA, L. A. C. et al. SAGE - um sistema aberto para a evolução.

[60] ORACLE. Remote method invocation home. Disponível em: <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html>. Acesso em: 22 mai. 2013.

[61] CODIGOFONTE.NET. Códigofonte.net > códigos de javascript. Disponível em: <http://www.codigofonte.net/scripts/javascript>. Acesso em: 22 mai. 2013.

[62] 4LINUX FREE SOFTWARE SOLUTIONS. O que é LDAP e o que são serviços de diretórios? Disponível em: <http://www.4linux.com.br/que-ldap-que-sao-servicos-diretorios.html>. Acesso em: 22 mai. 2013.

[63] MENASCE, D. A. et al. *Performance by design: computer capacity planning by example*. [S.l.]: Prentice Hall PTR, 2004.

[64] HOWELL, F.; MCNAB, R. Simjava: A discrete event simulation library for java. *Simulation Series*, 1998, Citeseer, v. 30, p. 51–56, 1998.

[65] ORACLE. What are the system requirements for java 7? Disponível em: <http://www.java.com/en/download/help/sysreq.xml>. Acesso em: 27 mai. 2013.

[66] IBM. Developer kits, user guides and service information for java standard edition on aix. Disponível em: <http://www.ibm.com/developerworks/java/jdk/aix/service.html>. Acesso em: 27 mai. 2013.

[67] ORACLE. Learn about java technology. Disponível em: <http://www.java.com/en/about/>. Acesso em: 27 mai. 2013.

[68] ATRICORE. Who is using josso. Disponível em: <http://www.josso.org/confluence/display/JOSSO1/Who+is+using+JOSSO>. Acesso em: 27 mai. 2013.

[69] ATRICORE. Josso - java open single sign-on project home. Disponível em: <http://www.josso.org/confluence/display/JOSSO1/JOSSO+-+Java+Open+Single+Sign-On+Project+Home>. Acesso em: 27 mai. 2013.

[70] ATRICORE. Using your own login form. Disponível em: <http://www.josso.org/confluence/display/JOSSO1/Using+your+own+Login+Form>. Acesso em: 27 mai. 2013.

[71] ATRICORE. Josso - browse /josso at sourceforge.net. Disponível em: <http://sourceforge.net/projects/josso/files/JOSSO/>. Acesso em: 27 mai. 2013.

[72] FIELDING, R. T. Chapter 5: Representational state transfer (REST). Disponível em: <http://www.ics.uci.edu/ fielding/pubs/dissertation/rest_arch_style.htm>. Acesso em: 27 mai. 2013.

[73] JBOSS COMMUNITY. RESTEasy - JBoss Community. Disponível em: <http://www.jboss.org/resteasy>. Acesso em: 27 mai. 2013.

[74] JBOSS COMMUNITY. RESTEasy - browse files at sourceforge.net. Disponível em: <http://sourceforge.net/projects/resteasy/files/>. Acesso em: 27 mai. 2013.

[75] ZEILENGA, K. Lightweight directory access protocol (ldap): Technical specification road map. 2006, 2006.

[76] FREIER, A.; KARLTON, P.; KOCHER, P. The secure sockets layer (ssl) protocol version 3.0. 2011, 2011.

Apêndice A

Modelo de Documentação de Arquitetura de Software

<Nome da organização>

<Nome do projeto>

Documento de Arquitetura de Software

<Local>

<Mês>, <Ano>

Revisões

Autores	Descrição	Versão	Data
<Autor 1>, <Autor 2>, ..., <Autor n>	<Resumo das modificações feitas>	<Número da versão>	<Data da revisão>

A.1 Introdução

Este documento corresponde à documentação arquitetural do sistema <Nome do projeto>. A sua estrutura é baseada nas recomendações existentes no padrão ISO/IEC/IEEE 42010 [25] e é composta pelas seguintes seções:

1. **Introdução:** provê informações sobre as referências utilizadas na preparação deste documentação e trás um resumo das seções existentes.
2. **Termos e definições:** contém definições de diversos termos utilizados neste documento.
3. **Sistema de interesse:** trata do contexto em que o sistema se encontra, seu escopo, propósitos, decisões-chave, partes interessadas e seus interesses.
4. **Pontos de vista:** cita os diversos pontos de vista que serão utilizados como base na definição das visões. Para cada ponto de vista são informados seus interesses, as partes interessadas e os tipos de modelos que podem ser utilizados.
5. **Visões:** possui informações sobre as visões arquiteturais. O conjunto dessas visões define a arquitetura do sistema.
6. **Bibliografia:** contém o registro de documentos utilizados neste estudo.

A.2 Termos e Definições

A.2.1 Engenharia de Software

Esta seção tem como objetivo evitar possíveis dúvidas e mal entendidos gerados pela falta de consenso na definição de diversos termos na área de engenharia de software.

Sistema

No contexto deste documento, arquiteturas ajudam na definição de "**sistemas** que são produzidos por homens e podem ser configurados com um ou mais: hardware, software, dados, humanos, processos (ex.: processos provendo serviços a usuários), procedimentos (ex: instruções para um operador), instalações, materiais e entidades que ocorrem naturalmente." [52]

Arquitetura de Software

A **arquitetura** de software de um programa ou sistema computacional é a estrutura das **estruturas do sistema**, que por sua vez é composta por elementos de software, **propriedades externamente visíveis** desses elementos e as relações entre eles [26].

As **estruturas do sistema** são subdivididas em estruturas **estáticas** e **dinâmicas**. As estruturas **estáticas** podem ser módulos, classes, pacotes, procedimentos catalogados em bancos de dados, etc. Já as estruturas **dinâmicas** podem ser fluxos de informação entre elementos (A manda uma mensagem para B), execução paralela/sequencial de tarefas do sistema (C invoca uma rotina de um elemento D), etc.

As propriedades externamente visíveis podem ser de:

- **comportamento** (o que o sistema faz): interações entre o sistema e o seu ambiente, incluindo informações de entrada/saída do sistema ou o contrato/API que a arquitetura possui para "o mundo externo";
- **qualidade** (como o sistema faz): desempenho, segurança, escalabilidade, etc.

Arquiteturas de software auxiliam na comunicação entre partes interessadas, no fornecimento do contexto do sistema, na alocação de trabalho, na redução de custos de manutenção e evolução e também no reuso e integração com sistemas legados e softwares de terceiros.

Parte Interessada

Uma parte interessada é qualquer pessoa, time ou organização participante de um sistema. Segue abaixo uma pequena lista de possíveis partes interessadas:

- **Usuários finais:** indivíduos que realmente utilizarão o sistema. Dependendo do projeto, o cliente também pode ser considerado um usuário final;
- **Clientes:** aqueles que contrataram aprodução do sistema e que provavelmente realizam o pagamento do mesmo;
- **Desenvolvedores:** indivíduos que irão desenvolver o sistema a partir de restrições impostas pela arquitetura e por outras partes interessadas;

- **Testadores:** indivíduos que realizarão testes no sistema a fim de verificar o correto comportamento do mesmo. Diferentes tipos de testes podem ser encontrados aqui;
- **Implantadores:** responsáveis por implantar o sistema e torná-lo operacional;
- **Gerentes de desenvolvimento:** responsáveis pela coordenação da equipe de desenvolvimento de software, escolha das tecnologias, orientação da equipe de programadores e controle dos processos e tarefas [34];
- **Gerentes de projeto:** responsáveis pelo cronograma e atribuição de recursos;
- **Projetistas:** Um projetista trabalha sob direção de um gerente de projetos ou de um arquiteto e age no processo de design, desenvolvimento e implementação dos requisitos de um novo sistema de informação [53];
- **Arquitetos:** indivíduos responsáveis pela definição da arquitetura e, mais formalmente, pela produção da descrição arquitetural [35]. Os arquitetos devem garantir que todas as outras partes interessadas estejam satisfeitas. Parte do seu trabalho é fazer negociações a fim de resolver problemas causados por requisitos concorrentes.

Cada projeto tem partes interessadas específicas e as mesmas devem ser identificadas corretamente para que riscos sejam evitados.

Interesse

Um interesse pode ser manifestado de diversas maneiras com relação às necessidades de uma ou mais partes interessadas: objetivos, expectativas, responsabilidades, requisitos, restrições de projeto, suposições, dependências, atributos de qualidade, decisões arquiteturais, riscos ou outras questões pertinentes ao sistema [25].

Requisito Funcional

Um requisito funcional define uma função de um sistema de software ou de algum de seus componentes. Uma função é descrita como um conjunto de entradas, o comportamento e as saídas. Requisitos funcionais podem ser cálculos, detalhes técnicos, manipulação/processamento de dados ou qualquer funcionalidade especifica que defina **o que** o sistema deve realizar.

Requisito Não-funcional

Os requisitos não funcionais, também conhecidos como atributos de qualidade, são aqueles que expressam como algo deve ser feito. Em geral se relacionam com padrões de qualidade, como confiabilidade, performance, robustez, etc. São muito importantes pois definem se o sistema será eficiente para a tarefa que se propõe a fazer ou não. Um sistema ineficiente certamente não será usado. Neles também são apresentadas restrições e especificações de uso para os requisitos funcionais.

Decisão Arquitetural

De acordo com [16], uma decisão pode ser considerada arquitetural quando a mesma se propõe a alcançar um ou mais atributos de qualidade do sistema, por meio de estruturas ou regras que ela envolve ou define.

Não existe uma regra decisiva para determinar se uma decisão é arquitetural ou não, embora deve-se investigar se tal decisão é relevante para atingir os requisitos gerais do sistema; caso positivo, ela é arquitetural.

Ponto de Vista Arquitetural

É um arcabouço conceitual que define elementos, conexões e técnicas que compõem uma visão arquitetural, além de especificar seu propósito de acordo com seus interessados [16].

Para facilitar o entendimento entre a relação de ponto de vista e visão, podemos considerar linguagens de programação e programas: uma linguagem de programação pode ser utilizada para desenvolver diversos programas; assim como um ponto de vista pode ser utilizado para elaborar diversas visões arquiteturais.

Visão Arquitetural

Visões arquiteturais expressam a arquitetura de um sistema ou de parte dele a partir da perspectiva de um conjunto de interesses relacionados [25].

O uso de visões facilita os processos de design e documentação da arquitetura, pois o arquiteto pode abstrair detalhes desnecessários da visão em que ele esteja trabalhando no momento.

A.2.2 <Termo Específico do Projeto>

<Informar aqui a definição do termo específico. Para cada termo, deve existir uma seção. Se necessário, pode-se criar subseções para o caso de termos relativos a uma categoria específica, assim como foi feito na seção A.2.1.>

A.3 Sistema de Interesse

<Informar aqui o contexto em que o sistema se encontra.>
<Informar aqui o que o sistema se propõe a fazer.>

A.3.1 Partes Interessadas

Segue abaixo a listagem de todas as partes interessadas envolvidas no projeto:

Tipo	Nome
<Tipo da parte interessada - ver seção A.2.1>	<Nome da parte interessada>

A.3.2 Interesses

Esta seção documenta os interesses dos participantes, além de identificar quais deles possuem cada interesse.

Requisitos Funcionais

<Aqui são informados os requisitos funcionais do sistema. Se necessário, subseções podem ser criadas para categorizá-los.>

Identificador:	<Definir um nome para o requisito funcional. Sugere-se que o nome resuma a funcionalidade com o menor número de palavras possível.>
Descrição:	<Redigir uma pequena descrição da funcionalidade que deseja ser obtida. >
Justificativa:	<Descrever a justificativa para a exigência desse requisito>
Interesses relacionados:	<Listagem de interesses relacionados a este requisito funcional. A listagem deve ser por linha e sugere-se que o identificador seja utilizado.>
Partes interessadas:	<Listagem por linha das partes interessadas neste requisito. Sugere-se que somente o tipo seja informado: cliente, etc. Se todos os interesses tiverem partes interessadas em comum, pode-se comentar a parte interessada no início da seção e evitar a repetição desta listagem. Sendo assim, essa listagem pode ser utilizada somente naqueles requisitos em que as partes interessadas são diferentes.>
Casos de usos relacionados:	<Casos de uso relacionados>

Requisitos Não-funcionais

<Aqui são informados os requisitos não-funcionais do sistema. Se necessário, subseções podem ser criadas para categorizá-los.>

Identificador:	<Definir um nome para o requisito não-funcional. Sugere-se que o nome resuma o requisito com o menor número de palavras possível.>
Atributos:	<Colocar aqui os atributos relacionados a esse requisito não-funcional. Vários exemplos podem ser encontrados em https://en.wikipedia.org/wiki/Non-functional_requirement.>
Descrição:	<Redigir uma pequena descrição do que se deseja obter. Colocar também detalhes ou condições limite, por exemplo: máximo de dez minutos para o usuário aprender a fazer algo, tempo máximo de resposta de 2 segundos para tal operação, sistema deve funcionar em Windows Vista e 7, etc.>
Interesses relacionados:	<Listagem de interesses relacionados a este requisito funcional. A listagem deve ser por linha e sugere-se que o identificador seja utilizado.>
Partes interessadas:	<Listagem por linha das partes interessadas neste requisito. Sugere-se que somente o tipo seja informado: cliente, etc.>

Riscos

<Aqui são informados os riscos do sistema. Se necessário, subseções podem ser criadas para categorizá-los.>

Identificador:	<Definir um nome para o risco. Sugere-se que o nome resuma o risco.>
Descrição:	<Redigir uma pequena descrição do risco.>
Interesses relacionados:	<Listagem de interesses relacionados a este risco. A listagem deve ser por linha e sugere-se que o identificador seja utilizado.>
Providências a serem tomadas:	<Listagem de procedimentos que serão realizados para tentar mitigar o risco.>
Partes interessadas:	<Listagem por linha das partes interessadas neste risco. Sugere-se que somente o tipo seja informado: cliente, etc.>

A.4 Pontos de Vista

Os pontos de vista encontrados nesta seção são de autoria de Kruchten [41] e irão guiar o processo de elaboração das visões arquiteturais.

A.4.1 Lógico

O ponto de vista lógico trata das estruturas estáticas e do comportamento dinâmico do sistema.

Interesses

O interesse principal do ponto de vista lógico está nas funcionalidades do sistema.

Partes Interessadas

- **Clientes:** têm interesse nas funcionalidades existentes no sistema e qual o comportamento de cada uma;
- **Desenvolvedores:** têm interesse nas estruturas existentes no sistema, nas interfaces, suas operações, comportamentos esperados, nas tecnologias escolhidas para auxiliar a implementação das funcionalidades do sistema e realizam revisões de código;

- **Gerentes de desenvolvimento:** auxiliam os desenvolvedores no entendimento das estruturas do sistema, determinam quais funcionalidades devem ser feitas em cada iteração e realizam revisões de código;
- **Arquitetos:** realizam as decisões-chave auxiliados pelos gerentes de desenvolvimento, ajudam os desenvolvedores no entendimento das estruturas do sistema, verificam se a arquitetura está sendo implementada da forma esperada, determinam tecnologias a serem utilizadas, etc.

Tipos de modelo

Todos os diagramas abaixo fazem parte da linguagem de modelagem UML [38].

- **Diagramas de classes:** ilustram partes ou todas as classes e interfaces do sistema;
- **Diagramas de objetos:** mostram um conjunto de objetos e seus relacionamentos em um ponto no tempo;
- **Diagramas de sequência:** exibem interações entre objetos em uma sequência temporal;
- **Diagramas de colaboração:** mostram interações organizadas entre objetos e as ligações entre si. São organizados com a finalidade de enfatizar a organização estrutural;
- **Diagramas de estado:** permitem a modelagem do comportamento dentro de um único objeto.

A.4.2 Desenvolvimento

Concentra-se na organização real dos módulos no ambiente de desenvolvimento de software. O software é empacotado em pedaços pequenos - bibliotecas, subsistemas, etc. - que podem ser desenvolvidos por um ou um pequeno número de desenvolvedores [41].

Interesses

Os principais interesses do ponto de vista de desenvolvimento são organização, reuso e portabilidade.

Partes Interessadas

- **Desenvolvedores:** devem seguir a visão a fim de organizar o sistema como planejado;
- **Gerentes de desenvolvimento:** devem acompanhar o trabalho dos desenvolvedores a fim de garantir a separação correta dos módulos, verificar os prazos existentes, negociar com coordenadores e gerentes de projeto, etc;
- **Gerentes de projeto e coordenadores:** devem acompanhar se o desenvolvimento está de acordo com o cronograma.

Tipos de modelo

- **Diagramas de componentes:** componentes UML são partes físicas e substituíveis que disponibilizam e realizam um conjunto de interfaces [54];
- **Diagramas de pacotes:** ilustram as dependências entre pacotes que compõem um sistema.

A.4.3 Processos

Fornece uma decomposição dos módulos do sistema em processos (tarefas executáveis) e indica como *threads* são utilizados para controlar a execução dos elementos estruturais. [55]

Interesses

Os principais interesses do ponto de vista são desempenho, disponibilidade e tolerância a falhas.

Partes Interessadas

- **Gerentes de desenvolvimento:** interessados em verificar os *threads* existentes no sistema, identificar regiões críticas e repassar para os desenvolvedores.
- **Implantadores:** interessados em saber quais processos são executados pelo sistema, quantos *threads* são utilizados, etc.

Tipos de modelo

- **Diagramas de interação:** são um subconjunto dos diagramas comportamentais, enfatizam o fluxo de dados e controle entre as entidades modeladas no sistema. Podem ser diagramas de comunicação, de sequência, de interatividade ou de tempo.

A.4.4 Físico

O ponto de vista físico mostra como vários executáveis e outras entidades em tempo de execução são mapeadas em nodos de hardware.

Interesses

Os interesses principais são escalabilidade, disponibilidade e desempenho.

Partes Interessadas

- Projetistas

Tipos de modelo

- **Diagramas de implantação:** modelam a implantação física de artefatos em nós.

A.4.5 Cenários

A descrição de uma arquitetura é ilustrada usando um pequeno conjunto de casos de uso ou cenários, o que se torna um quinto ponto de vista. Os cenários descrevem sequências de interações entre objetos e processos e são usados para identificar elementos arquiteturais, ilustrar e validar o projeto arquitetural. Também servem como um ponto de início para testes de um protótipo de arquitetura [41].

Interesses

Os principais interesses são facilitar o entendimento da arquitetura, descobrir elementos arquiteturais, assim como uma forma de validar a arquitetura após finalizar o *design*.

Partes Interessadas

- **Usuários finais e clientes:** interessados em confirmar se os casos de uso e cenários representam o que o sistema realmente deve fazer.
- **Desenvolvedores:** interessados em um maior entendimento a partir dos casos de uso e de cenários.
- **Arquitetos:** interessados em validar a arquitetura.

Tipos de modelo

- **Casos de uso:** Podem ser utilizados para descrever a funcionalidade de um sistema em uma forma horizontal, ou seja, em vez de mostrar detalhes de características individuais do sistema, diagramas de caso de uso podem ser utilizados para mostrar todas as funcionalidades disponíveis [56];
- **Diagramas de sequência:** exibem interações entre objetos em uma sequência temporal.

A.5 Visões

A.5.1 Visão <Ponto de Vista Relacionado>

<Breve descrição do que será exibido nessa visão>
<Se possível, incluir um modelo que dê uma ideia geral do que será visto aqui.>

Termos e definições

<Incluir aqui definições de termos que serão necessários para o entendimento dos modelos, das decisões-chave, etc.>

Decisões-chave

<Incluir aqui decisões-chave tomadas com relação a esta visão. É interessante também colocar decisões que não foram tomadas e explicar os motivos, a fim de evitar que elas sejam feitas novamente no futuro, assim como manter um registro documentado das mesmas.>

<A tabela abaixo relaciona interesses com decisões-chave. Deve ser utilizada para facilitar a visualização dos interesses e decisões-chave de forma reduzida.>

Interesse	**Decisões-chave**
<Tipo do interesse (RF para requisito funcional, RNF para requisito não-funcional, RI para risco - Identificador do interesse.>	<Listagem das decisões-chave relacionadas ao interesse.>

Decisão-chave:	<Definir nome que resuma a decisão-chave>
Interesses relacionados:	<Listagem por linha dos interesses relacionados. Sugere-se informar o tipo e o identificador do interesse.>
Descrição:	<Descrição resumindo a decisão tomada.>
Justificativa:	<Justificativa da decisão tomada.>
Vantagens:	<Listagem por linha das vantagens da abordagem tomada.>
Desvantagens:	<Listagem por linha das desvantagens da abordagem tomada.>

Modelos

<Incluir aqui os modelos (seguindo os tipos especificados no ponto de vista), dando informações sobre os mesmos, interfaces, etc. Pode-se colocar cada modelo como uma subseção.>

Interfaces

<Esta seção deve ser utilizada na **visão lógica**. Ela contém detalhes sobre as interfaces existentes.>

Nome:	`<Nome da interface>`
Descrição:	<Breve descrição da interface>
Serviços providos:	<Listagem dos serviços providos no seguinte formato:

- Nome do serviço: breve descrição.>

Nodos x processos

<Esta seção deve ser utilizada na *visão física*. Ela contém o relacionamento entre nodos de hardware e processos em execução.>

Nodos de hardware	**Processos em execução**
<Identificadores dos nodos de hardware>	<Nomes dos processos em execução>

Apêndice B

Documento Arquitetural do Projeto SmartSwitch

Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Departamento de Sistemas e Computação

(UFCG/CEEI/DSC)

Companhia Hidro-Elétrica do São Francisco

(Chesf)

SmartSwitch

Documento de Arquitetura de Software

Campina Grande, Paraíba, Brasil

Janeiro, 2013

Revisões

Autores	Descrição	Versão	Data
Jacques Sauvé	Liberação como relatório do projeto SmartSwitch	1.6	30/01/2013
Eloi Rocha Neto Leandro José	Alterações na visão física.	1.5	05/12/2012
Leandro José	Alterações nos requisitos funcionais, não-funcionais e decisões-chave, remoção do cenário de exportação de manobras, alteração de outros cenários existentes, definição das operações de interfaces incompletas e atualização dos diagramas de componentes.	1.4	20/11/2012
Jacques Sauvé	Alterações pontuais de texto sobre módulos.	1.3	01/11/2012
Camilla Falconi Eloi Rocha Neto Jacques Sauvé Leandro José Pedro Sérgio Nicolletti	Inclusão de novas decisões-chave, novos requisitos funcionais e não-funcionais e alteração em diagramas decorrente da inclusão das propriedades do sistema.	1.2	06/07/2012
Camilla Falconi Eloi Rocha Neto Jacques Sauvé Leandro José Pedro Sérgio Nicolletti	Inclusão de termos e definições, novos diagramas para comportamento, novos interesses, correção das partes interessadas e seus interesses.	1.1	23/06/2012

Camilla Falconi Eloi Rocha Neto Jacques Sauvé Leandro José Pedro Sérgio Nicolletti	Versão inicial do documento de arquitetura de software do SmartSwitch.	1.0	11/06/2012

B.1 Introdução

O conteúdo desta seção, que pode ser lido em A.1, foi removido para evitar repetições no documento.

B.2 Termos e Definições

Parte do conteúdo desta seção, que pode ser lido em A.2, foi removido para evitar repetições no documento.

B.2.1 SisRTM

Sistema de Roteiro de Manobras - Sistema de Informação utilizado pela Chesf para gerenciamento de documentos referentes à operação do sistema elétrico.

B.2.2 *Snapshot*

Retrato de uma subestação do sistema elétrico, contendo todos os equipamentos da subestação e seus status.

B.2.3 GUI - *Graphical User Inteface*

Interface gráfica do sistema, usada para facilitar a comunicação com o usuário.

B.2.4 Manobra

Conjunto de operações que podem atuar sobre os dispositivos seccionadores - disjuntores e chaves seccionadoras; ou operações de aterramento de um equipamento, que atuam sobre as chaves-terra vinculadas a esses equipamentos; ou operações que fazem o ajuste de tap de transformadores.

B.2.5 PGM - Programa de Manobra

PGM são procedimentos que contemplam ações não padronizadas para liberação e normalização de equipamentos e linhas de transmissão [57].

B.2.6 RTM - Roteiro de Manobra

RTM são procedimentos que contemplam as ações padronizadas para liberação e normalização de equipamentos e linhas de transmissão [57].

B.3 Sistema de Interesse

Sistemas elétricos de potência são compostos por subsistemas de geração, transmissão e distribuição de energia elétrica. A geração de energia elétrica seja através de usinas nucleares, eólicas, hidrelétricas ou termoelétricas ocorre normalmente a longas distâncias dos grandes centros. Por isso, a energia gerada nessas usinas é transmitida através de linhas de transmissão para os centros de carga que estão espalhados por amplas áreas geográficas. O conjunto dessas linhas de transmissão forma redes interconectadas [58].

Prover energia aos consumidores em níveis aceitáveis de qualidade e de disponibilidade é um processo complexo. A rede elétrica está exposta a eventos aleatórios como sobrecarga de equipamento, falha de equipamento e problemas no sistema; mas, mesmo em situações como essas, o fornecimento de energia elétrica não pode ser interrompido. A continuidade do fornecimento de energia e a manutenção dos níveis de tensão dentro dos limites permitidos, são providos pela reconfiguração da rede elétrica, ou seja, através da execução de manobras e de ações de controle nas subestações [58].

Uma manobra é um conjunto de operações. Essas operações podem ser de abertura, de fechamento ou de ajuste, para mencionar apenas as principais. As operações de abertura e de fechamento são realizadas nos dispositivos seccionadores - disjuntores e chaves seccionadoras -, com o intuito de alterar os caminhos do fluxo de energia elétrica, para manter a continuidade do fornecimento de energia; e nas chaves-terra, para aterrar ou desaterrar um determinado equipamento. As operações de ajuste de tap são realizadas nos transformadores, para permitir a manutenção dos equipamentos da rede. [28]

As manobras devem ser bem planejadas, pois, operações inadequadas na rede, mesmo que mínimas, podem afetar negativamente grande parte do sistema elétrico. Antes da execução de uma manobra na rede, é feita uma análise prévia no sistema elétrico para prever possíveis conseqüências dessa manobra, de forma a garantir que essas operações não resultem em situações indesejadas, como por exemplo, sobrecarga de equipamentos, sobretensão

ou falhas. [28]

Na empresa Chesf, o processo de geração de PGMs é manual e está sujeito a erros de caráter técnico ou de digitação. Por ser composto por muitas etapas e ser executado por diferentes funcionários, é um processo demorado, a depender da complexidade da manobra e da quantidade de equipamentos envolvidos nessa manobra. Esses dois fatos aumentam a insegurança do sistema e as perdas associadas à indisponibilidade de equipamentos. [28]

O sistema SmartSwitch se propõe a gerar automaticamente programas de manobras que refletirão os principais cenários de manobras. Além disso, o sistema fará a avaliação de outros programas de manobras gerados ou editados manualmente pelos usuários. [28]

B.3.1 Partes Interessadas

Segue abaixo a listagem de todas as partes interessadas envolvidas no projeto:

Tipo	**Nome**
Usuários finais	Operadores de instalações e de centros regionais de operação.
	Os operadores são da pré-operação e da operação em tempo real.
Cliente	Sr. Denilson Silva dos Santos
Gerente de projeto	
Coordenador	Prof. Jacques Philippe Sauvé
Cliente *proxy*	Prof. Pedro Sérgio Nicolletti
Gerente de desenvolvimento	Eloi Rocha Neto
Arquitetos	Eloi Rocha Neto
	Leandro José Ventura Silva
Projetistas	Camilla Falconi Crispim
	Renato Almeida de Freitas
Desenvolvedores	Bruno Rafael Araújo Vasconcelos
	Irvile Rodrigues Lavor

Testadores	Bruno Rafael Araújo Vasconcelos (testes de unidade, testes de carga) Irvile Rodrigues Lavor (testes de unidade, testes de carga) Eloi Rocha Neto (testes de aceitação) Camilla Falconi (testes de aceitação)
Implantadores	Eloi Rocha Neto Funcionários do STI da Chesf (no futuro)

B.3.2 Interesses

Esta seção documenta os interesses dos participantes, além de identificar quais deles possuem cada interesse.

Requisitos Funcionais

Todos os requisitos **funcionais** a seguir são de interesse do **cliente**.

Login

Identificador:	Efetuar login
Descrição:	O usuário deve efetuar login para ter acesso ao Sistema SmartSwitch. O login do SmartSwitch é integrado ao login do Sistema SisRTM, logo, as informações de usuário e senha devem ser compatíveis.
Interesses relacionados:	Nenhum.

Snapshot

Identificador:	Tirar *Snapshot*
Descrição:	O usuário deverá tirar um *snapshot* da topologia atual da rede elétrica. O *snapshot* contém, essencialmente, o status (aberto ou fechado) dos equipamentos seccionadores (disjuntor e chave seccionador), se equipamentos estão energizados e/ou impedidos.
Interesses relacionados:	Nenhum.

Identificador:	Alterar *Snapshot*
Descrição:	Um *snapshot* poderá ser alterado pelo usuário, que pode alterar o status de um equipamento seccionador e/ou informar o impedimento de um ou mais equipamentos.
Interesses relacionados:	Nenhum.

Identificador:	Salvar *Snapshot*
Descrição:	Um *snapshot* pode ser salvo a qualquer momento, quer tenha sofrido alguma alteração ou não. Todos os usuários têm acesso à mesma lista de *snapshots* salvos.
Interesses relacionados:	Nenhum.

Identificador:	Buscar *Snapshot*
Descrição:	Os *snapshots* salvos, correspondentes a cada usuário, estão acessíveis e podem ser recuperados a qualquer momento.
Interesses relacionados:	Nenhum.

Identificador:	Excluir *Snapshot*
Descrição:	Um *snapshot* pode ser excluído a qualquer momento da lista de *snapshots* do usuário.
Interesses relacionados:	Nenhum.

Identificador:	Renomear *Snapshot*
Descrição:	Qualquer *snapshot* da lista de *snapshots* do usuário pode ter seu nome alterado.
Interesses relacionados:	Nenhum.

Manobras

Identificador:	Gerar Manobra
Descrição:	Um usuário pode, tomando como base um *snapshot*, gerar automaticamente as operações seccionadoras de uma manobra. Essa manobra pode ser referente à (i) liberação de um ou mais equipamentos da topologia; (ii) normalização de um ou mais equipamentos da topologia; (iii) transferência de um equipamento entre barras.
Interesses relacionados:	Nenhum.

Identificador:	Criar Manobra
Descrição:	Manualmente, a partir do unifilar gráfico da GUI, um usuário pode informar cada uma das ações seccionadoras para uma manobra. Essas ações podem estar, ou não, na ordem correta. A manobra, assim como no caso anterior pode ser referente à (i) liberação de um ou mais equipamentos da topologia; (ii) normalização de um ou mais equipamentos da topologia; (iii) transferência de um equipamento entre barras.
Interesses relacionados:	Nenhum.

Identificador:	Executar Manobra
Descrição:	Graficamente, as ações de seccionamento devem ser exibidas, no seu estado inicial e final, no unifilar da subestação correspondente, através da GUI.
Interesses relacionados:	Nenhum.

Identificador:	Editar Manobra
Descrição:	Uma manobra pode ser alterada pelo usuário, quer ela tenha sido gerada automaticamente ou não. Ações complementares podem ser inseridas, via GUI ou via texto.
Interesses relacionados:	Nenhum.

Identificador:	Salvar Manobra
Descrição:	Uma manobra, quer tenha sido alterada ou não, pode ser salva pelo usuário. Ao ser salva, a manobra, tipicamente um documento PGM (Programa de Manobra) é salvo no banco de dados do Sistema SisRTM.
Interesses relacionados:	Nenhum.

Propriedades

Identificador:	Visualizar Propriedades
Descrição:	Um usuário, dependendo das suas permissões, pode visualizar as propriedades existentes do sistema. As propriedades são, por exemplo, o endereço do Smart Model, endereço do SisRTM, endereço do Smart Unifilar, etc.
Interesses relacionados:	Nenhum.

Identificador:	Alterar Propriedade
Descrição:	Um usuário, dependendo das suas permissões, pode alterar uma propriedades do sistema em tempo de execução.
Interesses relacionados:	Nenhum.

Requisitos Não-funcionais

Identificador:	Tempo real
Atributos:	Interoperabilidade
Descrição:	Ao tirar *snapshots*, o sistema deve se comunicar com sistemas externos a fim de utilizar informações de tempo real da topologia da rede elétrica, bem como da disponibilidade atual dos equipamentos da rede.
Interesses relacionados:	Requisito funcional - Tirar *Snapshot*
Partes interessadas:	• Cliente; • Coordenador de projeto: possibilidade de efeitos financeiros; • Arquiteto: integrações com outros sistema levam a decisões envolvendo atributos como tolerância a falhas (por exemplo, qual será o comportamento esperado se o sistema externo estiver fora do ar?).

Identificador:	Aplicável a qualquer subestação
Atributos:	Manutenibilidade
Descrição:	O sistema deve ser genérico, aplicando-se a qualquer subestação, ou seja, não devem existir pedaços de código para tratar de arranjos específicos para em subestações.
Interesses relacionados:	Requisito funcional - Executar Manobra Requisito funcional - Gerar Manobra
Partes interessadas:	• Cliente; • Projetista (Camilla): responsável pela técnica para gerar manobras.

Identificador:	Suporte a mudanças na topologia
Atributos:	Manutenibilidade
Descrição:	Se houver alteração na topologia da rede, pouquíssimo ou nenhum esforço de manutenção deverá ser necessário.
Interesses relacionados:	Requisito não-funcional - Aplicável a qualquer subestação Requisito funcional - Executar Manobra
Partes interessadas:	• Cliente; • Projetista (Camilla): responsável pela técnica para gerar manobras.

Identificador:	Configuração manual mínima
Atributos:	Manutenibilidade
Descrição:	O sistema deve exigir o mínimo de configuração manual possível (para cadastrar intertravamentos especiais, por exemplo).
Interesses relacionados:	Nenhum.
Partes interessadas:	• Cliente; • Projetista (Camilla): responsável pela técnica para gerar manobras.

Identificador:	Interface intuitiva
Atributos:	Usabilidade
Descrição:	O usuário deve ser capaz de aprender funcionalidades básicas - por exemplo, gerar comandos para isolar um disjuntor - em menos de 15 minutos.
Interesses relacionados:	Nenhum.
Partes interessadas:	Cliente, cliente proxy, usuário: deverão realizar testes para atingir este requisito.

Identificador:	Usuários simultâneos
Atributos:	Desempenho
Descrição:	O sistema deve ter tempos de resposta de até 15 segundos para todas as funcionalidades considerando 10 usuários simultâneos e com o Smart Model, SisRTM e Smart Unifilar em funcionamento.
Interesses relacionados:	Nenhum.
Partes interessadas:	• Cliente; • Testadores: responsáveis pela preparação de testes de carga para garantir as condições impostas por esse requisito.

Identificador:	Milhares de equipamentos da topologia
Atributos:	Desempenho
Descrição:	O sistema deve ser capaz de carregar até 15000 equipamentos da topologia da rede elétrica em até 30 segundos e mantê-los carregados durante toda a execução do sistema.
Interesses relacionados:	Requisito funcional - Criar Manobra Requisito funcional - Gerar Manobra Requisito funcional - Editar Manobra Requisito funcional - Executar Manobra
Partes interessadas:	• Cliente; • Testadores: responsáveis pela preparação de testes de carga para garantir as condições impostas por esse requisito.

Identificador:	Sistema portável
Atributos:	Portabilidade
Descrição:	O sistema deve ser executável em diversos sistemas operacionais: Windows, Linux e AIX.
Interesses relacionados:	Nenhum.
Partes interessadas:	• Cliente; • Arquiteto e gerente de desenvolvimento: responsáveis pela escolha das tecnologias a serem empregadas.

Identificador:	Interface,
Atributos:	Tipo da interface
Descrição:	O sistema deve possuir interface *web*, permitindo assim seu uso com independência de localização. A interface do sistema, mesmo sendo *web*, deve ser semelhante a aplicações desktop: uso de janelas (redimensionáveis, minimizáveis e maximizáveis), caixas de diálogo, abas, etc.
Interesses relacionados:	Nenhum.
Partes interessadas:	• Cliente; • Gerente de desenvolvimento: responsável por guiar os desenvolvedores na preparação das telas do sistema.

Identificador:	Manobras elaboradas com unifilar
Atributos:	Usabilidade
Descrição:	O sistema deve permitir a elaboração de manobras através da manipulação gráfica de um unifilar, permitindo zoom e movimentação do mesmo.
Interesses relacionados:	Requisito funcional - Editar Manobra Requisito funcional - Alterar *Snapshot*
Partes interessadas:	• Cliente; • Gerente de desenvolvimento: responsável por guiar os desenvolvedores na preparação da tela de unifilares.

Identificador:	Baixo acoplamento
Atributos:	Extensibilidade Manutenibilidade
Descrição:	Deve-se primar pelo baixo acoplamento no sistema a fim de facilitar a manutenção, evitar mudanças em cascata, etc.
Interesses relacionados:	Nenhum.
Partes interessadas:	• Arquiteto e gerente de desenvolvimento: responsáveis pela escolha e validação das técnicas a serem empregadas para diminuir o acoplamento; • Desenvolvedor: responsável pela implementação das técnicas escolhidas.

Identificador:	Testes automáticos
Atributos:	Testabilidade
Descrição:	Todos os testes que existirem no sistema, uma vez iniciados, não devem requerer a intervenção humana.
Interesses relacionados:	Nenhum.
Partes interessadas:	• Arquiteto e gerente de desenvolvimento: responsáveis pela escolha das técnicas de teste a serem empregadas; • Gerente de desenvolvimento: responsável pelos testes de aceitação; • Desenvolvedor: responsável pelos testes de unidade e de carga; • Cliente: responsável pela validação dos testes.

Identificador:	Operações registradas
Atributos:	Auditabilidade Segurança
Descrição:	Todas as operações no sistema devem ser registradas, para facilitar a auditoria e a descoberta de possíveis bugs e falhas de software.
Interesses relacionados:	Nenhum.
Partes interessadas:	• Cliente; • Arquiteto e gerente de desenvolvimento: responsáveis pela escolha da biblioteca a ser utilizada e pelo que deve ser registrado.

Identificador:	Acesso restrito
Atributos:	Segurança
Descrição:	Todas as operações no sistema devem ser feitas somente por usuários autenticados.
Interesses relacionados:	Requisito funcional - Efetuar login
Partes interessadas:	• Cliente; • Arquiteto e gerente de desenvolvimento: responsáveis pela escolha da técnica de autenticação a ser utilizada.

Riscos

Identificador:	Uso dos unifilares em formato SVG
Descrição:	O Internet Explorer 8.0 e versões inferiores não trás suporte nativo ao formato SVG.
Interesses relacionados:	Requisito não-funcional - Manobras elaboradas a partir de manipulação gráfica do unifilar
Providências a serem tomadas:	• Ver com o cliente se é aceitável usar algum *browser* que seja compatível com o formato SVG; • Caso negativo, investigar outras tecnologias possíveis quando SVG não estiver disponivel e identificar limitações.
Partes interessadas:	Desenvolvedor, gerente de desenvolvimento, projetista e arquiteto.

Identificador:	Uso do sistema pelos operadores
Descrição:	Pelo fato de não ser um sistema de informação e de utilizar inteligência artificial, é difícil mostrar ao usuário que o sistema será benéfico e que poderá acelerar seu trabalho nas tarefas diárias.
Interesses relacionados:	Nenhum.
Providências a serem tomadas:	Realizar sessões de treinamento.
Partes interessadas:	Gerente de projeto e coordenador.

B.4 Pontos de Vista

O conteúdo desta seção, que pode ser lido em A.4, foi removido para evitar repetições no documento.

B.5 Visões

B.5.1 Visão Lógica

Nesta seção serão vistos termos e definições utilizados para esta visão, decisões-chave para a estrutura estática e comportamental do sistema, modelos em UML e a especificação dos elementos de software expressos nos modelos.

Termos e definições

SAGE O SAGE, Sistema Aberto de Gerenciamento de Energia, implementa as funções de gerenciamento de energia em centros de controle, suportado por uma arquitetura que contempla em toda a sua plenitude as características de sistemas abertos. Sua funcionalidade pode ser configurada para diversas aplicações no processo de automação das empresas, desde aplicações locais em usinas e subestações, com arquiteturas de baixo custo (PCs), até aplicações em centros de operação de grande porte suportadas por redes locais heterogêneas e hardware (*workstations*) de diferentes fabricantes [59].

Smart Model O Smart Model é um módulo responsável pelo gerenciamento em tempo real da topologia da rede elétrica, mantendo informações como estado dos equipamentos, grandezas elétricas, etc.

Smart Unifilar Módulo responsável pela conversão dos arquivos dos unifilares, em formato SigDraw, para formatos visualizáveis em navegadores *web*.

REST *REpresentational State Transfer* (REST) é um estilo de arquitetura de software para sistemas distribuidos como a *World Wide Web*. Arquiteturas no estilo REST consistem de clientes e servidores. Clientes iniciam requisições aos servidores; servidores as processam e retornam respostas apropriadas. Requisições e respostas são construídas em torno da transferência de representações de recursos. Um recurso pode ser qualquer conceito coerente e com algum significado que seja endereçavel. A representação de um recurso é tipicamente um documento que captura o estado corrente de um recurso.

Java Java é uma linguagem de programação orientada a objeto desenvolvida na década de 90 por uma equipe de programadores chefiada por James Gosling, na empresa *Sun Microsystems*. Diferentemente das linguagens convencionais, que são compiladas para código nativo, a linguagem Java é compilada para um *bytecode* que é executado por uma máquina virtual. A linguagem de programação Java é a linguagem convencional da Plataforma Java, mas não sua única linguagem.

RMI Java RMI, *Remote Method Invocation*, permite ao programador criar aplicações Java distribuídas, em que métodos de objetos Java remotos sejam acessíveis de outras máquinas virtuais Java, possivelmente de diferentes hosts [60].

GWT *Google Web Toolkit* (GWT) é um conjunto de ferramentas para desenvolver e otimizar aplicações *web* complexas.

XML XML, *eXtensible Markup Language*, é uma linguagem de marcadores desenhada para descrever dados de forma hierárquica.

JavaScript JavaScript é uma linguagem de programação criada pela *Netscape* em 1995 que, no lado cliente (o navegador do usuário), efetua comandos sem a necessidade de se processar no lado servidor (o servidor em que a aplicação está hospedada). Sua sintaxe é semelhante à do Java [61].

SVG *Scalable Vector Graphics* (SVG) são uma família de especificações de um formato baseado em XML para gráficos vetoriais em duas dimensões, tanto estáticos quanto dinâmicos.

Façade É um objeto que provê uma interface simplificada para uma grande quantidade de código.

Publish-subscribe *Publish–subscribe* é um padrão de troca de mensagens em que o transmissor de mensagens (*publisher*) não programa as mensagens para serem enviadas diretamente aos receptores (*subscribers*). Em vez disso, as mensagens publicadas são caracterizadas em classes, sem conhecimento do que os receptores são. De forma similar, os receptores expressam interesse em uma ou mais classes e só recebem mensagens daqueles interesses, sem nenhum conhecimento do que os transmissores são.

Apache Tomcat Servidor *web* que proporciona um ambiente para aplicações em Java serem executadas.

JAR Um arquivo no formato JAR, *Java ARchive*, é utilizado para distribuir classes Java, metadados e recursos como imagens, textos, etc.

WAR Um arquivo no formato WAR, *Web Application ARchive*, é um arquivo JAR utilizado para distribuir uma aplicação *web* feita em Java.

AJAX AJAX, *Asynchronous Javascript and XML*, é o uso metodológico de tecnologias como Javascript e XML, providas por navegadores, para tornar páginas *web* mais interativas com o usuário, utilizando-se de solicitações assíncronas de informações.

Java Persistence API Define um meio de mapeamento objeto-relacional para objetos Java simples e comuns, denominados beans de entidade.

SSH O protocolo *Secure Shell* permite a comunicação entre dois *hosts* de forma segura, garantindo a execução de comandos remotamente.

SFTP O protocolo *Secure File Transfer Protocol* permite a transferência de arquivos entre dois hosts de maneira segura.

Callback Uma *callback* é uma referência a um pedaço de código executável que é passado como argumento para outro código.

Decisões-chave

Esta seção trás uma tabela com o relacionamento entre interesses e as decisões-chave tomadas; em seguida, a explicação detalhada de cada decisão-chave citada.

Interesse	**Decisões-chave**
RF - Efetuar login	Utilizar REST para se comunicar com o SisRTM
RF - Tirar *Snapshot*	Utilizar REST para se comunicar com o Smart Model
RF - Alterar *Snapshot*	Manter *snapshots* em um banco de dados Utilizar uma implementação da *Java Persistence API* (JPA)
RF - Salvar *Snapshot*	Manter *snapshots* em um banco de dados Utilizar uma implementação da Java Persistence API (JPA)
RF - Buscar *Snapshot*	Manter *snapshots* em um banco de dados Utilizar uma implementação da *Java Persistence API* (JPA)

RF - Renomear *Snapshot*	Manter *snapshots* em um banco de dados Utilizar uma implementação da *Java Persistence API* (JPA)
RF - Gerar Manobra	Considerar intertravamento elétrico Gerar manobras com escopo de única subestação Utilizar biblioteca do Smart Model para carregar os equipamentos da topologia
RF - Criar Manobra	Utilizar topologia para as manobras
RF - Executar Manobra	Considerar intertravamento elétrico Utilizar topologia para as manobras Utilizar biblioteca do Smart Model para carregar os equipamentos da topologia
RF - Editar Manobra	Gerar manobras com escopo de única subestação Utilizar topologia para as manobras Utilizar REST para se comunicar com o SisRTM
RF - Salvar Manobra	Utilizar REST para se comunicar com o SisRTM Utilizar topologia para as manobras
RF - Importar Manobra	Utilizar REST para se comunicar com o SisRTM Utilizar topologia para as manobras Utilizar biblioteca do Smart Model para carregar os equipamentos da topologia Mostrar erro caso uma manobra a ser importada utilize um equipamento inexistente
RF - Visualizar Propriedades	Manter propriedades em arquivos

RF - Alterar Propriedade	Manter propriedades em arquivos
RNF - Tempo real	Utilizar o Smart Model para acessar informações em tempo real da topologia
RNF - Aplicável a qualquer subestação	Utilizar topologia para as manobras Realizar testes com diversos cenários e subestações
RNF - Suporte a mudanças na topologia	Utilizar o estado padrão de novos equipamentos na topologia
RNF - Configuração manual mínima	Utilizar regras de intertravamentos genéricas, independentes de configuração da topologia e/ou arranjos
RNF - Interface intuitiva	Utilizar mensagens detalhadas ao informar usuários Usar ícones e títulos em botões sempre que possível Informar como preencher caixas de entrada quando necessário Colocar botões de ajuda em telas mais complexas
RNF - Usuários simultâneos	Realizar testes de carga Não considerar uso simultâneo a partir de um centro
RNF - Milhares de equipamentos da topologia	Utilizar biblioteca do Smart Model para carregar os equipamentos da topologia
RNF - Sistema portável	Utilizar Java O sistema deve permitir implantação no servidor *Apache Tomcat*
RNF - Interface *web*	Utilizar *Google Web Toolkit* (GWT) O sistema deve permitir implantação no servidor *Apache Tomcat*

RNF - Manobras elaboradas com unifilar	Utilizar SVG para gerar os unifilares Utilizar o sistema Smart Unifilar para ler unifilares no formato SigDraw Descarregar periodicamente arquivos no formato SigDraw
RNF - Baixo acoplamento	Utilizar uma *façade* para acessar a lógica de negócio O sistema deve ser dividido em dois grandes módulos
RNF - Testes automáticos	Utilizar uma *façade* para acessar a lógica de negócio Usar o *EasyAccept* para testes de aceitação Realizar testes com diversos cenários e subestações
RNF - Operações registradas	Registrar qualquer acesso à *façade*
RNF - Acesso restrito	Utilizar REST para se comunicar com SisRTM

Seguem abaixo as decisões-chave de forma detalhada:

Decisão-chave:	Utilizar o Smart Model para acessar informações em tempo real da topologia
Interesses relacionados:	Requisito funcional - Tirar *Snapshot* Requisito não-funcional - Tempo real
Descrição:	O sistema Smart Model possui e provê acesso à topologia de equipamentos das subestações da Chesf. Com o seu uso é possível determinar estados de chaves, disjuntores, acessar medidas analógicas, etc.
Vantagens:	• Sistema já desenvolvido e em pleno funcionamento na Chesf; • Interfaces de acesso via REST e RMI; • Não há necessidade de criar novos módulos para acessar o SAGE.
Desvantagens:	Caso o Smart Model esteja fora do ar o acesso às informações em tempo real será interrompido.

Decisão-chave:	Utilizar o estado padrão de novos equipamentos na topologia
Interesses relacionados:	Requisito não-funcional - Suporte a mudanças na topologia
Descrição:	Caso existam novos equipamentos cujo estado não esteja disponível em uma *snapshot*, deve-se utilizar o estado padrão dos mesmos.
Vantagens:	Não há necessidade do usuário atualizar todas as manobras que sejam afetadas por novos equipamentos.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Utilizar topologia para as manobras
Interesses relacionados:	Requisito não-funcional - Aplicável a qualquer subestação Requisito funcional - Criar Manobra Requisito funcional - Editar Manobra Requisito funcional - Executar Manobra Requisito funcional - Gerar Manobra Requisito funcional - Salvar Manobra
Descrição:	Para realizar operações como gerar novas manobras é necessário se basear na topologia com o uso de algoritmos de travessia em grafos.
Vantagens:	• Desempenho dos algoritmos utilizados; • Sistema mais genérico.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Realizar testes com diversos cenários e subestações
Interesses relacionados:	Requisito não-funcional - Aplicável a qualquer subestação Requisito não-funcional - Testes automáticos
Descrição:	Testes executáveis automaticamente devem ser preparados contemplando diversos cenários em subestações distintas para maximizar o número de situações cobertas pelo sistema.
Vantagens:	• Sistema torna-se mais confiável; • Detectar *bugs*; • Medir desempenho.
Desvantagens:	Não há como testar todos os casos.

Decisão-chave:	Considerar intertravamento elétrico
Interesses relacionados:	Requisito funcional - Gerar Manobra Requisito funcional - Executar Manobra
Descrição:	O sistema deve considerar regras de intertravamento elétrico, com base na topologia da rede.
Vantagens:	Maior fidelidade com os cenários reais.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Gerar manobras com escopo de única subestação
Interesses relacionados:	Requisito funcional - Gerar Manobra Requisito funcional - Editar Manobra
Descrição:	As manobras geradas terão escopo de uma única subestação, com exceção de disjuntores de subestações vizinhas que poderão ser manipulados para desenergizar uma linha.
Vantagens:	Facilidade na implementação do algoritmo.
Desvantagens:	Podem existir cenários em que outras subestações devam ser consideradas.

Decisão-chave:	Realizar testes de carga
Interesses relacionados:	Requisito não-funcional - Usuários simultâneos
Descrição:	Testes de carga devem ser feitos para garantir que o sistema suporte as condições impostas pelo requisito não-funcional de interesse. Caso as condições não sejam satisfeitas, ferramentas de perfilamento para detectar gargalos deverão ser utilizadas.
Vantagens:	• Antecipar possíveis mudanças necessárias; • Garantir desempenho do sistema antes de entrar em produção.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Utilizar regras de intertravamentos genéricas, independentes de configuração da topologia e/ou arranjos
Interesses relacionados:	Requisito não-funcional - Configuração manual mínima
Descrição:	Quando há mudança na topologia, nenhuma regra deve ser alterada e/ou adicionada. Além disso, essas regras são transparentes para o usuário, ou seja, ele não precisa ter conhecimento da existência das mesmas e/ou ter que inseri-las para que o sistema funcione corretamente. Por exemplo, em intertravamentos elétricos de chaves, para sabermos quando se pode ou não manusear uma chave-seccionadora, utiliza-se o Circuito Equivalente de Thevenin.
Vantagens:	Menor número de intervenções do usuário.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Utilizar mensagens detalhadas ao informar usuários
Interesses relacionados:	Requisito não-funcional - Interface intuitiva
Descrição:	Fazer uso de mensagens como 'Erro ao salvar *snapshot*: Não foi possível se conectar ao Smart Model do CROL' em vez de 'Erro ao salvar'.
Vantagens:	• O usuário dispõe de mais informação para tomar providências; • Facilita a passagem de informações para os responsáveis pelo sistema.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Usar ícones e títulos em botões sempre que possível
Interesses relacionados:	Requisito não-funcional - Interface intuitiva
Descrição:	Há situações em que uma simples imagem consegue dar uma ideia do que acontecerá quando um botão é clicado, logo, nessas situações deverão ser utilizados ícones além do título - esse título é opcional.
Vantagens:	• Uso somente de ícones aumenta o espaço livre na interface; • Facilita o entendimento.
Desvantagens:	Às vezes é difícil encontrar um ícone bem explicativo.

Decisão-chave:	Informar como preencher caixas de entrada quando necessário
Interesses relacionados:	Requisito não-funcional - Interface intuitiva
Descrição:	Existem dados que devem ser digitados pelo usuário que estão sujeitos a restrições na hora da entrada, como o código de um equipamento, por exemplo. Nesses casos, deve-se colocar o link para um espaço com a explicação de como informar esses dados ou colocar na própria tela quando for possível.
Vantagens:	• Facilita o entendimento. • Evita erros do usuário.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Colocar botões de ajuda em telas mais complexas
Interesses relacionados:	Requisito não-funcional - Interface intuitiva
Descrição:	Telas como a de edição de manobra devem dispor de um botão de ajuda contendo explicações - com imagens e até videos, se possível - ao usuário para garantir o uso correto das funcionalidades.
Vantagens:	• Facilita o entendimento; • Evita erros do usuário.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Utilizar biblioteca do Smart Model para carregar os equipamentos da topologia
Interesses relacionados:	Requisito não-funcional - Milhares de equipamentos da topologia Requisito funcional - Gerar Manobra Requisito funcional - Executar Manobra Requisito funcional - Importar Manobra
Descrição:	A topologia será carregada na inicialização do sistema, para que não haja atraso ao gerar manobras, executá-las, etc.
Vantagens:	• Tempo economizado por fazer uso de um código já implementado; • Confiabilidade por utilizar uma biblioteca já consolidada; • Não há necessidade do Smart Model estar no ar para gerar e executar manobras.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Utilizar Java
Interesses relacionados:	Requisito não-funcional - Sistema portável
Descrição:	A linguagem de desenvolvimento principal será Java e toda a parte da lógica do sistema deve ser desenvolvida utilizando-a.
Vantagens:	• Linguagem bastante utilizada no mercado; • Mantida por uma grande organização; • Grande número de ferramentas e bibliotecas gratuitas existentes.
Desvantagens:	Por ser uma linguagem de programação que não é compilada diretamente em linguagem de máquina, pode ser mais lenta para certas operações.

Decisão-chave:	Utilizar *Google Web Toolkit* (GWT)
Interesses relacionados:	Requisito não-funcional - Interface *web*
Descrição:	GWT facilita a criação de aplicações *web* com suporte a ajax e não obriga conhecimento em JavaScript, pois o código é feito em Java e o compilador GWT o transforma em Javascript.
Vantagens:	• A interface *web* pode ser escrita em Java ou XML; • Não há necessidade de conhecer JavaScript; • Funciona em diversos navegadores *web*.
Desvantagens:	Pode não funcionar corretamente em dispositivos móveis.

Decisão-chave:	Utilizar SVG para gerar os unifilares
Interesses relacionados:	Requisito não-funcional - Manobras elaboradas com unifilar
Descrição:	O formato SVG será utilizado para exibir unifilares nos *browsers web* pois é um formato leve e permite animações.
Vantagens:	• Como é um formato vetorial, permite zoom e movimentação do gráfico sem que a qualidade seja perdida; • Formato compatível com a maioria dos *browsers* existentes.
Desvantagens:	• Internet Explorer 9 suporta um conjunto de características básico do SVG; • Internet Explorer 8 e inferiores não suportam, nativamente, o formato SVG.

Decisão-chave:	Utilizar REST para se comunicar com o SisRTM
Interesses relacionados:	Requisito funcional - Efetuar login Requisito funcional - Editar Manobra Requisito funcional - Salvar Manobra
Descrição:	O SisRTM já possui uma interface REST que contempla as funcionalidades de salvar manobras e efetuar login.
Vantagens:	Já existem bibliotecas prontas para acessar a interface REST do SisRTM.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Mostrar erro caso uma manobra a ser importada utilize um equipamento inexistente
Interesses relacionados:	Requisito funcional - Importar Manobra
Descrição:	O sistema deve monstrar uma mensagem de erro caso o usuário tente importar uma manobra e a *snapshot* associada não contenha um equipamento associado a alguma ação.
Vantagens:	Deixa o usuário mais informado sobre o que aconteceu.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Utilizar REST para se comunicar com o Smart Model
Interesses relacionados:	Requisito funcional - Tirar *Snapshot*
Descrição:	O Smart Model possui uma interface REST que permite recuperar o estado da topologia atual da rede elétrica.
Vantagens:	Já existem bibliotecas prontas para acessar a interface REST do Smart Model.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Utilizar o sistema Smart Unifilar para ler unifilares no formato SigDraw
Interesses relacionados:	Requisito não-funcional - Manobras elaboradas com unifilar
Descrição:	O sistema Smart Unifilar lê arquivos SigDraw e gera um conjunto de objetos Java. Uma biblioteca do Smart Unifilar será importada e a comunicação será feita utilizando a *façade* desse sistema.
Vantagens:	Separação de interesses.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Descarregar periodicamente arquivos no formato SigDraw
Interesses relacionados:	Requisito não-funcional - Manobras elaboradas com unifilar
Descrição:	Os unifilares devem ser descarregados periodicamente para que estejam sempre atualizados. Isto será feito utilizando SSH e SFTP.
Vantagens:	Simplicidade na implementação.
Desvantagens:	Dependendo do período escolhido, os unifilares podem ficar desatualizados por um tempo indesejável.

Decisão-chave:	Utilizar uma *façade* para acessar a lógica de negócio
Interesses relacionados:	Requisito não funcional - Baixo acoplamento Requisito não-funcional - Testes automáticos
Descrição:	A camada de lógica de negócio deve ser acessada através de uma *façade*, um objeto único. Os métodos da *façade* permitem acessar toda a lógica de negócio.
Vantagens:	• Baixo acoplamento; • Ocultação de informação; • Facilita a expressão de testes de aceitação.
Desvantagens:	A *façade* tende a ficar muito grande quando a lógica de negócio é extensa.

Decisão-chave:	O sistema deve ser dividido em dois grandes módulos
Interesses relacionados:	Requisito não-funcional - Baixo acoplamento
Descrição:	Os dois grandes módulos são: 1. Elementos de software relativos à interface gráfica; 2. Elementos de software relativos à lógica de negócio. O primeiro módulo deve conhecer o segundo, mas não o contrário. Caso seja necessária uma comunicação partindo do módulo da lógica de negócio para o modulo da interface gráfica, sugere-se o uso do padrão de projeto *publish-subscribe* ou semelhante.
Vantagens:	• Reduzir acoplamento; • Facilitar a criação de novas formas de apresentar os dados: interface voltada para dispositivos móveis, etc; • Facilita o trabalho da equipe em paralelo.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Registrar qualquer acesso à *façade*
Interesses relacionados:	Requisito não-funcional - Operações registradas
Descrição:	A *façade* deve registrar qualquer operação feita, incluindo falhas. O registro deve conter o horário, o usuário autenticado e descrição da operação.
Vantagens:	• Possibilita auditoria. • Facilita a descoberta de erros.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	O sistema deve permitir implantação no servidor *Apache Tomcat*
Interesses relacionados:	Requisito não-funcional - Interface *web* Requisito não-funcional - Sistema portável
Descrição:	O SmartSwitch deve possibilitar o empacotamento em um arquivo WAR para ser implantado no *Tomcat*.
Vantagens:	• Apache Tomcat é gratuito; • Utilizado mundialmente.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Usar o *EasyAccept* para testes de aceitação
Interesses relacionados:	Requisito não-funcional - Testes automáticos
Descrição:	Uma ferramenta especializada para expressar testes de aceitação e que possa executar todos os testes de lógica de negócio automaticamente deve ser utilizada.
Vantagens:	Facilidade de realizar os testes
Desvantagens:	Provável necessidade de gerar mais uma *façade* a fim de facilitar e possibilitar os testes de aceitação.

Decisão-chave:	Manter *snapshots* em um banco de dados
Interesses relacionados:	Requisito funcional - Salvar *Snapshot* Requisito funcional - Buscar *Snapshot* Requisito funcional - Excluir *Snapshot* Requisito funcional - Renomear *Snapshot* Requisito funcional - Alterar *Snapshot*
Descrição:	Um sistema de gerenciamento de banco de dados (SGBD) será utilizado para que todas as *snapshots* sejam gravadas em disco. Para facilitar a manutenção pode ser interessante utilizar um banco de dados embarcado.
Vantagens:	Segurança, praticidade, organização e rapidez para gerenciar os dados.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Manter propriedades em arquivos
Interesses relacionados:	Requisito funcional - Alterar Propriedade Requisito funcional - Visualizar Propriedades
Descrição:	Um arquivo de propriedades será utilizado para guardar informações como endereço dos servidores Smart, endereço do SisRTM, etc. Além disso, existirá um XML necessário para configurar a *Java Persistence API*.
Vantagens:	Facilidade na mudança de propriedades.
Desvantagens:	Informações em diferentes locais dificulta o *backup*.

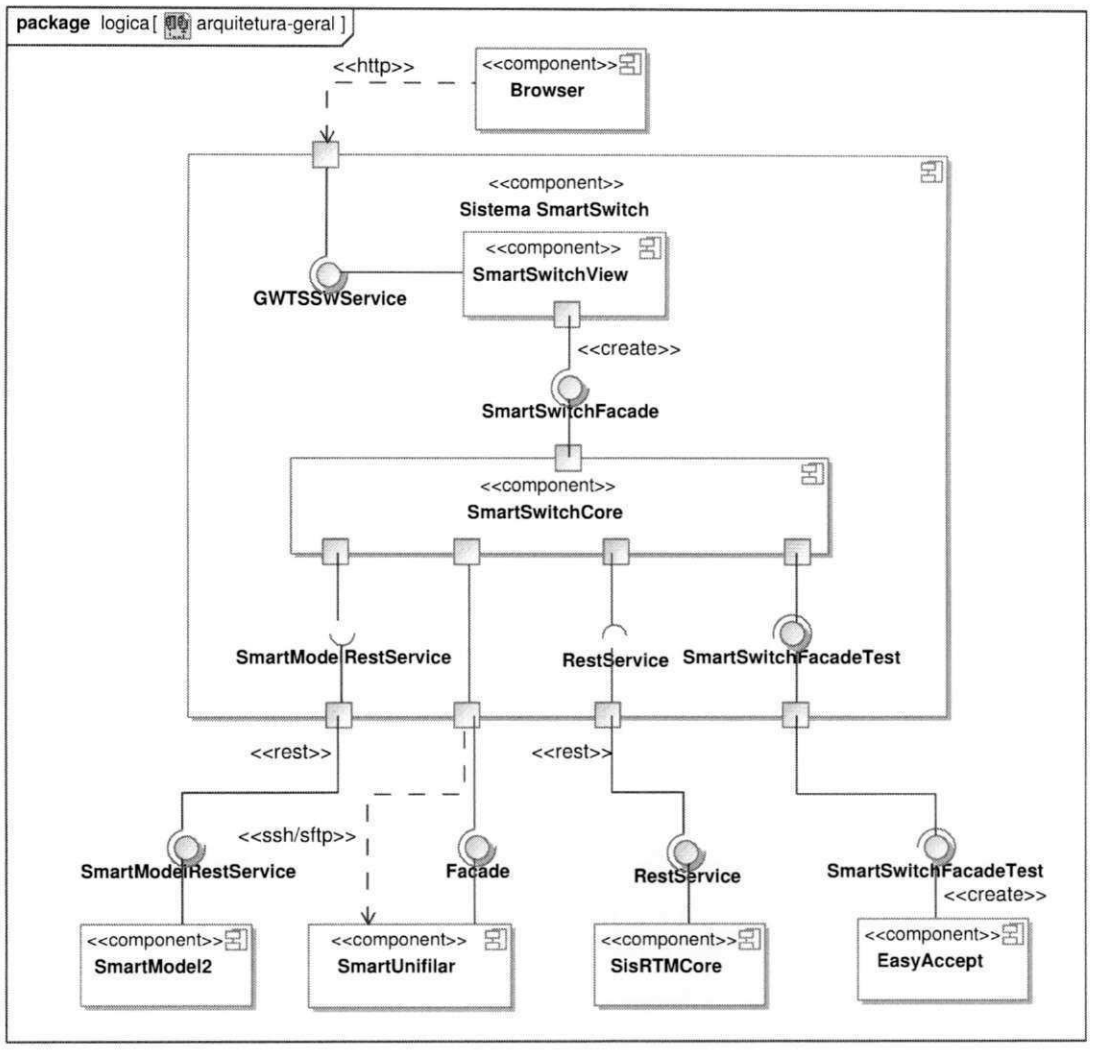
Decisão-chave:	Utilizar uma implementação da *Java Persistence API* (JPA)
Interesses relacionados:	Requisito funcional - Salvar *Snapshot* Requisito funcional - Buscar *Snapshot* Requisito funcional - Excluir *Snapshot* Requisito funcional - Renomear *Snapshot* Requisito funcional - Alterar *Snapshot*
Descrição:	O uso da *Java Persistence API* facilitará o mapeamento entre objetos e tabelas do banco.
Vantagens:	• Não há necessidade de grande conhecimento de SQL; • Existem várias implementações da JPA.
Desvantagens:	Pequena perda de desempenho.

Modelos

Arquitetura Geral

- **`SmartSwitchCore`**: componente responsável por toda a lógica de negócio do sistema. Mais detalhes podem ser vistos na seção B.5.1;
- **`SmartSwitchView`**: componente responsável por toda a lógica relativa à interface *web* do sistema. Mais detalhes podem ser vistos na seção B.5.1;
- O acesso às interfaces `SmartModelRestService` e `RestService` será feito utilizando REST, então, todas as chamadas realizadas a esses serviços resultarão em chamadas HTTP;
- Para simplificar o diagrama, o servidor *Tomcat* que será utilizado para hospedar o sistema SmartSwitch não foi inserido;
- Os acessos à interface `GWTSSWService` serão feitos a partir do *browser* utilizando AJAX e resultarão em chamadas em HTTP. Essa comunicação será de forma assíncrona;

Figura B.1: Arquitetura Geral do SmartSwitch

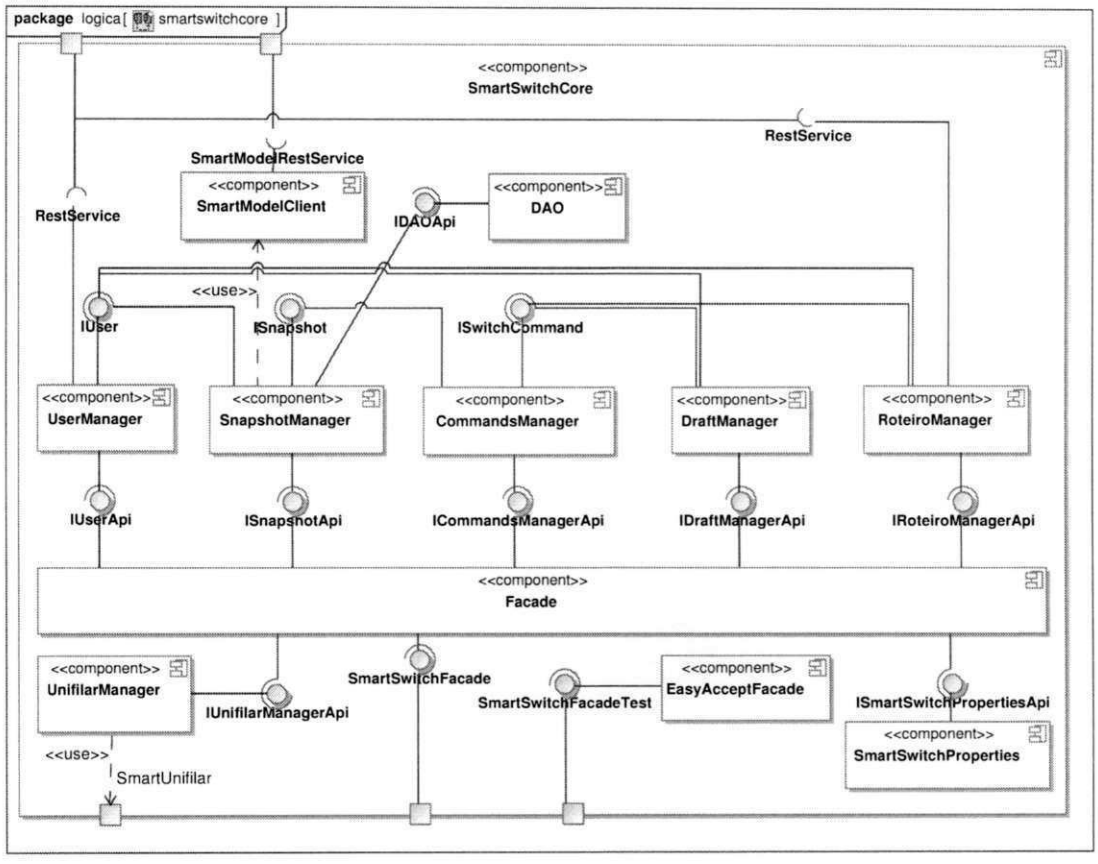


- O acesso ao `SmartSwitchCore` a partir do `SmartSwitchView` não causará nenhuma chamada HTTP, RMI ou equivalente;
- Os acessos ao Smart Unifilar podem ser via *façade*, não causando nenhuma chamada HTTP, RMI ou equivalente; via protocolo SSH, para execução de comandos e via protocolo SFTP, para a transferência dos unifilares.

Componente `SmartSwitchCore`

- **`UserManager`**: responsável pela autenticação dos usuários através de consultas ao

Figura B.2: Modelo do componente `SmartSwitchCore`



SisRTM;

- **`SnapshotManager`**: responsável por incluir, alterar e remover um *snapshot*. Também se comunica com o Smart Model a fim de obter o estado atual da topologia e gerar um *snapshot*;
- **`DAO`**: responsável por gravar, recuperar, atualizar e remover dados;
- **`CommandsManager`**: responsável por gerar um conjunto de comandos que representam a manobra;
- **`DraftManager`**: responsável por gerenciar o "rascunho" do PGM gerado automaticamente. Fora chamado de rascunho porque alterações podem ser feitas nesse PGM antes de ser exportado para o SisRTM. Essas alterações incluem mudança da ordem

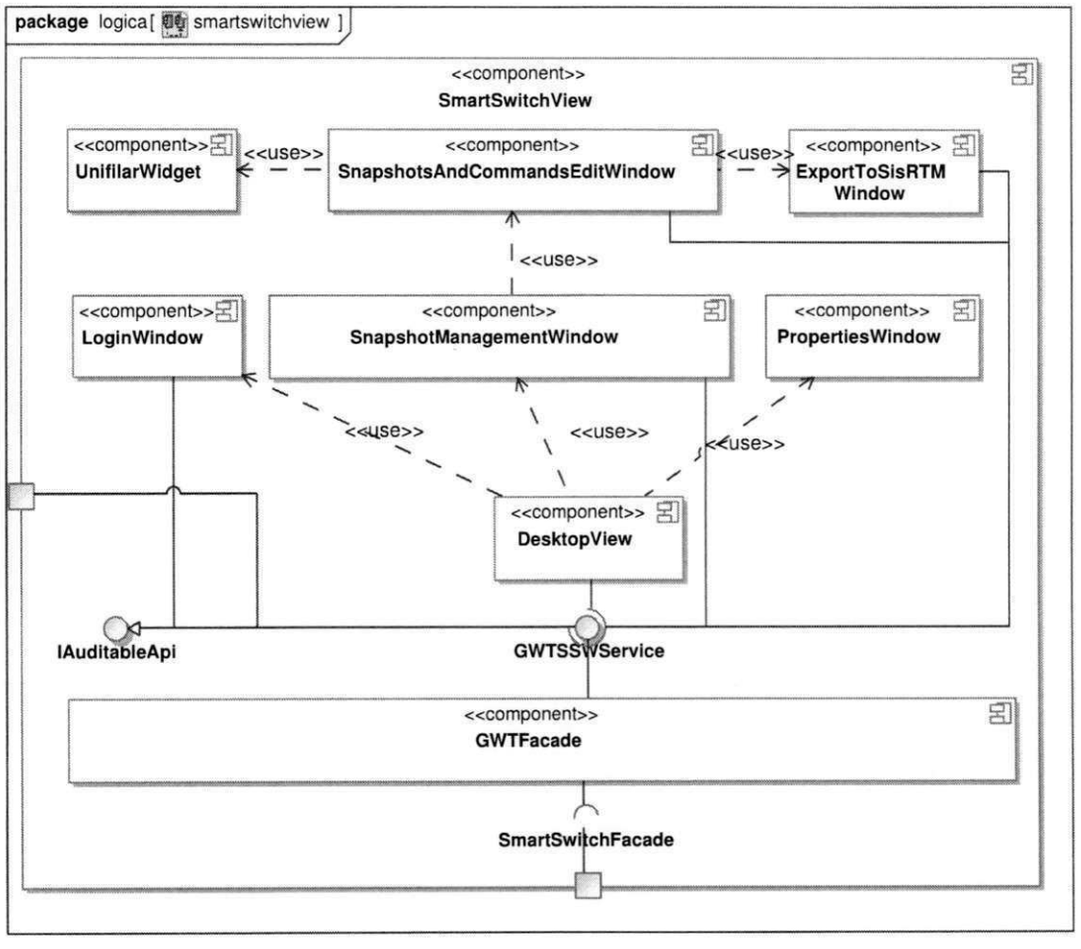
de comandos, inserir ou remover comandos, entre outros;

- **`RoteiroManager`**: responsável por converter uma lista de **`ISwitchCommand`** em um roteiro e vice-versa, além de exportar/importar um documento para o SisRTM;
- **`Facade`**: responsável pela *façade* que dá acesso à lógica de negócio do sistema;
- **`UnifilarManager`**: responsável pela comunicação com o sistema Smart Unifilar;
- **`SmartSwitchProperties`**: responsável por recuperar e armazenar propriedades, como endereço do SisRTM, endereço do Smart Model, etc;
- **`EasyAcceptFacade`**: responsável pela implentação de uma *façade* que será acessada pelo *EasyAccept*.

Componente `SmartSwitchView`

- **`GWTFacade`**: componente responsável por implementar a *façade* que será acessada assincronamente pelos componentes relativos à interface gráfica;
- **`IAuditableApi`**: responsável pelo registro de todas as operações realizadas;
- **`DesktopView`**: componente responsável pela tela inicial do sistema;
- **`LoginWindow`**: responsável por implementar a tela de login;
- **`PropertiesWindow`**: responsável por implementar a tela de propriedades, possibilitando a edição das mesmas;
- **`SnapshotManagementWindow`**: responsável por implementar a tela de gerenciamento, possibilitando gerar, buscar, renomear e remover *snapshots*;
- **`SnapshotsAndCommandsEditWindow`**: responsável pela edição de *snapshots*, permitindo abrir/fechar seccionadores, impedir equipamentos e pela edição de manobras, permitindo alterar, executar e exportar manobras;
- **`ExportToSisRTMWindow`**: responsável pela exportação de manobras;
- **`UnifilarWidget`**: responsável pela exibição, zoom, movimentação, etc. de unifilares.

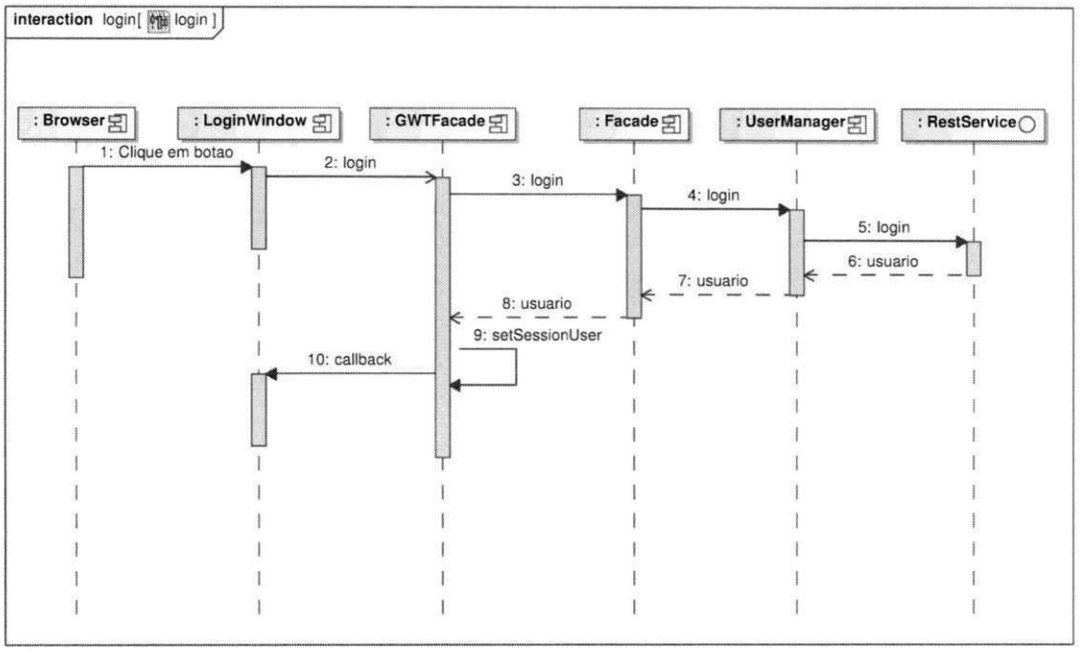
Figura B.3: Modelo do componente `SmartSwitchView`



Cenário: Login Para ter acesso ao sistema, o usuário precisa informar seus dados de acesso. Eis as etapas que ocorrem quando o botão para login é pressionado:

- Uma chamada assíncrona a partir do *browser* do cliente, utilizando AJAX, é feita à *façade* GWT;
- A *façade* GWT do `SmartSwitchView` acessa a *façade* do `SmartSwitchCore`;
- O serviço REST do SisRTM é consultado com o uso dos dados de acesso informados pelo usuário;
- Se o usuário for válido, o mesmo é salvo na sessão da *façade* GWT;

Figura B.4: Diagrama de sequência para o cenário "Login"

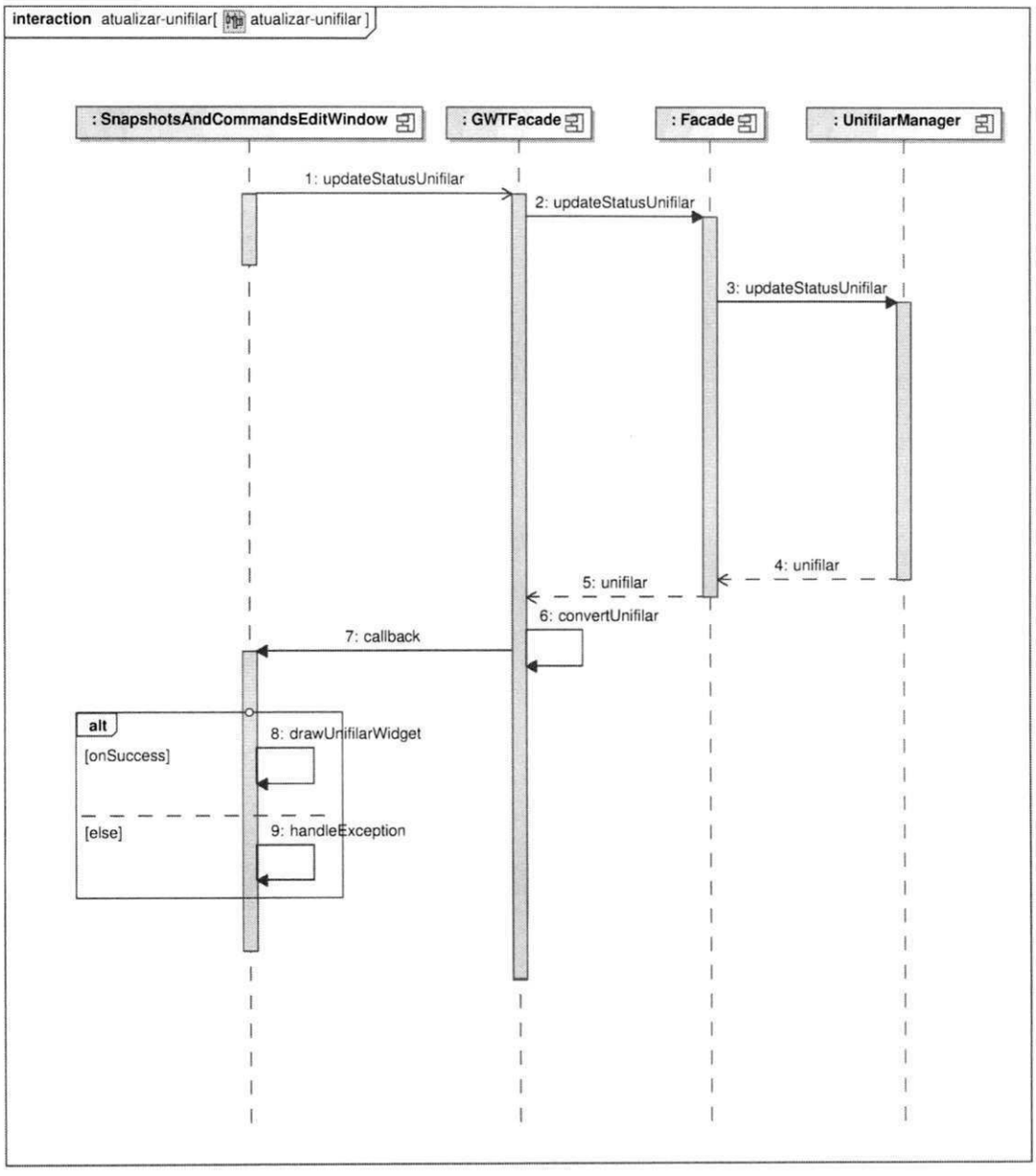


- A *façade* GWT faz uma chamada à callback localizada na tela de login no *browser* do cliente. A *callback* receberá informações como: dados do usuário, caso os dados de acesso estejam corretos, ou a exceção que ocorreu, caso os dados de acesso estejam incorretos, ocorreram problemas na comunicação com o SisRTM, etc.

Cenário: Atualização de unifilares em tempo real Ao abrir a tela de geração de *snapshots*, deve-se escolher uma subestação a ser visualizada. Após a carga do unifilar, as seguintes etapas ocorrem a cada cinco segundos:

- Uma chamada assíncrona a partir do *browser* do cliente, utilizando AJAX, é feita à *façade* GWT;
- A *façade* GWT do `SmartSwitchView` acessa a *façade* do `SmartSwitchCore`;
- O `UnifilarManager` é acessado para que o unifilar seja atualizado, ou seja, os estados de disjuntores, chaves, etc. é sincronizado com o Smart Model.;
- Este unifilar é convertido em um conjunto de objetos GWT;

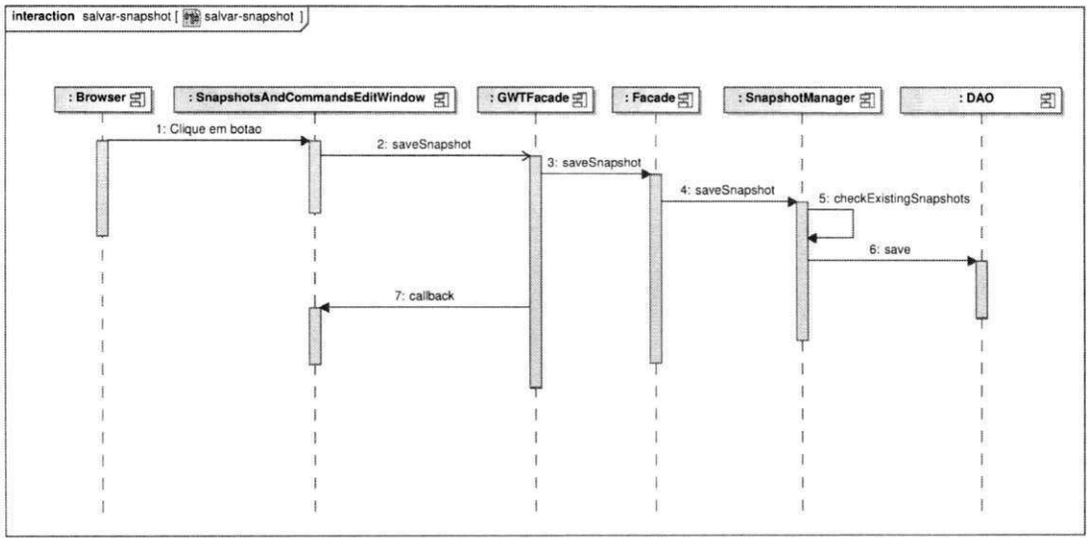
Figura B.5: Diagrama de sequência para o cenário "Atualização de unifilares em tempo real"



- A *façade* GWT faz uma chamada à *callback* localizada na tela de gerenciamento de *snapshots* no *browser* do cliente. A *callback* receberá a versão atualizada do unifilar ou a exceção que ocorreu, no caso de problemas na comunicação com o Smart Model, etc;

- No caso de sucesso na atualização do unifilar, ele é redesenhado na tela.

Figura B.6: Diagrama de sequência para o cenário "Salvar *Snapshot*"

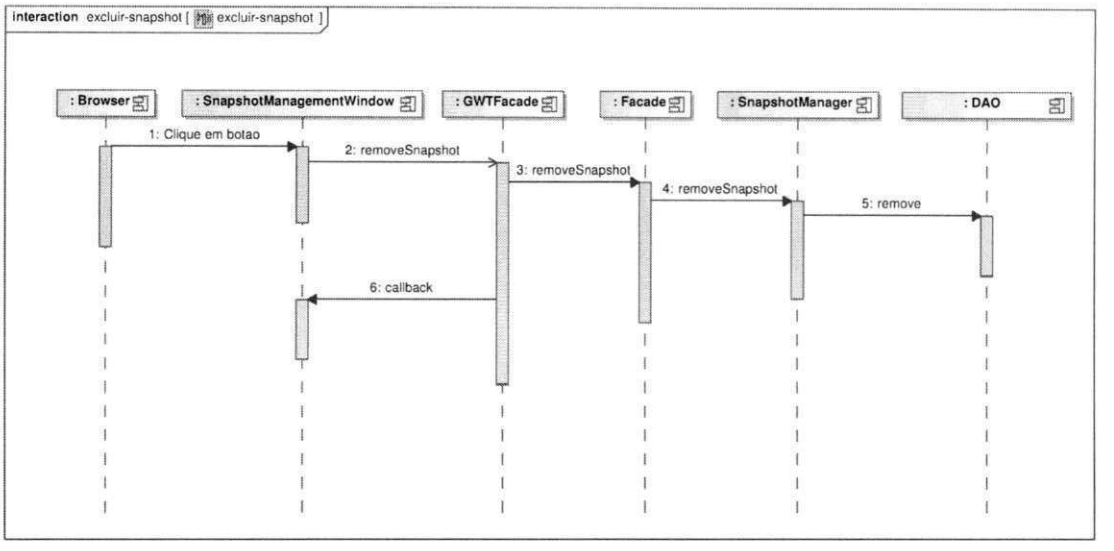


Cenário: Salvar *Snapshot* Para salvar um *snapshot* o usuário deve acessar a tela de edição de *snapshots* e manobras. Eis as etapas que ocorrem quando o botão para salvar é pressionado:

- Uma chamada assíncrona a partir do *browser* do cliente, utilizando AJAX, é feita à *façade* GWT;
- A *façade* GWT do `SmartSwitchView` acessa a *façade* do `SmartSwitchCore`;
- O componente de persistência é acessado com a passagem da *snapshot* a ser salva;
- A *façade* GWT faz uma chamada à *callback* localizada na tela de edição de *snapshots* no *browser* do cliente. A *callback* receberá informações como: exceção que ocorreu, no caso de problemas na comunicação com o banco de dados, etc.

Cenário: Excluir *Snapshot* Para excluir um *snapshot* o usuário deve acessar a tela de busca de *snapshots*. Eis as etapas que ocorrem quando o botão para excluir é pressionado:

Figura B.7: Diagrama de sequência para o cenário "Excluir *Snapshot*"

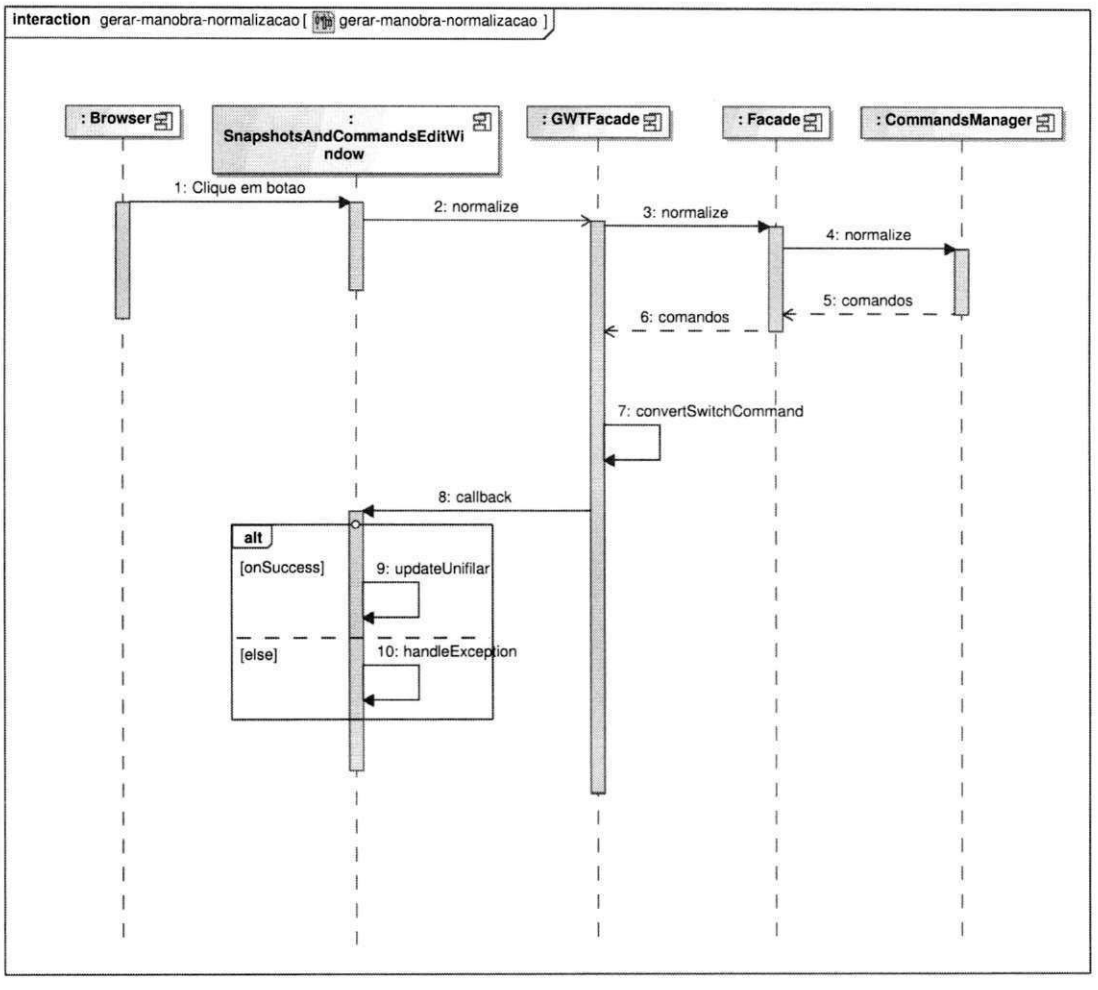


- Uma chamada assíncrona a partir do *browser* do cliente, utilizando AJAX, é feita à *façade* GWT;
- A *façade* GWT do `SmartSwitchView` acessa a *façade* do `SmartSwitchCore`;
- O componente de persistência é acessado com a passagem da *snapshot* a ser excluída;
- A *façade* GWT faz uma chamada à *callback* localizada na tela de gerenciamento de *snapshots* no *browser* do cliente. A *callback* receberá informações como: exceção que ocorreu, no caso de problemas na comunicação com o banco de dados, etc.

Cenário: Gerar Manobra - Normalizar Para gerar uma manobra de normalização - outros tipos de manobra são de forma equivalente - o usuário deve acessar a tela de edição de *snapshots* e manobras. Eis as etapas que ocorrem quando o botão para geração é pressionado:

- Uma chamada assíncrona a partir do *browser* do cliente, utilizando AJAX, é feita à *façade* GWT;
- A *façade GWT* do `SmartSwitchView` acessa a *façade* do `SmartSwitchCore`;
- O gerenciador de manobras é chamado e as manobras são geradas;

Figura B.8: Diagrama de sequência para o cenário "Gerar Manobra - Normalizar"

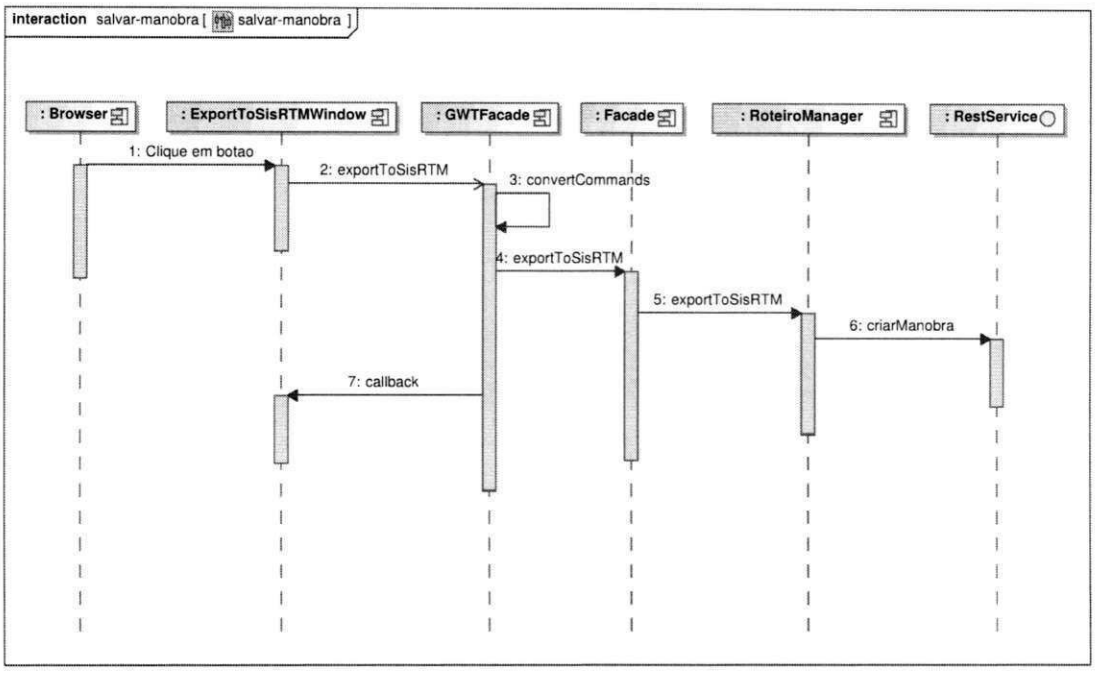


- A *callback* receberá informações como: lista de manobras ou a exceção que ocorreu, no caso de problemas na comunicação com o SisRTM, etc;
- No caso de sucesso na geração da manobra, a tela do unifilar deve ser atualizada.

Cenário: Salvar Manobra Para salvar uma manobra o usuário deve acessar a tela de edição de *snapshots* e manobras. Eis as etapas que ocorrem quando o botão para salvar é pressionado:

- Uma chamada assíncrona a partir do *browser* do cliente, utilizando AJAX, é feita à *façade* GWT;

Figura B.9: Diagrama de sequência para o cenário "Salvar Manobra"

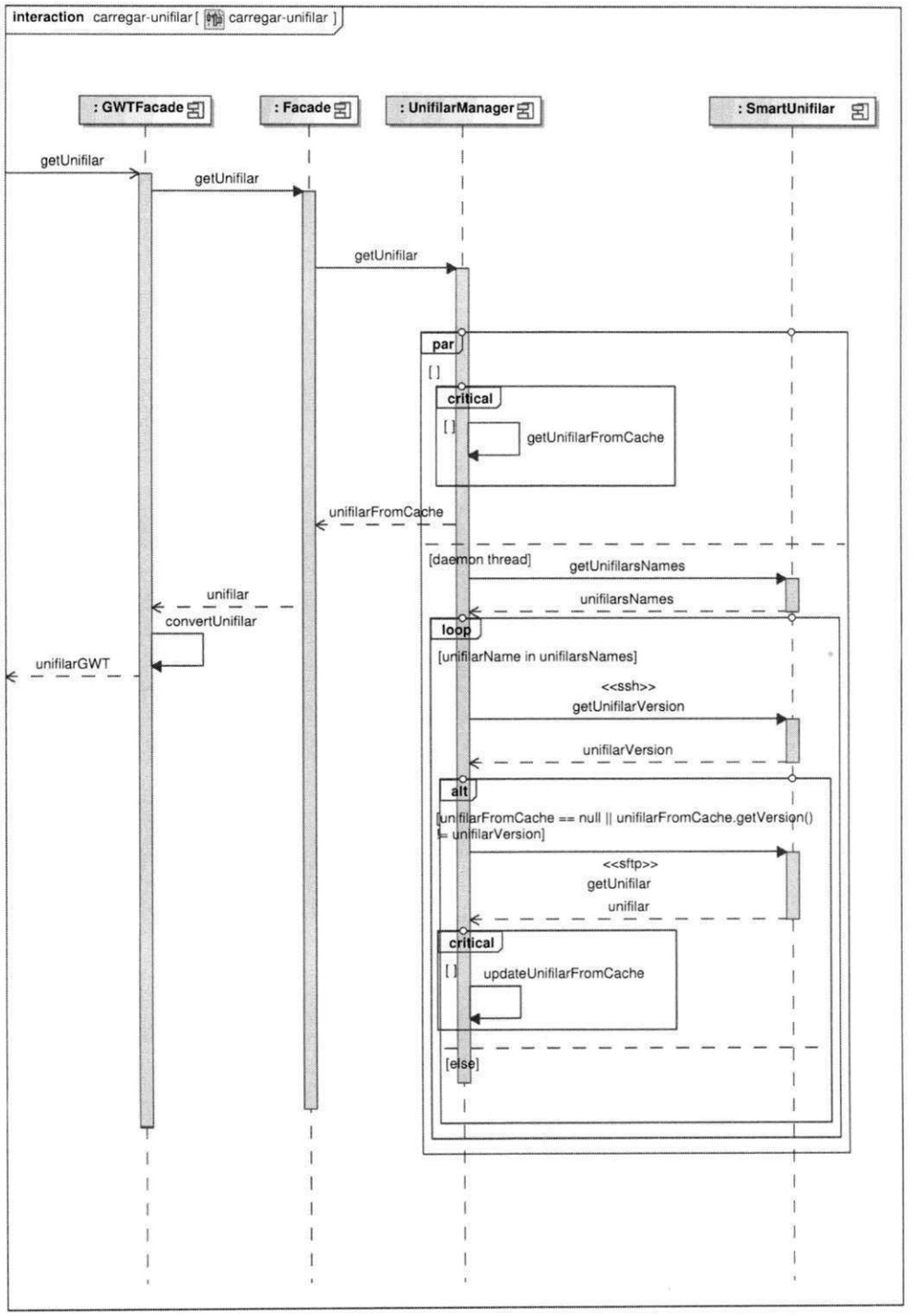


- A *façade* GWT do `SmartSwitchView` acessa a *façade* do `SmartSwitchCore`;
- A *façade* do `SmartSwitchCore` acessa o `RoteiroManager`, responsável por convertê-las em um roteiro e exportá-lo para o SisRTM;
- A *façade* GWT faz uma chamada à *callback* localizada na tela de edição de exportação de manobras no *browser* do cliente. A *callback* receberá informações como: exceção que ocorreu, no caso de problemas na comunicação com o SisRTM, etc.

Cenário: Carregar unifilar Um unifilar será exibido na edição de um *snapshot* ou de uma manobra, então, é necessário carregá-lo antes de exibir a tela de edição. Eis as etapas que ocorrem:

- Uma chamada é feita à *façade* GWT;
- A *façade* GWT do `SmartSwitchView` acessa a *façade* do `SmartSwitchCore`;
- A unifilar existente em *cache* é retornada;

Figura B.10: Diagrama de sequência para o cenário "Carregar unifilar"



- O unifilar é convertido em objetos GWT e é retornado.

Em paralelo, existe um *thread* que verifica periodicamente se novos unifilares estão disponíveis:

- O Smart Unifilar é acessado utilizando SSH e os nomes dos unifilares são retornados.
- Para cada unifilar:
  - O Smart Unifilar é acessado utilizando SSH e o número da versão do unifilar é retornado.
  - Se não existir um unifilar em cache ou a versão existente é desatualizada, o novo unifilar é descarregado utilizando SFTP;
  - Para atualizar a unifilar em *cache*, é necessário que sejam utilizados mecanismos de sincronização pois podem existir outros *threads* fazendo requisições às unifilares em *cache*.

Interfaces

Segue abaixo a lista de todas as interfaces a serem implementadas pelo sistema.

Nome:	`IUser`
Descrição:	Representa um usuário do sistema.
Serviços providos:	`getLogin`: recupera o login do usuário; `getSenha`: recupera a senha encriptada do usuário; `getNomeCompleto`: recupera o nome completo do usuário; `getPapeis`: recupera os papéis do usuário; `getOrigens`: recupera as origens do usuário.

Nome:	`IUserApi`
Descrição:	Provê a autenticação de usuários.
Serviços providos:	`login`: recebe login e senha e retorna o usuário associado.

Nome:	`ISnapshot`
Descrição:	Representa um *snapshot*.
Serviços providos:	`getDate`: retorna a data de criação do *snapshot*; `getName`: retorna o nome do *snapshot*; `getSnapshotElements`: retorna uma coleção de `SnapshotEmement`. Um `SnapshotElement` é composto pelo nome de um equipamento e suas propriedades, como aberto ou não, etc; `getCurrentSubstation`: retorna a subestação representada por esse *snapshot*. Pode ser nulo caso seja um *snapshot* da topologia inteira; `isComplete`: retorna `true` caso o *snapshot* represente a topologia inteira.

Nome:	`ISnapshotApi`
Descrição:	Provê várias operações para o gerenciamento de *snapshots*.
Serviços providos:	`getSnapshot`: retorna um *snapshot* salva no SGBD; `removeSnapshot`: remove um *snapshot* do SGBD; `createSnapshot`: gera um *snapshot* em tempo real de toda a topologia; `createSubstationSnapshot`: gera um *snapshot* em tempo real de uma subestação; `findSnapshots`: retorna os *snapshots* encontrados de acordo com os critérios de busca informados.

Nome:	`IDAOApi`
Descrição:	Provê várias operações para a persistência de dados.
Serviços providos:	`save`: Salva um objeto; `load`: Carrega um objeto a partir de um identificador; `remove`: Remove um objeto a partir de um identificador; `search`: Busca objetos a partir de parâmetros de busca.

Nome:	`ISwitchCommand`
Descrição:	Representa uma manobra realizada a um seccionador.
Serviços providos:	`getSwitchOperation`: retorna o tipo da manobra; `getEquipment`: retorna o equipamento afetado.

Nome:	`ISmartSwitchPropertiesApi`
Descrição:	Provê as operações de gerenciamento de propriedades.
Serviços providos:	`getProperty`: retorna o valor de uma propriedade; `getSisRTMUrl`: retorna a URL de acesso ao SisRTM; `getRegionals`: retorna as regionais acessíveis pelo SmartSwitch; `isLoggingEnabled`: retorna `true` caso os registros das operações estão habilitados, `false` caso contrário; `getSmartModelUrl`: recebe uma regional e retorna a URL de acesso ao serviço REST de um Smart Model; `loadProperties`: carrega as propriedades.

Nome:	`ICommandManagerApi`
Descrição:	Provê as operações de gerenciamento de manobras.
Serviços providos:	`release`: gera um conjunto de ações para liberar disjuntores, chaves, linhas de transmissão, barramentos, transformadores, reatores ou bancos de capacitores; `normalize`: gera um conjunto de ações para normalizar disjuntores, chaves, linhas de transmissão, barramentos, transformadores, reatores ou bancos de capacitores; `transferEquipment`: gera um conjunto de ações para transferir um equipamento; `isolate`: gera um conjunto de ações para isolar um disjuntor; `canOpen`: verifica se um seccionador pode ser aberto; `canClose`: verifica se um seccionador pode ser fechado.

Nome:	`IDraftManagerApi`
Descrição:	Provê as operações para edição de manobras.
Serviços providos:	`addAction`: adiciona uma ação; `removeAction`: remove uma ação; `getActions`: retorna a lista de ações geradas; `clearActions`: limpa a lista de ações.

Nome:	`IRoteiroManagerApi`
Descrição:	Provê as operações de exportar roteiros, dentre outras.
Serviços providos:	`exportToSisRTM`: exporta uma lista de ações para o SisRTM, gerando um PGM.

Nome:	`IAuditableApi`
Descrição:	Provê as operações para registro de operações no sistema.
Serviços providos:	`logResponseDetails`: Registra as operações de acesso ao sistema, incluindo usuário, IP da máquina, operação realizada, duração da operação e memória em uso pela máquina virtual Java.

Nome:	`IUnifilarManagerApi`
Descrição:	Provê as operações de acesso aos unifilares.
Serviços providos:	`getUnifilar`: retorna um objeto que representa um unifilar a partir da leitura de um arquivo SigDraw; `checkUnifilarsFromCache`: verifica se há mudanças nos unifilares existentes a partir de acessos SSH/SFTP ao Smart Unifilar. Mais detalhes em B.5.1.

Nome:	`SmartSwitchFacade`
Descrição:	Representa a *façade* de acesso ao sistema.
Serviços providos:	Junção dos serviços providos por `IUserApi`, `ISnapshotApi`, `ICommandManagerApi`, `IDraftManagerApi`, `IRoteiroManagerApi`, `IAuditableApi` e `IUnifilarManagerApi`.

Nome:	`SmartSwitchFacadeTest`
Descrição:	Representa a *façade* que será acessada para os testes de aceitação.
Serviços providos:	Todos os serviços providos por `SmartSwitchCoreFacade` com valores de retorno e parâmetros alterados para se adaptar à solução de testes de aceitação.

B.5.2 Visão de Desenvolvimento

Nesta seção serão apresentados diagramas com os pacotes a serem utilizados pelos desenvolvedores.

Modelos

Pacotes para o `SmartSwitchCore`

- O `SmartSwitchCore` deve ser empacotado em um arquivo de formato JAR.

Pacotes para o `SmartSwitchView`

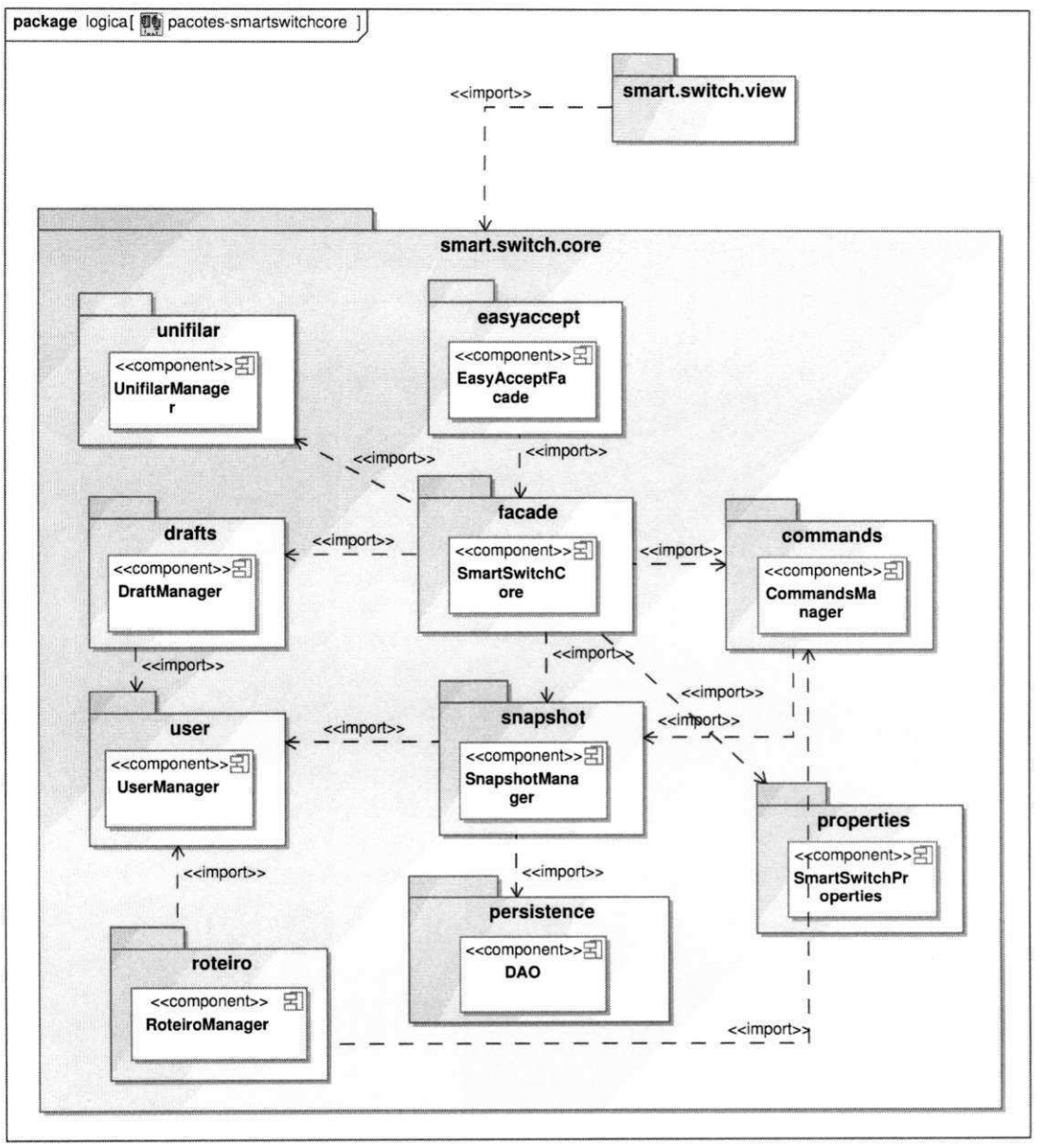
- O `SmartSwitchView` deve ser empacotado em um arquivo de formato WAR.

B.5.3 Visão de Processos

Não é necessário entrar em muitos detalhes nesta visão pois não há requisitos relativos a escalabilidade, concorrência, etc. Serão listados abaixo detalhes não óbvios:

- Cada requisição feita à *façade* `GWTFacade` a partir do *browser* via chamadas AJAX causa a utilização de um novo *thread*;
- Existe um *thread* sempre em execução responsável pela checagem da atualização de novos unifilares. Mais detalhes em B.5.1;
- Não é possível que dois usuários alterem a mesma manobra simultaneamente;
- O SmartSwitch roda em um único processo, ou seja, a lógica de negócio e a lógica relativa à interface gráfica são executadas juntamente. Não é necessária a separação dos processos devido à pequena escala do sistema;

Figura B.11: Diagrama de pacotes do `SmartSwitchCore`



- Os testes de aceitação da lógica de negócio são executados em um processo à parte, sem necessidade de haver um servidor de aplicação.

Figura B.12: Diagrama de pacotes do `SmartSwitchView`

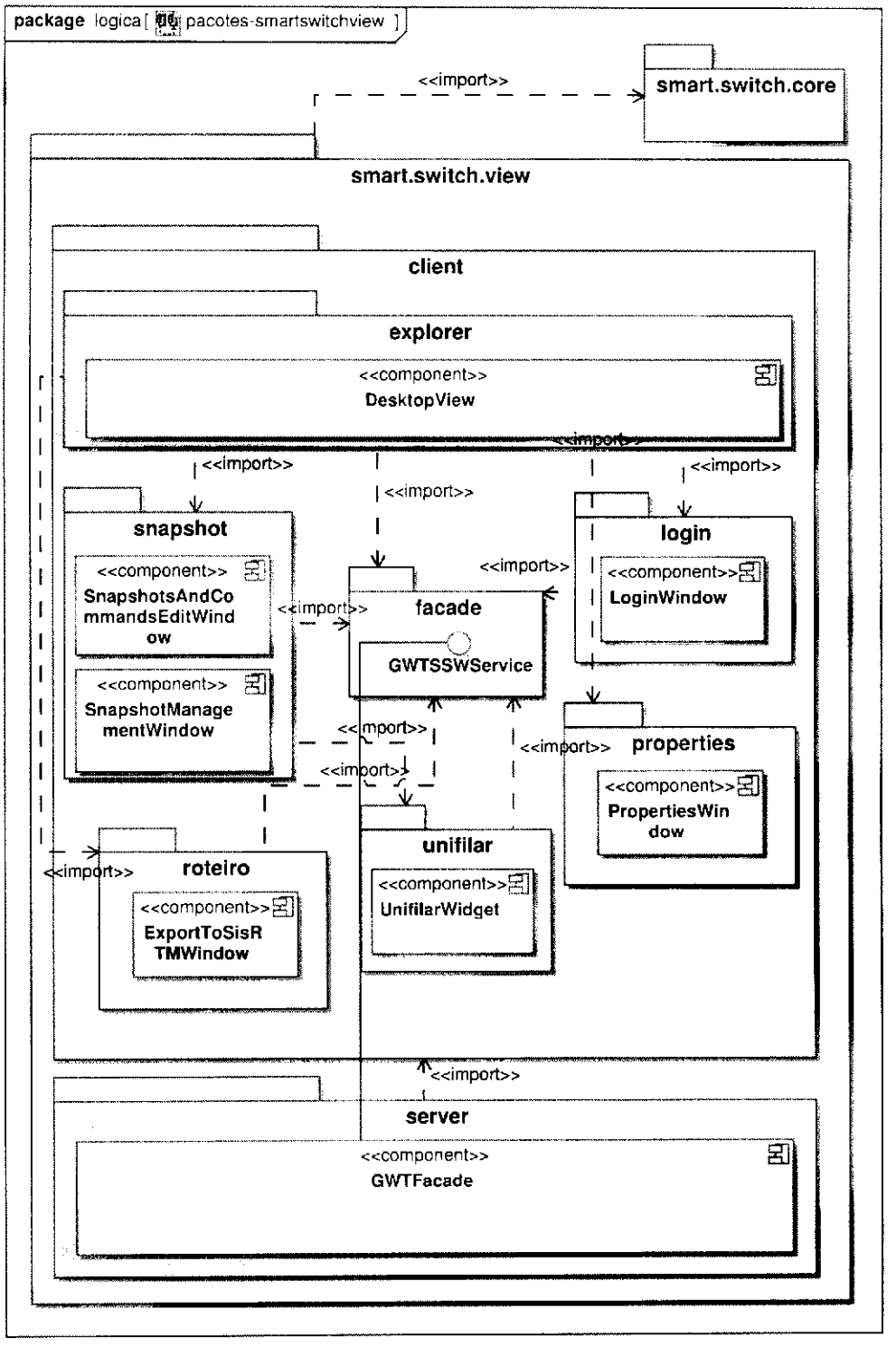
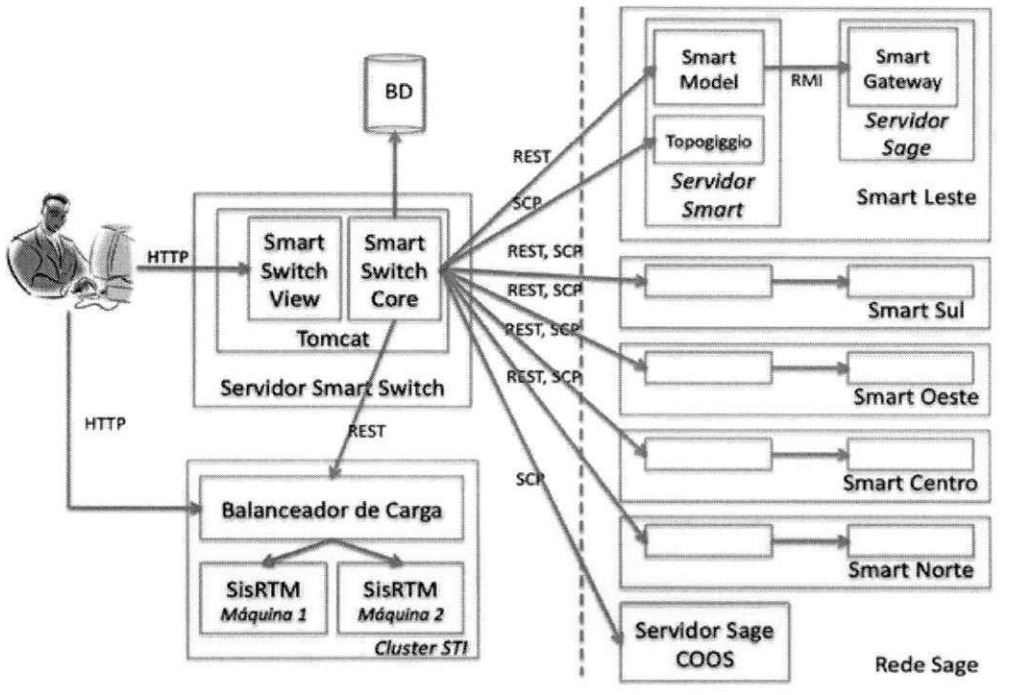


Figura B.13: Esquema resumido da visão física



B.5.4 Visão Física

O SmartSwitch deve ser instalado em uma máquina da rede corporativa da Chesf. É necessário que um servidor *web* em execução nessa máquina, uma vez que o SmartSwitch precisa de um container *web* para funcionar.

Além do servidor *web*, os seguintes sistemas ou recursos precisam ser acessados para que o SmartSwitch funcione corretamente:

- **Banco de dados:** O banco de dados pode estar rodando em qualquer máquina da rede corporativa. Devido ao fato da quantidade de dados ser pequena e pelo fato do SmartSwitch não precisar de muito processamento ou memória, o banco de dados pode estar rodando na mesma máquina do SmartSwitch;

- **SisRTM:** O SisRTM encontra-se instalado em um cluster localizado na rede corporativa. O acesso é feito a um balanceador de carga que direciona as requisições a uma das máquinas do cluster. O SmartSwitch acessa o SisRTM através de chamadas

HTTP/REST. Algumas telas do SmartSwitch consistem de telas do SisRTM que são acessadas pelo usuário através de chamadas HTTP;

- **SAGE das Regionais:** Uma rede dedicada ao SAGE é utilizada para evitar que eventuais problemas na rede corporativa comprometa o funcionamento do sistema de supervisão.

O SmartSwitch acessa todos os centros de controle da Chesf. Esse acesso é realizado através das máquinas Smart localizadas em cada regional. Cada máquina Smart possui um módulo do Smart Model que conhece o estado topológico de todos os equipamentos supervisionados pelo sistema SCADA. O Smart Model acessa o SAGE através do módulo Smart Gateway implantado nos servidores do SAGE. O Smart Gateway recupera as informações do SAGE através do *logs* e da base de tempo real do mesmo. As máquinas Smart localizam-se na mesma rede do SAGE. A comunicação entre o SmartSwitch e o Smart Model é realizada através de chamadas REST.

A topologia do sistema Chesf é recuperada através do módulo Topogiggio localizado nos servidores Smart nos centros de controle. O SmartSwitch recupera o arquivo contendo os dados topológicos através do protocolo SCP.

As telas dos unifilares são recuperadas acessando diretórios específicos dos servidores SAGE. Existem duas alternativas: (i) acessar o servidor do COOS (que contém todos os unifilares da Chesf); (ii) acessar os servidores SAGE das regionais (cada servidor possui os arquivos específicos de sua regional).

Apêndice C

Documento Arquitetural do Projeto PSI

Universidade Federal de Campina Grande

Secretaria de Planejamento e Orçamento

Serviço da Tecnologia da Informação

Portal de Sistemas Integrados - PSI

Documento de Arquitetura de Software

Campina Grande, Paraíba, Brasil

Março, 2013

Revisões

Autores	Descrição	Versão	Data
Leandro José	Adição de novas informações necessárias diante da validação da arquitetura utilizando o MVAS.	1.1	11/03/2013
Leandro José	Versão inicial do documento de arquitetura do Portal de Sistemas Integrados.	1.0	03/01/2013

C.1 Introdução

O conteúdo desta seção, que pode ser lido em A.1, foi removido para evitar repetições no documento.

C.2 Termos e Definições

Parte do conteúdo desta seção, que pode ser lido em A.2, foi removido para evitar repetições no documento.

C.2.1 Sistema integrado

Qualquer sistema desenvolvido pela instituição e acessível pelo PSI é considerado um sistema integrado. Eis alguns exemplos de futuros sistemas integrados:

- Sistema de controle de bolsas (SICOB);
- Sistema de relatórios para pontos eletrônicos (SIRPE);
- Questionário do Plano diretor de TI (Equest).

C.3 Sistema de Interesse

A UFCG é composta cerca de milhares de servidores e alunos, mas poucos são os sistemas desenvolvidos e implantados para auxiliar os diversos processos existentes na instituição.

Os sistemas existentes sofrem de problemas como:

- Falta de um processo de desenvolvimento definido;
- Falta de um conjunto padrão de tecnologias utilizadas;
- Falta de um padrão de nomenclatura para bancos de dados;
- Falta de um arcabouço para o desenvolvimento de sistemas;
- Falta de um ponto único para acesso aos sistemas da instituição.

Os três primeiros problemas citados foram resolvidos para os sistemas desenvolvidos a partir de 2011, o penúltimo foi resolvido parcialmente e o último problema ainda existe. Com isso, o objetivo do PSI é fornecer um arcabouço para futuros sistemas a serem desenvolvidos e um meio de acesso a todos os sistemas da instituição.

O arcabouço será um conjunto de bibliotecas de software que serão utilizadas na produção de novos sistemas institucionais. O ponto único de acesso será um sistema *web* que permitirá o uso de diversos sistemas pelos alunos e servidores da instituição, de forma que a autenticação seja feita somente uma vez.

C.3.1 Partes Interessadas

Segue abaixo a listagem de todas as partes interessadas envolvidas no projeto:

Tipo	**Nome**
Usuários finais	Servidores Públicos da Universidade Federal de Campina Grande
Clientes	Oscar William
	Marzina Vidal
Gerente de projeto	Ianna Kobayashi
Arquiteto	Leandro José
Desenvolvedores	Diogo Vilar
	José Flávio
	Leandro José
	Sebastião Lemos
Arquitetos	Eloi Rocha Neto
	Leandro José Ventura Silva
Testador	Camila de Luna
Implantador	Leandro José

C.3.2 Interesses

Esta seção documenta os interesses dos participantes, além de identificar quais deles possuem cada interesse.

Requisitos Funcionais

Todos os requisitos **funcionais** a seguir são de interesse do **cliente**.

Autenticação

Identificador:	Efetuar login
Descrição:	O usuário deve se autenticar para ter acesso aos sistemas da instituição.
Interesses relacionados:	Alterar senha Recuperar login Recuperar senha

Identificador:	Alterar senha
Descrição:	Após se autenticar, o usuário pode alterar sua senha, devendo informar a senha antiga e a nova senha duas vezes.
Interesses relacionados:	Efetuar login Recuperar login Recuperar senha

Identificador:	Recuperar login
Descrição:	Caso o usuário perca o login é possível que o mesmo seja enviado para o e-mail associado à conta.
Interesses relacionados:	Efetuar login Alterar senha Recuperar senha

Identificador:	Recuperar senha
Descrição:	Caso o usuário perca a senha, é possível que uma nova seja gerada e enviada para o e-mail associado à conta.
Interesses relacionados:	Efetuar login Alterar senha Recuperar login

Identificador:	Acessar um sistema
Descrição:	Após a autenticação, o usuário deve visualizar a lista de sistemas disponíveis e acessar qualquer um.
Interesses relacionados:	Nenhum.

Administração

Identificador:	Criar usuário
Descrição:	Novos usuários podem ser criados e devem ser associados a um ou mais sistemas. Para cada sistema, diferentes papéis podem ser aplicados.
Interesses relacionados:	Alterar usuário Desativar usuário Buscar usuários

Identificador:	Alterar usuário
Descrição:	Usuários existentes podem ser alterados com relação aos sistemas acessíveis, papéis associados, senha, e-mail, situação e outros.
Interesses relacionados:	Criar usuário Desativar usuário Buscar usuários

Identificador:	Desativar usuário
Descrição:	Um usuário pode ser desativado, ou seja, ele não poderá se autenticar no PSI e acessar os sistemas institucionais.
Interesses relacionados:	Criar usuário Alterar usuário Buscar usuários

Identificador:	Buscar usuários
Descrição:	A busca de usuários é feita por login, sistemas associados, nome, matrícula SIAPE e outras opções.
Interesses relacionados:	Criar usuário Alterar usuário Desativar usuário

Identificador:	Criar papel
Descrição:	Cada sistema possui um conjunto fixo de permissões existentes; já os papéis podem ser criados dinamicamente.
Interesses relacionados:	Alterar papel Remover papel Buscar papéis

Identificador:	Alterar papel
Descrição:	Um papel pode ser alterado a fim de adicionar novas permissões ou remover alguma existente.
Interesses relacionados:	Criar papel Remover papel Buscar papéis

Identificador:	Remover papel
Descrição:	Um papel pode ser removido caso não exista nenhum usuário ou sistema associado a ele.
Interesses relacionados:	Criar papel Alterar papel Buscar papéis

Identificador:	Buscar papéis
Descrição:	A busca de papéis é feita por nome ou sistema associado.
Interesses relacionados:	Criar papel Alterar papel Remover papel

Identificador:	Atualizar banco de dados de servidores
Descrição:	Todo mês uma base atualizada dos servidores da UFCG é gerada e deve ser atualizada no PSI. Essa atualização deve estar disponível na interface gráfica.
Interesses relacionados:	Nenhum.

Identificador:	Monitorar sistemas integrados
Descrição:	Qualquer sistema integrado ao PSI será monitorável via interface gráfica. As seguintes informações devem ser exibidas: • Status atual: *online* ou *offline*; • Último horário da checagem; • Memória consumida na JVM; • Memória total da JVM; • Quantidade de threads em execução; • Tempo médio de resposta das últimas 100 requisições; • Espaço livre em disco. Os sistemas podem ser monitorados em tempo real ou um período de tempo pode ser escolhido. O PSI deve dar a opção de informar valores mínimos, médios e máximos para cada métrica informada.
Interesses relacionados:	RNF - API para integração

Identificador:	Gerar relatórios de acesso
Descrição:	Deve ser possível gerar relatórios diários, semanais, mensais ou anuais de acesso por servidor ou setor. Esses relatórios podem ser usados para gerar estatísticas de acesso ou para auditoria.
Interesses relacionados:	Nenhum.

Identificador:	Visualizar propriedades
Descrição:	Algumas propriedades do sistema devem ser visualizadas na interface gráfica, como por exemplo: quantidade máxima de tentativas inválidas de acesso; período máximo sem trocar a senha; quantidade mínima de letras e números na senha; etc.
Interesses relacionados:	Alterar propriedade

Identificador:	Alterar propriedade
Descrição:	Todas as propriedades visualizáveis na interface gráfica podem ser alteradas. As mudanças devem ser propagadas imediatamente.
Interesses relacionados:	Visualizar propriedades

Requisitos Não-funcionais

Identificador:	Usuários simultâneos
Atributos:	Desempenho
Descrição:	O sistema deve ter tempos de resposta de até 15 segundos para as funcionalidades mais comuns considerando 200 sessões simultâneas.
Interesses relacionados:	RNF - Várias instâncias em execução
Partes interessadas:	• Clientes; • Testador: responsáveis pela preparação de testes de carga para garantir as condições impostas por esse requisito.

Identificador:	Sistema portável
Atributos:	Portabilidade
Descrição:	O sistema deve ser executável em diversos sistemas operacionais: Windows e Linux.
Interesses relacionados:	Nenhum.
Partes interessadas:	• Clientes; • Arquiteto: responsável pela escolha das tecnologias a serem empregadas.

Identificador:	Interface *web*
Atributos:	Tipo da interface
Descrição:	O sistema deve possuir interface *web*, permitindo assim seu uso com independência de localização. A interface do sistema, mesmo sendo *web*, deve ser semelhante a aplicações *desktop*: uso de janelas (redimensionáveis, minimizáveis e maximizáveis), caixas de diálogo, abas, etc.
Interesses relacionados:	Nenhum.
Partes interessadas:	Clientes.

Identificador:	Baixo acoplamento
Atributos:	Extensibilidade Manutenibilidade
Descrição:	Deve-se primar pelo baixo acoplamento no sistema a fim de facilitar a manutenção, evitar mudanças em cascata, etc.
Interesses relacionados:	Nenhum.
Partes interessadas:	• Arquiteto: responsável pela escolha e validação das técnicas a serem empregadas para diminuir o acoplamento; • Desenvolvedores: responsáveis pela implementação das técnicas escolhidas.

Identificador:	Testes automáticos
Atributos:	Testabilidade
Descrição:	Todos os testes que existirem no sistema, uma vez iniciados, não devem requerer a intervenção humana.
Interesses relacionados:	Nenhum.
Partes interessadas:	• Arquiteto: responsável pela escolha das técnicas de teste a serem empregadas; • Desenvolvedores: responsáveis pelos testes de unidade e de carga; • Clientes: responsáveis pela validação dos testes.

Identificador:	Operações registradas
Atributos:	Auditabilidade Segurança
Descrição:	Todas as operações no sistema devem ser registradas, para facilitar a auditoria e a descoberta de possíveis *bugs* e falhas de software.
Interesses relacionados:	RF - Gerar relatórios de acesso
Partes interessadas:	• Clientes; • Arquiteto: responsável pela escolha da biblioteca a ser utilizada e pelo que deve ser registrado.

Identificador:	Acesso restrito
Atributos:	Segurança
Descrição:	Todas as operações no sistema devem ser feitas somente por usuários autenticados.
Interesses relacionados:	RF - Efetuar login RNF - Login único RNF - Comunicação segura
Partes interessadas:	• Clientes; • Arquiteto: responsável pela escolha da técnica de autenticação a ser utilizada.

Identificador:	Login único
Atributos:	Segurança
Descrição:	Uma vez que o usuário se autenticou no PSI, não deve ser necessário o uso de login e senha para acessar os sistemas integrados.
Interesses relacionados:	RF - Efetuar login RNF - Acesso restrito RNF - API para integração
Partes interessadas:	• Clientes; • Arquiteto: responsável pela escolha da técnica de autenticação a ser utilizada.

Identificador:	Várias instâncias em execução
Atributos:	Tolerância a falhas Escalabilidade Desempenho
Descrição:	Deve ser possível que mais de uma instância da aplicação rode ao mesmo tempo para evitar pontos únicos de falha e diminuir o tempo médio de resposta do PSI.
Interesses relacionados:	RNF - Usuários simultâneos
Partes interessadas:	• Clientes; • Arquiteto: responsável pela escolha das técnicas e tecnologias a serem utilizadas.

Identificador:	*Backups* regulares
Atributos:	Confiabilidade Segurança
Descrição:	*Backups* devem ser feitos regularmente em máquinas diferentes e se possível em locais externos aos servidores de produção.
Interesses relacionados:	Nenhum.
Partes interessadas:	• Clientes: devem verificar a necessidade de um local externo para o armazenamento dos backups; • Arquiteto: responsável pela escolha das técnicas e tecnologias a serem utilizadas.

Identificador:	API para integração
Atributos:	Integração com outros produtos
Descrição:	Uma API deve ser fornecida para que os sistemas integrados se comuniquem com o PSI.
Interesses relacionados:	RNF - Login único RF - Monitorar sistemas integrados RF - Fornecer dados de autenticação para outros sistemas
Partes interessadas:	• Arquiteto: responsável pela especificação da API; • Desenvolvedores: responsáveis pela implementação da API.

Identificador:	Fornecer dados de autenticação para outros sistemas
Atributos:	Integração com outros produtos
Descrição:	Os dados de autenticação devem ser disponibilizados para sistemas desenvolvidos por terceiros de forma segura a fim de evitar múltiplos logins e senhas para os servidores da instituição.
Interesses relacionados:	RNF - API para integração
Partes interessadas:	• Clientes; • Gerente de projeto.

Identificador:	Permissões por sistema
Atributos:	Segurança
Descrição:	Cada sistema integrado possui um conjunto fixo de permissões. Uma permissão é associada a um sistema e pode ser associada a mais de um papel.
Interesses relacionados:	RF - Criar papel RF - Alterar papel RF - Remover papel RNF - Papéis por sistema
Partes interessadas:	• Clientes; • Arquiteto.

Identificador:	Papéis por sistema
Atributos:	Segurança
Descrição:	Papéis são conjuntos de permissões e são associados a sistemas e a usuários desses sistemas.
Interesses relacionados:	RF - Criar papel RF - Alterar papel RF - Remover papel RNF - Permissões por sistema
Partes interessadas:	• Clientes; • Arquiteto.

Identificador:	Bloqueio de acessos inválidos
Atributos:	Segurança
Descrição:	Caso existam várias tentativas inválidas de acesso dentro de um intervalo estabelecido, o usuário e IP de origem devem ser bloqueados.
Interesses relacionados:	RNF - Acesso restrito
Partes interessadas:	• Clientes; • Gerente de projeto.

Identificador:	Exigir troca de senha periodicamente
Atributos:	Segurança
Descrição:	Todo usuário deve trocar sua senha quando a senha permanecer a mesma por um tempo maior que o estabelecido em propriedade.
Interesses relacionados:	RNF - Acesso restrito RF - Visualizar propriedades RF - Alterar propriedades
Partes interessadas:	• Clientes; • Gerente de projeto.

Riscos

Identificador:	Permitir várias instâncias simultâneas do sistema em execução
Descrição:	Nenhum dos envolvidos no projeto tem experiência em aplicações que permitam várias instâncias em execução ao mesmo tempo.
Interesses relacionados:	RNF - Várias instâncias em execução
Providências a serem tomadas:	Estudar técnicas para desenvolver aplicações com alta disponibilidade e escalabilidade.
Partes interessadas:	Arquiteto e desenvolvedores.

C.4 Pontos de Vista

O conteúdo desta seção, que pode ser lido em A.4, foi removido para evitar repetições no documento.

C.5 Visões

C.5.1 Visão Lógica

Nesta seção serão vistos termos e definições utilizados para esta visão, decisões-chave para a estrutura estática e comportamental do sistema, modelos em UML e a especificação dos elementos de software expressos nos modelos.

Termos e definições

REST *REpresentational State Transfer* (REST) é um estilo de arquitetura de software para sistemas distribuidos como a *World Wide Web*. Arquiteturas no estilo REST consistem de clientes e servidores. Clientes iniciam requisições aos servidores; servidores as processam e retornam respostas apropriadas. Requisições e respostas são construídas em torno da transferência de representações de recursos. Um recurso pode ser qualquer conceito coerente e com algum significado que seja endereçavel. A representação de um recurso é tipicamente um documento que captura o estado corrente de um recurso.

Java Java é uma linguagem de programação orientada a objeto desenvolvida na década de 90 por uma equipe de programadores chefiada por James Gosling, na empresa *Sun Microsystems*. Diferentemente das linguagens convencionais, que são compiladas para código nativo, a linguagem Java é compilada para um *bytecode* que é executado por uma máquina virtual. A linguagem de programação Java é a linguagem convencional da Plataforma Java, mas não sua única linguagem.

GWT *Google Web Toolkit* (GWT) é um conjunto de ferramentas para desenvolver e otimizar aplicações *web* complexas.

XML XML, *eXtensible Markup Language*, é uma linguagem de marcadores desenhada para descrever dados de forma hierárquica.

JavaScript JavaScript é uma linguagem de programação criada pela *Netscape* em 1995 que, no lado cliente (o navegador do usuário), efetua comandos sem a necessidade de se processar no lado servidor (o servidor em que a aplicação está hospedada). Sua sintaxe é semelhante à do Java [61].

Façade É um objeto que provê uma interface simplificada para uma grande quantidade de código.

Apache Tomcat Servidor *web* que proporciona um ambiente para aplicações em Java serem executadas.

JAR Um arquivo no formato JAR, *Java ARchive*, é utilizado para distribuir classes Java, metadados e recursos como imagens, textos, etc.

WAR Um arquivo no formato WAR, *Web Application ARchive*, é um arquivo JAR utilizado para distribuir uma aplicação *web* feita em Java.

AJAX AJAX, *Asynchronous Javascript and XML*, é o uso metodológico de tecnologias como Javascript e XML, providas por navegadores, para tornar páginas *web* mais interativas com o usuário, utilizando-se de solicitações assíncronas de informações.

Java Persistence API Define um meio de mapeamento objeto-relacional para objetos Java simples e comuns, denominados beans de entidade.

Callback Uma *callback* é uma referência a um pedaço de código executável que é passado como argumento para outro código.

SSO *Single sign-on* (SSO) é uma propriedade de controle de acesso a sistemas relacionados porém independentes. Com essa propriedade, um usuário se autentica uma vez e obtém acesso a todos os sistemas sem ser solicitado para se autenticar novamente em cada um deles.

JOSSO JOSSO, *Java Open Single Sign-On*, é uma solução em código-aberto desenvolvida em Java que implementa o padrão SSO.

SGBD SGBD, sistema de gerenciamento de banco de dados, é um sistema de software que usa um método padrão para catalogar, recuperar e executar consultas em dados. O SGBD gerencia dados de entrada, organiza-os e provê maneiras para que os mesmos sejam modificados ou extraídos por usuários ou outros programas. Eis alguns exemplos de SGBDs: MySQL, PostgreSQL, Microsoft Access, SQL Server, FileMaker, Oracle, RDBMS, dBase, Clipper e FoxPro.

LDAP *Lightweight Directory Access Protocol* (LDAP) ou Protocolo Leve de Acesso à Diretórios é um conjunto de protocolos desenhados para acessar informações centralizadas em uma rede.

O servidor LDAP tem a função de verificar as credenciais do cliente, verificar se as informações solicitadas estão armazenadas neste servidor e permitir ou não que o cliente realize consultas e modificações. As formas de armazenamento dos dados, tipo de gerenciador de bancos de dados usado e sistema operacional de base fazem parte da implementação específica do LDAP [62].

SSL SSL significa *Secure Sockets Layer* e é um protocolo de Internet para criptografia e autenticação baseada em sessão, fornecendo um canal seguro entre o cliente e o servidor. O

SSL fornece uma autenticação de servidor e uma autenticação opcional de cliente para derrotar espionagem, adulteração e falsificação de mensagens em aplicativos cliente-servidor. [27]

WYSIWIG Em computação, um editor WYSIWIG, *"what you see is what you get"* é um sistema cujo conteúdo (textos e gráficos) que é exibido na tela durante a edição está próximo de sua aparência quando impresso ou exibido como o produto final.

Decisões-chave

Esta seção trás uma tabela com o relacionamento entre interesses e as decisões-chave tomadas; em seguida, a explicação detalhada de cada decisão-chave citada.

Interesse	**Decisões-chave**
RF - Efetuar login	Utilizar Java Open Single Sign On (JOSSO) Utilizar login definido pelo servidor público Utilizar um SGBD para armazenar dados Utilizar uma implementação da *Java Persistence API* (JPA)
RF - Alterar senha	Utilizar um SGBD para armazenar dados Armazenar a senha criptografada Recusar senhas já utilizadas Utilizar uma implementação da *Java Persistence API* (JPA)
RF - Recuperar login	Utilizar um SGBD para armazenar dados Utilizar uma implementação da *Java Persistence API* (JPA)

RF - Recuperar senha	Utilizar um SGBD para armazenar dados Armazenar a senha criptografada Recusar senhas já utilizadas Utilizar uma implementação da *Java Persistence API* (JPA) Definir padrão seguro para senhas
RF - Acessar um sistema	Utilizar *Java Open Single Sign On* (JOSSO) Sistemas integrados podem estar implantados em diferentes servidores de aplicação
RF - Criar usuário	Utilizar um SGBD para armazenar dados Utilizar login definido pelo servidor público Armazenar a senha criptografada Utilizar uma implementação da *Java Persistence API* (JPA) Exigir permissões de administrador para gerenciar usuários e papéis Todo usuário deve possuir um e-mail institucional Criar múltiplos usuários para casos em que o servidor público possui dois cargos na mesma instituição Definir padrão seguro para senhas

RF - Alterar usuário	Utilizar um SGBD para armazenar dados Armazenar a senha criptografada Recusar senhas já utilizadas Utilizar uma implementação da *Java Persistence API* (JPA) Exigir permissões de administrador para gerenciar usuários e papéis Todo usuário deve possuir um e-mail institucional Definir padrão seguro para senhas
RF - Desativar usuário	Utilizar um SGBD para armazenar dados Utilizar uma implementação da *Java Persistence API* (JPA) Exigir permissões de administrador para gerenciar usuários e papéis Bloquear acesso de usuários desativados
RF - Buscar usuários	Utilizar um SGBD para armazenar dados Utilizar uma implementação da *Java Persistence API* (JPA) Exigir permissões de administrador para gerenciar usuários e papéis
RF - Criar papel	Utilizar um SGBD para armazenar dados Utilizar uma implementação da *Java Persistence API* (JPA) Exigir permissões de administrador para gerenciar usuários e papéis

RF - Alterar papel	Utilizar um SGBD para armazenar dados Utilizar uma implementação da *Java Persistence API* (JPA) Exigir permissões de administrador para gerenciar usuários e papéis
RF - Remover papel	Utilizar um SGBD para armazenar dados Utilizar uma implementação da *Java Persistence API* (JPA) Exigir permissões de administrador para gerenciar usuários e papéis
RF - Buscar papéis	Utilizar um SGBD para armazenar dados Utilizar uma implementação da *Java Persistence API* (JPA) Exigir permissões de administrador para gerenciar usuários e papéis
RF - Monitorar sistemas integrados	Registrar tempos de resposta para qualquer acesso à *façade* Armazenar dados de monitoramento dos sistemas integrados Utilizar um SGBD para armazenar dados
RF - Atualizar banco de dados de servidores	Utilizar um SGBD para armazenar dados Utilizar uma implementação da *Java Persistence API* (JPA)
RF - Gerar relatórios de acesso	Utilizar *JasperReports* na geração de relatórios
RF - Visualizar propriedades	Utilizar um SGBD para armazenar dados Utilizar uma implementação da *Java Persistence API* (JPA)

RF - Alterar propriedade	Utilizar um SGBD para armazenar dados Utilizar uma implementação da *Java Persistence API* (JPA)
RNF - Usuários simultâneos	Realizar testes de carga Utilizar balanceadores de carga Utilizar *caches* distribuídos quando necessário Configurar replicação no SGBD
RNF - Sistema portável	Utilizar Java O sistema deve permitir implantação no servidor *Apache Tomcat*
RNF - Interface *web*	Utilizar *Google Web Toolkit* (GWT) O sistema deve permitir implantação no servidor *Apache Tomcat* Utilizar SSL
RNF - Baixo acoplamento	Utilizar uma *façade* para acessar a lógica de negócio
RNF - Testes automáticos	Utilizar uma *façade* para acessar a lógica de negócio Usar o *EasyAccept* para testes de aceitação Usar o *Selenium* para testes da interface gráfica
RNF - Operações registradas	Registrar qualquer acesso à *façade* Registrar tempos de resposta para qualquer acesso à *façade*
RNF - Acesso restrito	Utilizar *Java Open Single Sign On* (JOSSO) Adaptar GWT para integrar com a biblioteca JOSSO

RNF - Login único	Utilizar *Java Open Single Sign On* (JOSSO) Adaptar GWT para integrar com a biblioteca JOSSO Sistemas integrados podem estar implantados em diferentes servidores de aplicação
RNF - Várias instâncias em execução	Utilizar *caches* distribuídos quando necessário Configurar replicação no SGBD Utilizar balanceadores de carga
RNF - *Backups* regulares	Criar *scripts* para *backups* dos bancos de dados do SGBD
RNF - API para integração	Criar interfaces REST e bibliotecas Java para sistemas integrados Utilizar *RestEasy* Utilizar SSL
RNF - Permissões por sistema	Utilizar um SGBD para armazenar dados
RNF - Papéis por sistema	Utilizar um SGBD para armazenar dados
RNF - Bloqueio de acessos inválidos	Utilizar *Java Open Single Sign On* (JOSSO)
RNF - Fornecer dados de autenticação para outros sistemas	Utilizar LDAP para fornecer informações de autenticação para sistemas de terceiros
RNF - Comunicação segura	Utilizar SSL
RNF - Exigir troca de senha periodicamente	Recusar senhas já utilizadas

Seguem abaixo as decisões-chave de forma detalhada:

Decisão-chave:	Utilizar *Java Open Single Sign On* (JOSSO)
Interesses relacionados:	RF - Efetuar login RF - Acessar um sistema RNF - Acesso restrito RNF - Login único RNF - Bloqueio de acessos inválidos
Descrição:	A biblioteca JOSSO será utilizada para evitar que o usuário se autentique mais de uma vez para acessar diferentes sistemas integrados na mesma sessão.
Vantagens:	• Biblioteca de código aberto; • Utilizada em grandes instituições; • Permite o login único mesmo que as aplicações estejam em diferentes servidores de aplicação.
Desvantagens:	• Suporte pago; • Pouca documentação; • Versão gratuita não funciona em um cluster.

Decisão-chave:	Utilizar login definido pelo servidor público
Interesses relacionados:	RF - Efetuar login RF - Criar usuário
Descrição:	Um login definido pelo usuário evita que o servidor público precise decorar sua matrícula SIAPE ou qualquer código composto por números. Pode-se sugerir ao usuário um login padrão, por exemplo: **nome.sobrenome**.
Vantagens:	Facilidade para decorar os dados de autenticação.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Configurar replicação no SGBD
Interesses relacionados:	RNF - Usuários simultâneos RNF - Várias instâncias em execução
Descrição:	O tempo de resposta de consultas ao banco de dados pode ser reduzido com múltiplas instâncias do SGBD em execução porque consultas para leitura são realizadas em qualquer instância e consultas para atualização não precisam ser feitas de forma síncrona.
Vantagens:	• Aumento da disponibilidade do sistema; • Menores tempos de resposta.
Desvantagens:	• Complexidade da configuração; • Necessidade de maior monitoramento dos serviços; • Possibilidade de restrições impostas pela tecnologia utilizada para replicar os dados.

Decisão-chave:	Utilizar um SGBD para armazenar dados
Interesses relacionados:	RF - Efetuar login RF - Alterar senha RF - Recuperar login RF - Recuperar senha RF - Criar usuário RF - Alterar usuário RF - Desativar usuário RF - Buscar usuários RF - Criar papel RF - Alterar papel RF - Remover papel RF - Buscar papéis RF - Atualizar banco de dados de servidores RF - Visualizar propriedades RF - Alterar propriedade RNF - Permissões por sistema RNF - Papéis por sistema
Descrição:	SGBDs facilitam a gravação de dados no disco de forma confiável e permitem múltiplas leituras e gravações concorrentes.
Vantagens:	• Simplicidade; • Confiabilidade; • Segurança.
Desvantagens:	Custo inicial para projetar a estrutura do banco.

Decisão-chave:	Utilizar uma implementação da *Java Persistence API* (JPA)
Interesses relacionados:	RF - Efetuar login RF - Alterar senha RF - Recuperar login RF - Recuperar senha RF - Criar usuário RF - Alterar usuário RF - Desativar usuário RF - Buscar usuários RF - Criar papel RF - Alterar papel RF - Remover papel RF - Buscar papéis RF - Atualizar banco de dados de servidores RF - Visualizar propriedades RF - Alterar propriedade
Descrição:	O uso da *Java Persistence API* facilitará o mapeamento entre objetos e tabelas do banco.
Vantagens:	• Não há necessidade de grande conhecimento de SQL; • Existem várias implementações da JPA.
Desvantagens:	Pequena perda de desempenho.

Decisão-chave:	Criar múltiplos usuários para casos em que o servidor público possui dois cargos na mesma instituição
Interesses relacionados:	RF - Criar usuário
Descrição:	Para evitar maior complexidade para os sistemas integrados, um servidor público poderá ter mais de um login caso ele possua mais de um cargo na instituição.
Vantagens:	Implementação simplificada.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Armazenar a senha criptografada
Interesses relacionados:	RF - Alterar senha
	RF - Recuperar senha
	RF - Criar usuário
	RF - Alterar usuário
Descrição:	Para maior segurança dos dados e do próprio servidor, sua senha deve ser criptografada ao armazenar.
Vantagens:	Segurança.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Definir padrão seguro para senhas
Interesses relacionados:	RF - Alterar senha
	RF - Recuperar senha
	RF - Criar usuário
	RF - Alterar usuário
Descrição:	Todas as senhas utilizadas no sistema devem seguir um padrão seguro para evitar o uso de senhas fáceis pelos usuários.
Vantagens:	Maior segurança.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Todo usuário deve possuir um e-mail institucional
Interesses relacionados:	RF - Criar usuário RF - Alterar usuário
Descrição:	Para uma forma de comunicação mais segura, os usuários devem ter um e-mail institucional associado ao PSI.
Vantagens:	• Evitar falha no envio de e-mails a partir do PSI para e-mails não institucionais; • Auditoria mais detalhada.
Desvantagens:	Necessidade de servidores adequados para hospedar os e-mails de todos os servidores públicos com disponibilidade e tempos de resposta aceitáveis.

Decisão-chave:	Utilizar LDAP para fornecer informações de autenticação para sistemas de terceiros
Interesses relacionados:	RNF - Fornecer dados de autenticação para outros sistemas
Descrição:	Muitos sistemas permitem autenticação utilizando LDAP, facilitando a sua implantação em ambiente institucional.
Vantagens:	• Alta compatibilidade; • Evitar logins e senhas para diferentes aplicações de terceiros; • Escalabilidade.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Sistemas integrados podem estar implantados em diferentes servidores de aplicação
Interesses relacionados:	RF - Acessar um sistema RNF - Login único
Descrição:	Sistemas integrados implantados em diferentes servidores evitam pontos únicos de falha, isolam possíveis erros de execução e são escaláveis. Ainda assim, deve-se garantir o login único de um usuário.
Vantagens:	• Menores tempos de resposta; • Evita uso de um único servidor que hospeda todas as aplicações da instituição.
Desvantagens:	Configurações mais complexas.

Decisão-chave:	Bloquear acesso de usuários desativados
Interesses relacionados:	RF - Desativar usuário
Descrição:	O usuário autenticado deve ser verificado em cada acesso à *façade* e seu acesso deve ser bloqueado caso ele esteja desativado. Caso um usuário esteja autenticado e ele seja desativado nessa sessão, o mesmo não poderá realizar nenhuma ação.
Vantagens:	Maior segurança.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Exigir permissões de administrador para gerenciar usuários e papéis
Interesses relacionados:	RF - Criar usuário RF - Alterar usuário RF - Desativar usuário RF - Buscar usuários RF - Criar papel RF - Alterar papel RF - Remover papel RF - Buscar papéis
Descrição:	Existirão dois tipos de usuários: administradores e super administradores. Os primeiros só podem gerenciar usuários e papéis de um número limitado de sistemas integrados. Já os segundos podem gerenciar usuários e papéis de quaisquer sistemas, assim como criar novos administradores e super administradores.
Vantagens:	• Maior flexibilidade; • Evita concentração de muitos poderes para um grupo pequeno de usuários.
Desvantagens:	Implementação mais complexa.

Decisão-chave:	Utilizar *JasperReports* na geração de relatórios
Interesses relacionados:	RF - Gerar relatórios de acesso
Descrição:	Deve-se utilizar *JasperReports* pois é uma biblioteca bastante utilizada e flexível na geração de diversos tipos de relatórios.
Vantagens:	• Biblioteca consolidada; • Muitos exemplos disponíveis; • Ferramenta de edição WYSIWIG.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Registrar tempos de resposta para qualquer acesso à *façade*
Interesses relacionados:	RF - Monitorar sistemas integrados RNF - Operações registradas
Descrição:	Qualquer acesso à *façade* deve ser monitorado para que possíveis gargalos sejam identificados e corrigidos.
Vantagens:	• Facilidade na identificação de gargalos; • Monitoramento detalhado.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Armazenar dados de monitoramento dos sistemas integrados
Interesses relacionados:	RF - Monitorar sistemas integrados
Descrição:	Os dados de monitoramento devem ser armazenados no SGBD para que seja possível verificar as indisponibilidades, tempos de resposta, espaço livre em disco, etc. de um sistema integrado em um certo período de tempo.
Vantagens:	Viabiliza informações de desempenho, disponibilidade, etc. do sistema em tempos anteriores.
Desvantagens:	Quantidade de registros do banco podem crescer rapidamente dependendo da quantidade de sistemas integrados e do intervalo de tempo para gravação dos dados de monitoramento.

Decisão-chave:	Realizar testes de carga
Interesses relacionados:	RNF - Usuários simultâneos
Descrição:	Testes de carga devem ser feitos para garantir que o sistema suporte as condições impostas pelo requisito não-funcional de interesse. Caso as condições não sejam satisfeitas, ferramentas de perfilamento para detectar gargalos deverão ser utilizadas.
Vantagens:	• Antecipar possíveis mudanças necessárias; • Garantir desempenho do sistema antes de entrar em produção.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Utilizar balanceadores de carga
Interesses relacionados:	RNF - Usuários simultâneos RNF - Várias instâncias em execução
Descrição:	Balanceadores de carga distribuem a carga para vários servidores em um *cluster*; links de rede; discos rígidos; etc. São utilizados para aumentar a vazão, diminuir o tempo de resposta e reduzir sobrecarga dos recursos.
Vantagens:	Escalabilidade.
Desvantagens:	• Maior complexidade; • Aplicações devem ser preparadas para permitir o balanceamento.

Decisão-chave:	Utilizar *caches* distribuídos quando necessário
Interesses relacionados:	RNF - Usuários simultâneos RNF - Várias instâncias em execução
Descrição:	Para que seja possível várias instâncias da aplicação em execução se faz necessário o uso de caches distribuídos sempre que existir dados que devem ser compartilhados entre as múltiplas instâncias da aplicação.
Vantagens:	• Permite várias instâncias em execução; • Evita pontos únicos de falha.
Desvantagens:	• Sistemas integrados devem ser adaptados para o seu uso; • Implementação mais complexa; • Pode causar erros inesperados por falha na conexão. Casos assim devem ser tratados.

Decisão-chave:	Utilizar Java
Interesses relacionados:	RNF - Sistema portável
Descrição:	A linguagem de desenvolvimento principal será Java e toda a parte da lógica do sistema deve ser desenvolvida utilizando-a.
Vantagens:	• Linguagem bastante utilizada no mercado; • Mantida por uma grande organização; • Grande número de ferramentas e bibliotecas gratuitas existentes.
Desvantagens:	Por ser uma linguagem de programação que não é compilada diretamente em linguagem de máquina, pode ser mais lenta para certas operações.

Decisão-chave:	O sistema deve permitir implantação no servidor *Apache Tomcat*
Interesses relacionados:	RNF - Sistema portável RNF - Interface *web*
Descrição:	O PSI deve possibilitar o empacotamento em um arquivo WAR para ser implantado no *Tomcat*.
Vantagens:	• *Apache Tomcat* é gratuito; • Utilizado mundialmente.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Utilizar *Google Web Toolkit* (GWT)
Interesses relacionados:	RNF - Interface *web*
Descrição:	GWT facilita a criação de aplicações *web* com suporte a AJAX e não obriga conhecimento em JavaScript, pois o código é feito em Java e o compilador GWT o transforma em Javascript.
Vantagens:	• A interface *web* pode ser escrita em Java ou XML; • Não há necessidade de conhecer JavaScript; • Funciona em diversos navegadores *web*.
Desvantagens:	Pode não funcionar corretamente em dispositivos móveis.

Decisão-chave:	Utilizar uma *façade* para acessar a lógica de negócio
Interesses relacionados:	RNF - Baixo acoplamento RNF - Testes automáticos
Descrição:	A camada de lógica de negócio deve ser acessada através de uma *façade*, um objeto único. Os métodos da *façade* permitem acessar toda a lógica de negócio.
Vantagens:	• Baixo acoplamento; • Ocultação de informação; • Facilita a expressão de testes de aceitação.
Desvantagens:	A *façade* tende a ficar muito grande quando a lógica de negócio é extensa.

Decisão-chave:	Usar o *EasyAccept* para testes de aceitação
Interesses relacionados:	RNF - Testes automáticos
Descrição:	Uma ferramenta especializada para expressar testes de aceitação e que possa executar todos os testes de lógica de negócio automaticamente deve ser utilizada.
Vantagens:	Facilidade de realizar os testes.
Desvantagens:	Provável necessidade de gerar mais uma *façade* a fim de facilitar e possibilitar os testes de aceitação.

Decisão-chave:	Usar o *Selenium* para testes da interface gráfica
Interesses relacionados:	RNF - Testes automáticos
Descrição:	O *Selenium* facilita e automatiza os testes de interfaces *web* e provê uma ferramenta para gravar e reproduzir os testes.
Vantagens:	Testes de interface gráfica são úteis pois testam a própria interface em um *browser* específico; a comunicação com o servidor e a lógica implementada.
Desvantagens:	• Exigem adaptações no código; • Complexidade para preparar testes.

Decisão-chave:	Registrar qualquer acesso à *façade*
Interesses relacionados:	RNF - Operações registradas
Descrição:	A *façade* deve registrar qualquer operação feita, incluindo falhas. O registro deve conter o horário, o usuário autenticado e descrição da operação.
Vantagens:	• Possibilita auditoria; • Facilita a descoberta de erros.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Adaptar GWT para integrar com a biblioteca JOSSO
Interesses relacionados:	RNF - Acesso restrito RNF - Login único
Descrição:	O JOSSO foi projetado para aplicações JSP, JSF e afins, de forma que, utilizando um arquivo de configuração, cada requisição a qualquer página *web* é verificada em termos de permissões. Como as aplicações em GWT geralmente só possuem uma página *web* e todas as trocas são feitas via AJAX, não é possível utilizar o JOSSO dessa maneira. Então, os sistemas desenvolvidos em GWT não devem possuir uma tela própria de login, pois o JOSSO possui tal tela; quaisquer páginas *web* devem ser restritas e a verificação das permissões às operações deve ser feita pelo próprio sistema.
Vantagens:	Existe a possibilidade de integrar JOSSO e GWT.
Desvantagens:	Não será possível utilizar todas as funcionalidades do JOSSO.

Decisão-chave:	Criar *scripts* para *backups* dos bancos de dados do SGBD
Interesses relacionados:	RNF - *Backups* regulares
Descrição:	Para evitar *backups* manuais, *scripts* devem ser criados para realizar o *backup* e a restauração dos bancos de dados, de forma que todas as operações sejam registradas. Caso algum erro ocorra, o responsável deve ser avisado via e-mail.
Vantagens:	Maior segurança dos dados.
Desvantagens:	Não evita completamente a perda de dados, pois o *backup* é feito periodicamente. Com isso, a replicação dos dados será feita para minimizar a perda.

Decisão-chave:	Criar interfaces REST e bibliotecas Java para sistemas integrados
Interesses relacionados:	RNF - API para integração
Descrição:	Para maior reuso de código e menor tempo gasto na codificação, bibiliotecas e interfaces REST serão acessíveis pelos sistemas integrados. Os recursos disponibilizados contemplarão métodos para monitoramento; gerenciamento de usuários e papéis; etc.
Vantagens:	• Reuso de código; • Facilita monitoramento dos sistemas.
Desvantagens:	As bibliotecas só poderão ser utilizadas por sistemas desenvolvidos em Java.

Decisão-chave:	Utilizar *RestEasy*
Interesses relacionados:	RNF - API para integração RNF - Utilizar SSL
Descrição:	*RestEasy* é uma implementação de REST em Java e é totalmente compatível com a especificação JSR 311.
Vantagens:	• Biblioteca atualizada frequentemente; • Boa documentação; • Compatível com o *Apache Tomcat*; • Servidor embutido para testes de unidade.
Desvantagens:	Nenhuma identificada.

Decisão-chave:	Utilizar SSL
Interesses relacionados:	RNF - Comunicação segura RNF - API para integração RNF - Utilizar SSL
Descrição:	Qualquer conexão REST ou conexão entre o navegador do usuário e os servidores de aplicação deve ser feita de forma segura para evitar a captura de dados por terceiros.
Vantagens:	Menor possibilidade da captura mal intencionada de informações.
Desvantagens:	• Maior quantidade de dados trafegados; • Maior processamento.

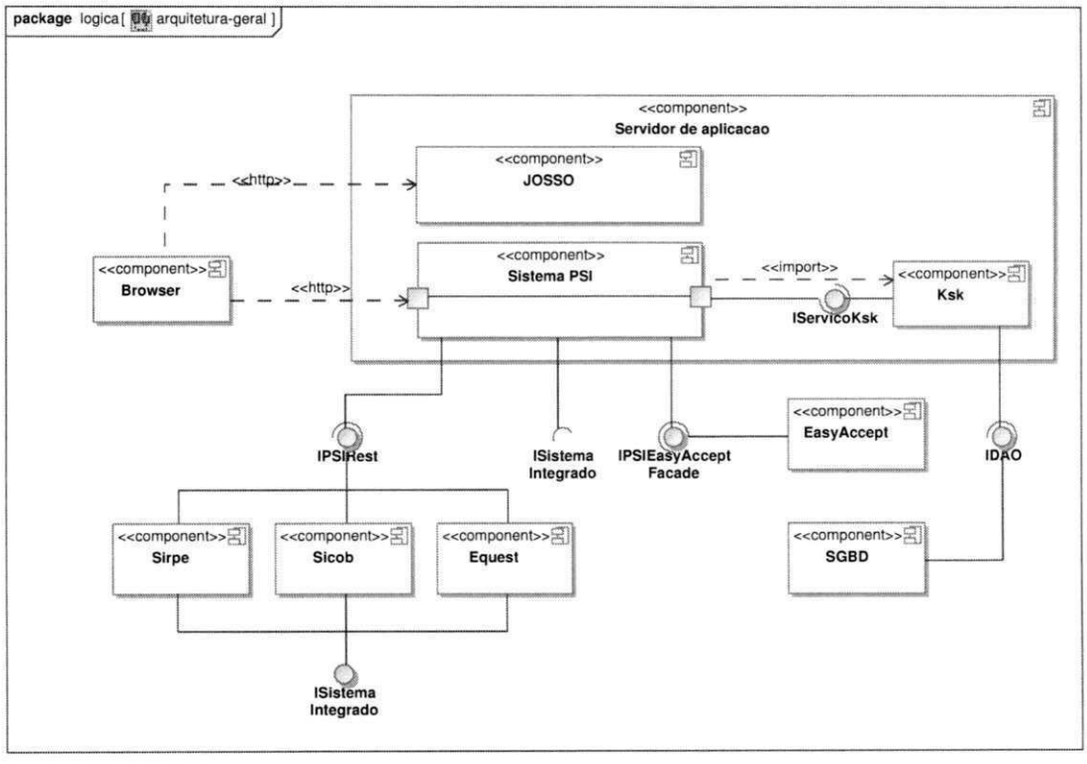
Modelos

Arquitetura Geral

- **`Sistema PSI`**: componente que representa o PSI, contemplando a lógica e as telas do sistema. Mais detalhes podem ser vistos em C.5.1;
- **`Ksk`**: biblioteca desenvolvida pelo STI que implementa funcionalidades comuns a diversos sistemas. Eis alguns exemplos: acesso ao SGBD utilizando JPA; componentes para interface gráfica; manipulação de arquivos; geração de relatórios; etc;
- O acesso à interface `IPSIRest` a partir dos sistemas integrados será feito utilizando REST, resultando em chamadas HTTP;
- Os três sistemas que utilizam a interface `IPSIRest` serão os primeiros a serem integrados;
- Os acessos à interface `ServicoKsk` serão feitos a partir do *browser* utilizando AJAX e resultarão em chamadas em HTTP. Essa comunicação será de forma assíncrona.

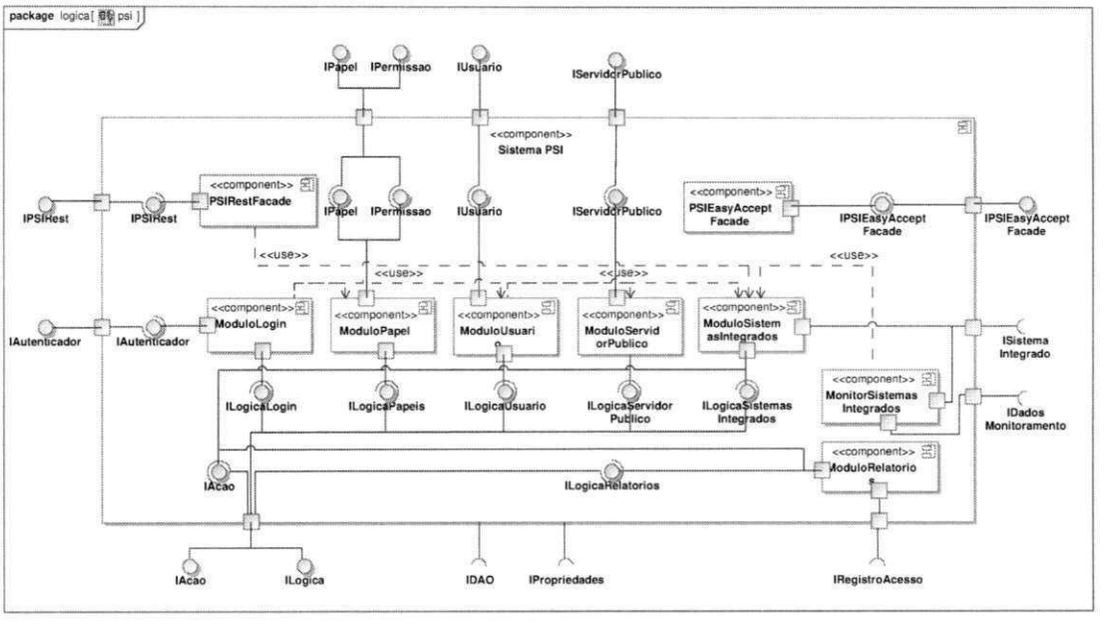
Componente `Sistema PSI`

Figura C.1: Arquitetura Geral do PSI



- **`PSIRestFacade`**: *façade* REST responsável por tratar as requisições dos sistemas integrados;
- **`PSIEasyAcceptFacade`**: *façade* acessada pelo *EasyAccept* para os testes de aceitação;
- **`ModuloLogin`**: responsável pela implementação da lógica de login, contemplando a autenticação e integração com o JOSSO;
- **`ModuloPapel`**: responsável pela implementação da lógica de papéis, contemplando inserção, alteração, busca e remoção, assim como a implementação das entidades que representam papéis e permissões;
- **`ModuloUsuario`**: responsável pela implementação da lógica de usuários, contemplando inserção, alteração, busca e desativação, assim como a implementação das entidades que representam os usuários;

Figura C.2: Modelo do sistema PSI



- **`ModuloServidorPublico`**: responsável pela implementação da lógica de servidores públicos, contemplando busca e importação dos mesmos, assim como a implementação das entidades que representam os servidores públicos;

- **`ModuloSistemasIntegrados`**: responsável pela implementação da lógica de sistemas integrados, contemplando informações sobre os sistemas existentes; quais estão ativos no momento; etc;

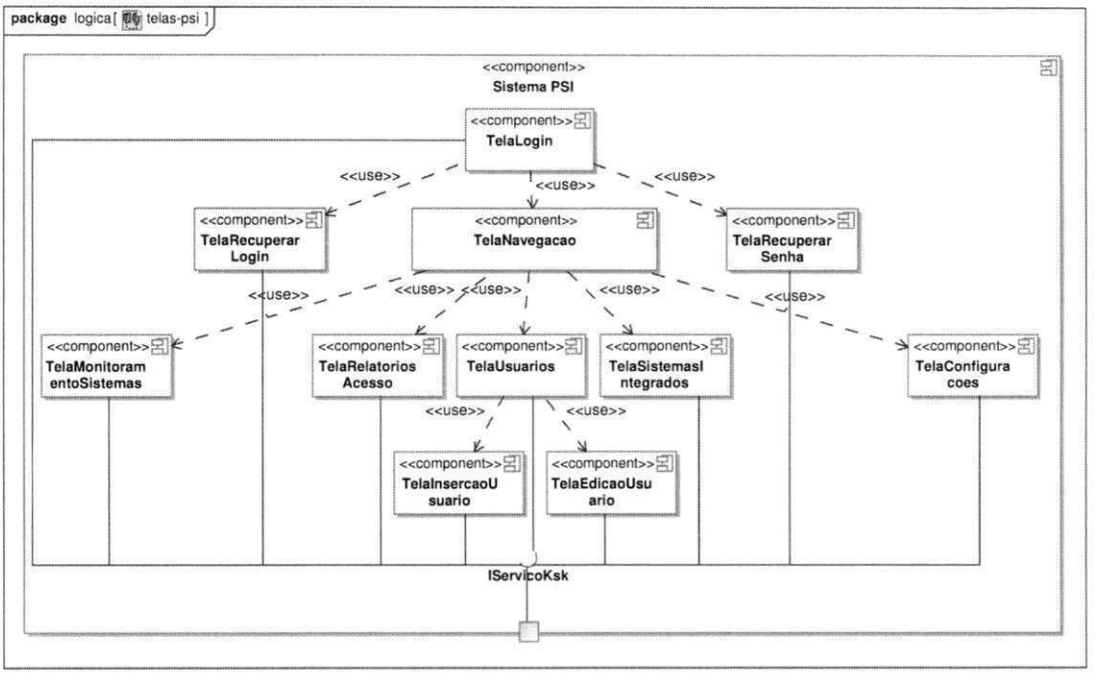
- **`ModuloRelatorios`**: responsável pela implementação da lógica de relatórios, contemplando a geração de relatórios de acesso;

- Cada componente que implementa uma lógica e várias ações é acessado pelo componente `ServicoKsk`, que por sua vez é acessado pelo *browser* do usuário.

- **`MonitorSistemasIntegrados`**: responsável pelo monitoramento dos sistemas integrados. Esse componente é utilizado pelo `ModuloSistemasIntegrados`. Mais detalhes podem ser vistos em C.5.1.

Telas do sistema PSI

Figura C.3: Modelo das telas do sistema PSI



- **`TelaLogin`**: responsável pela tela de login. O login e a senha devem ser informados para que a área restrita do sistema seja exibida. Provê acesso às telas de recuperar login, recuperar senha e navegação;

- **`TelaRecuperarLogin`**: responsável pela tela de recuperação do login. O usuário deve informar seu e-mail para que o login seja enviado por e-mail;

- **`TelaRecuperarSenha`**: responsável pela tela de recuperação da senha. O usuário deve informar seu e-mail e nome do usuário para que uma nova senha seja gerada e enviada por e-mail;

- **`TelaNavegacao`**: responsável pela tela que provê acesso a todas as funcionalidades restritas do sistema: monitoramento de sistemas; relatórios de acesso; gerenciamento de usuários; sistemas integrados e configurações;

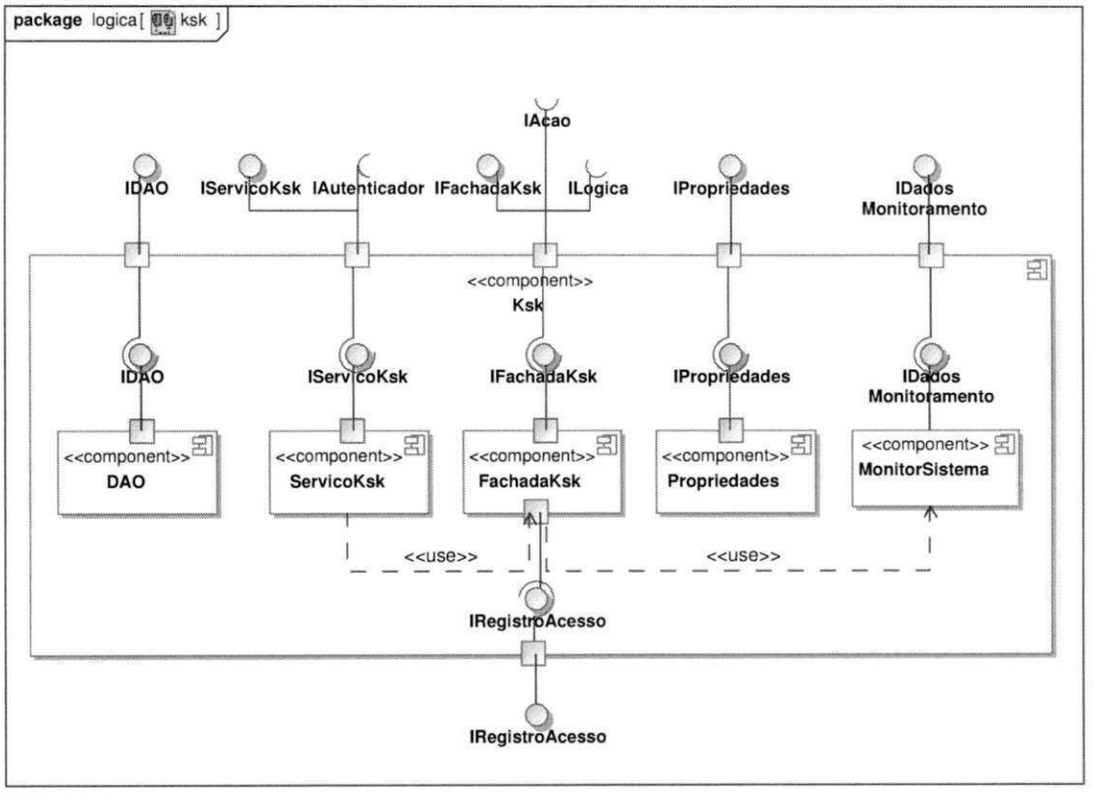
- **`TelaMonitoramentoSistemas`**: responsável pela tela de monitoramento de sistemas. Nessa tela é possível monitorar os sistemas em tempo real ou verificar as indisponibilidades em um determinado período;

- **TelaRelatoriosAcesso**: responsável pela tela de relatórios de acesso. Um período deve ser informado para que a geração seja feita;
- **TelaUsuarios**: responsável pela tela de gerenciamento de usuários. Nessa tela é possível inserir, editar, desativar e excluir usuários. Papéis também podem ser gerenciados nessa tela;
- **TelaInsercaoUsuario**: responsável pela tela de inserção de usuários;
- **TelaEdicaoUsuario**: responsável pela tela de edição de usuários;
- **TelaSistemasIntegrados**: responsável pela tela de acesso aos sistemas integrados. O usuário pode acessar qualquer sistema que seja exibido a ele;
- **TelaConfiguracoes**: responsável pela tela de configurações. As propriedades do sistema podem ser visualizadas e alteradas.

Componente Ksk

- **DAO**: responsável por gravar, recuperar, atualizar e remover dados no SGBD.
- **ServicoKsk**: componente responsável por implementar a *façade* que será acessada assincronamente pelos componentes relativos à interface gráfica;
- **FachadaKsk**: responsável pela *façade* que dá acesso às lógicas do sistema. Essa *façade* contém um conjunto de lógicas, `ILogica`, e é acessada utilizando um objeto que implementa uma `IAcao`. Cada `IAcao` é executada pela lógica responsável por ela. Mais informações podem ser encontradas em C.5.1;
- **Propriedades**: responsável por recuperar e armazenar propriedades, como quantidade máxima de tentativas inválidas de acesso; período máximo sem trocar a senha; quantidade mínima de letras e números na senha; etc;
- **MonitorSistema**: responsável pelo monitoramento do sistema atual, registrando informações como memória consumida na JVM; memória total da JVM; quantidade de *threads* em execução; etc.

Figura C.4: Modelo do componente `Ksk`



Cenário: Login Para ter acesso ao sistema, o usuário precisa informar seus dados de acesso. Eis as etapas que ocorrem:

1. O clique no botão de login causa uma requisição HTTP ao JOSSO;
2. O JOSSO verifica as credenciais enviadas via requisição HTTP;

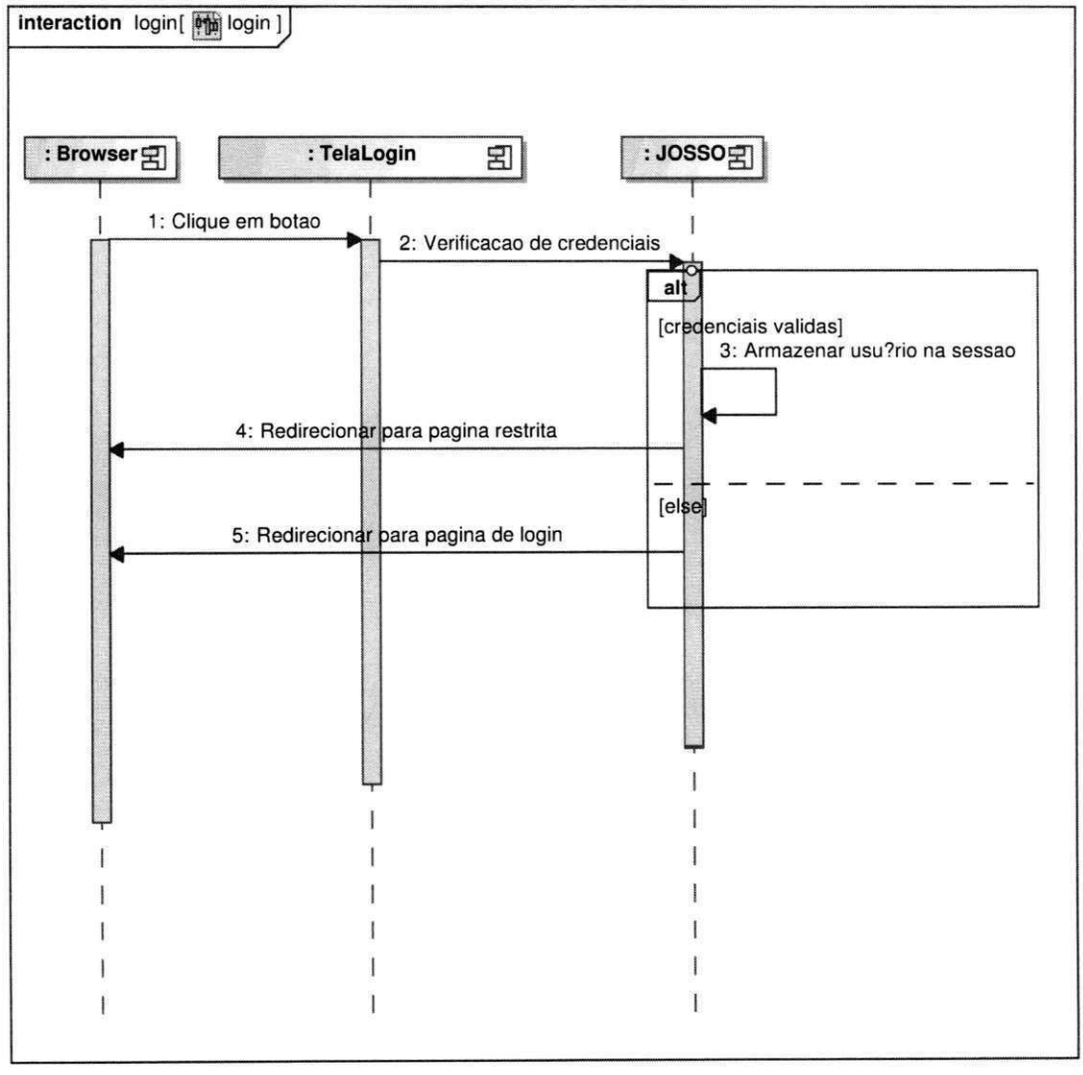
   Se as credenciais forem válidas:

3. O usuário é armazenado na sessão;
4. A página restrita é exibida no *browser*;

   Caso contrário:

5. A página de autenticação é exibida no *browser*.

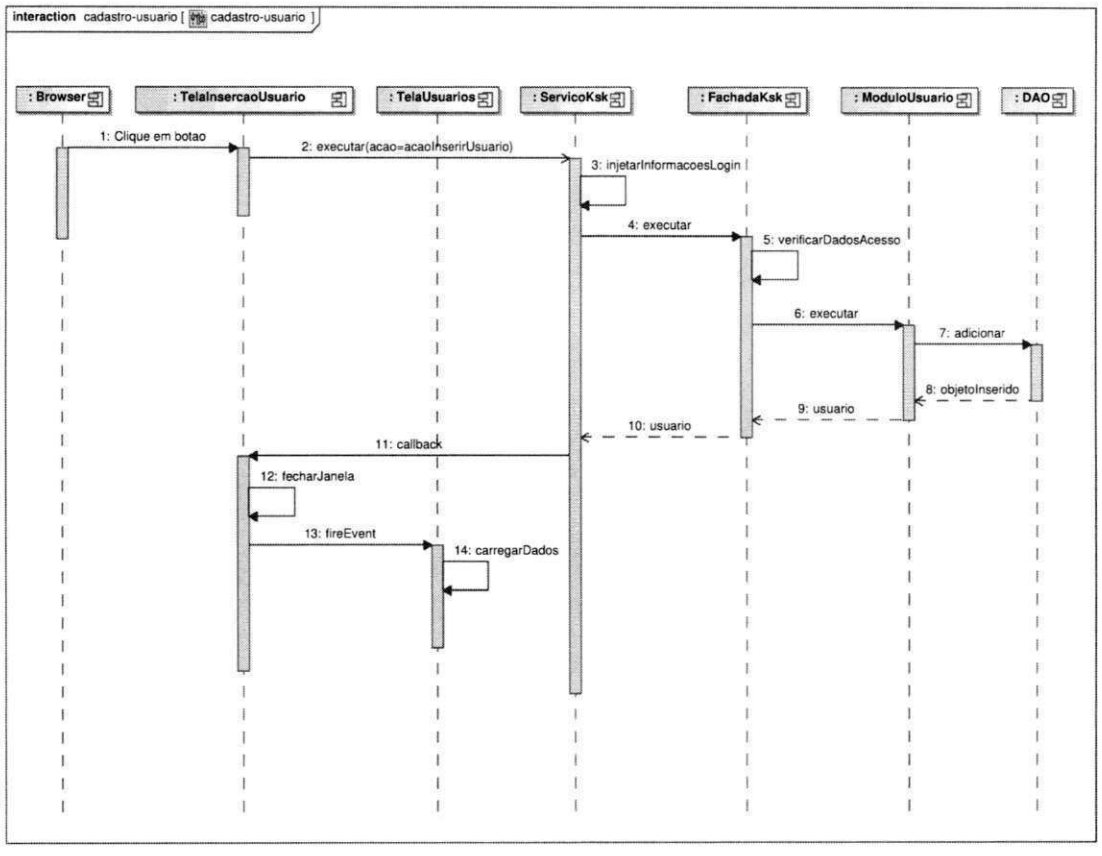
Figura C.5: Diagrama de sequência para o cenário "Login"



Cenário: Cadastro de um usuário Ao abrir a tela de usuários, deve-se clicar no botão para inserção de um novo usuário. Após informar todos os dados de cadastro, as seguintes etapas ocorrem:

1. O usuário clica no botão de salvar;
2. Uma chamada assíncrona a partir do *browser* do cliente, utilizando AJAX, é feita à *façade* GWT, `ServicoKsk`;

Figura C.6: Diagrama de sequência para o cenário "Cadastro de um usuário"



3. As informações de login são armazenadas na ação, `acaoInserirUsuario`, a ser realizada;

4. A *façade* `FachadaKsk` é acessada, recebendo a `acaoInserirUsuario`;

5. Os dados de acesso são verificados. Essa etapa envolve a checagem de um usuário autenticado e se o mesmo possui as permissões necessárias para executar tal ação;

6. A ação é repassada para a lógica responsável por ela e é executada;

7. O usuário a ser cadastrado é armazenado no SGBD;

8. O objeto inserido é retornado para a lógica responsável;

9. O usuário inserido é retornado para a *façade* `FachadaKsk`;

10. O usuário inserido é retornado para a *façade* GWT `ServicoKsk`;

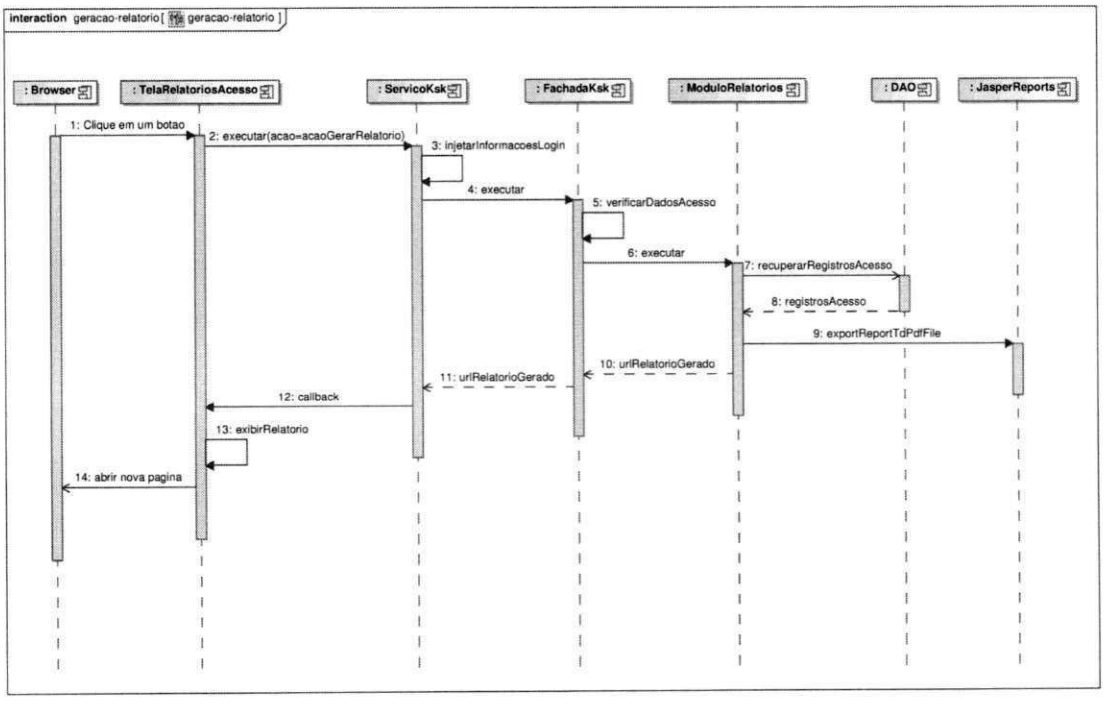
11. A *façade* GWT faz uma chamada à *callback* localizada na tela de inserção de usuários no *browser* do cliente. A *callback* receberá a informação de sucesso ou não da ação executada;

12. A tela de inserção de usuários é fechada se a ação foi executada com sucesso;

13. Um evento de atualização de dados é enviado para a tela de usuários;

14. A tela de usuários é atualizada contemplando o usuário que foi inserido.

Figura C.7: Diagrama de sequência para o cenário "Geração de um relatório"



Cenário: Geração de um relatório Ao abrir a tela de geração de relatórios, deve-se informar o período desejado para o relatório. Eis as etapas que ocorrem:

1. O usuário clica no botão para a geração do relatório;

2. Uma chamada assíncrona a partir do *browser* do cliente, utilizando AJAX, é feita à *façade* GWT, `ServicoKsk`;

3. As informações de login são armazenadas na ação, `acaoGerarRelatorio`, a ser realizada;

4. A *façade* `FachadaKsk` é acessada, recebendo a `acaoGerarRelatorio`;

5. Os dados de acesso são verificados. Essa etapa envolve a checagem de um usuário autenticado e se o mesmo possui as permissões necessárias para executar tal ação;

6. A ação é repassada para a lógica responsável por ela e é executada;

7. O componente `DAO` é acessado para que os registros de acesso sejam recuperados do SGBD;

8. Os registros de acesso são retornados para a lógica responsável;

9. O componente `JasperReports` gera o relatório em PDF;

10. A URL de acesso ao relatório gerado é retornada para a *façade* `FachadaKsk`;

11. A URL de acesso ao relatório gerado é retornada para a *façade* GWT `ServicoKsk`;

12. A *façade* GWT faz uma chamada à *callback* localizada na tela de geração de relatórios no *browser* do cliente. A *callback* receberá a informação de sucesso ou não da ação executada;

13. A operação para exibir relatório é chamada;

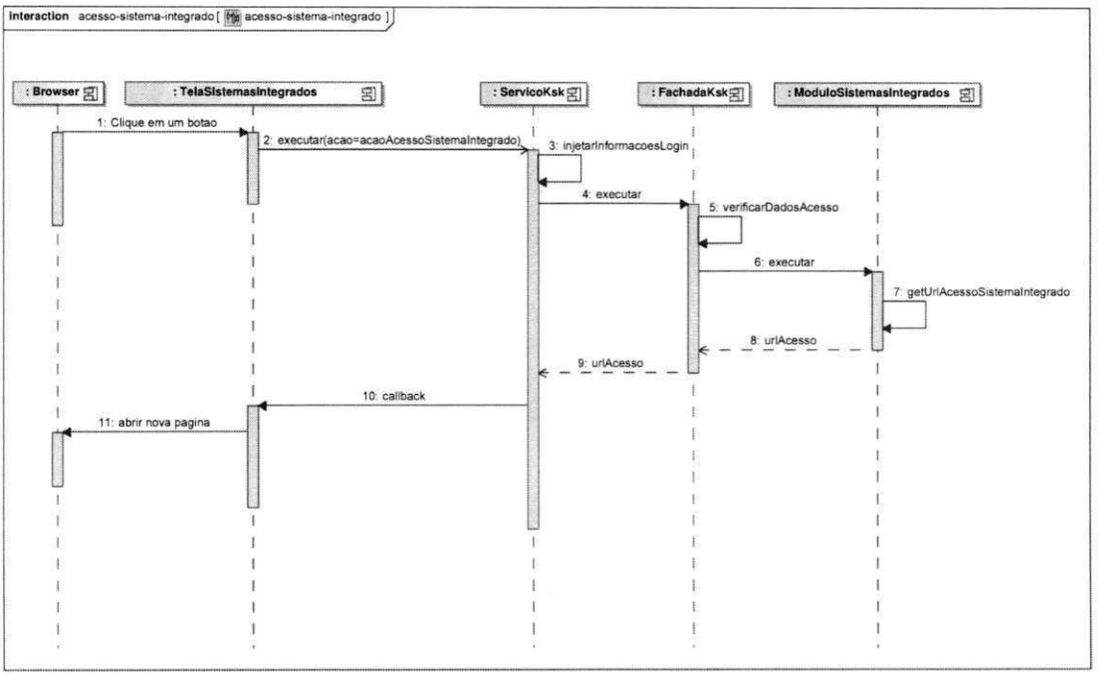
14. Tal operação causa a abertura de uma nova página com o relatório no *browser*.

Cenário: Acesso a um sistema integrado Ao se autenticar no sistema, a tela de sistemas integrados é exibida. Eis as etapas que ocorrem:

1. O usuário clica no botão para acessar um sistema;

2. Uma chamada assíncrona a partir do *browser* do cliente, utilizando AJAX, é feita à *façade* GWT, `ServicoKsk`;

Figura C.8: Diagrama de sequência para o cenário "Acesso a um sistema integrado"



3. As informações de login são armazenadas na ação, `acaoAcessoSistemaIntegrado`, a ser realizada;

4. A *façade* `FachadaKsk` é acessada, recebendo a `acaoAcessoSistemaIntegrado`;.

5. Os dados de acesso são verificados. Essa etapa envolve a checagem de um usuário autenticado e se o mesmo possui as permissões necessárias para executar tal ação;

6. A ação é repassada para a lógica responsável por ela e é executada;

7. A operação para recuperar a URL de acesso ao sistema integrado é chamada;

8. A URL de acesso é retornada para a *façade* `FachadaKsk`;

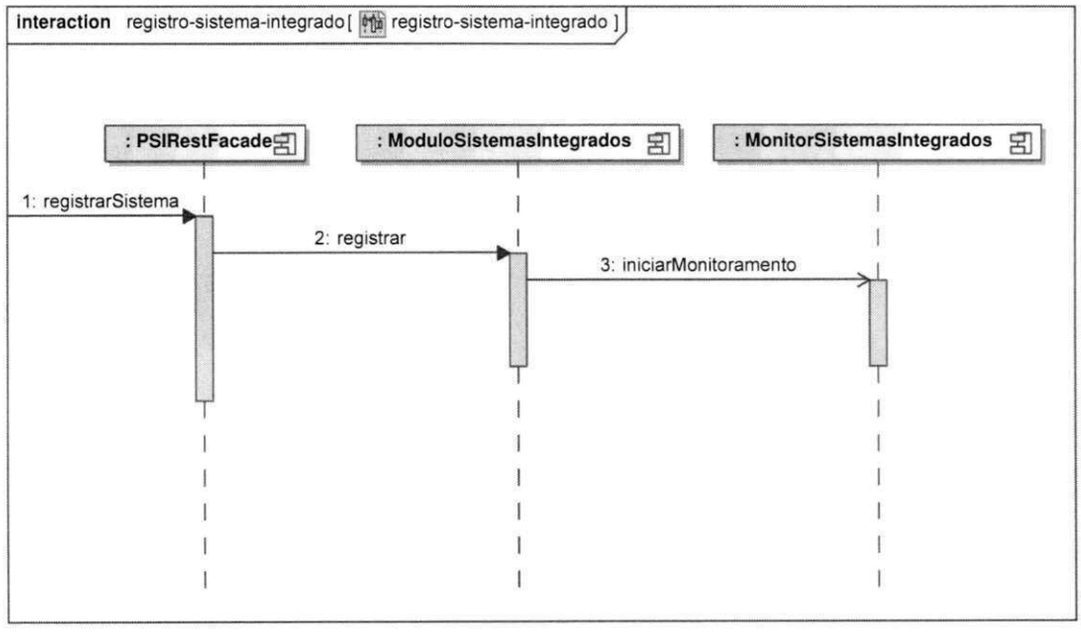
9. A URL de acesso é retornada para a *façade* GWT `ServicoKsk`.

10. A *façade* GWT faz uma chamada à *callback* localizada na tela de sistemas integrados no *browser* do cliente. A *callback* receberá a informação de sucesso ou não da ação

executada.

11. Uma nova página com o sistema integrado é aberta no *browser*.

Figura C.9: Diagrama de sequência para o cenário "Registro de um sistema integrado"



Cenário: Registro de um sistema integrado Assim que um sistema integrado é iniciado, as seguintes etapas ocorrem:

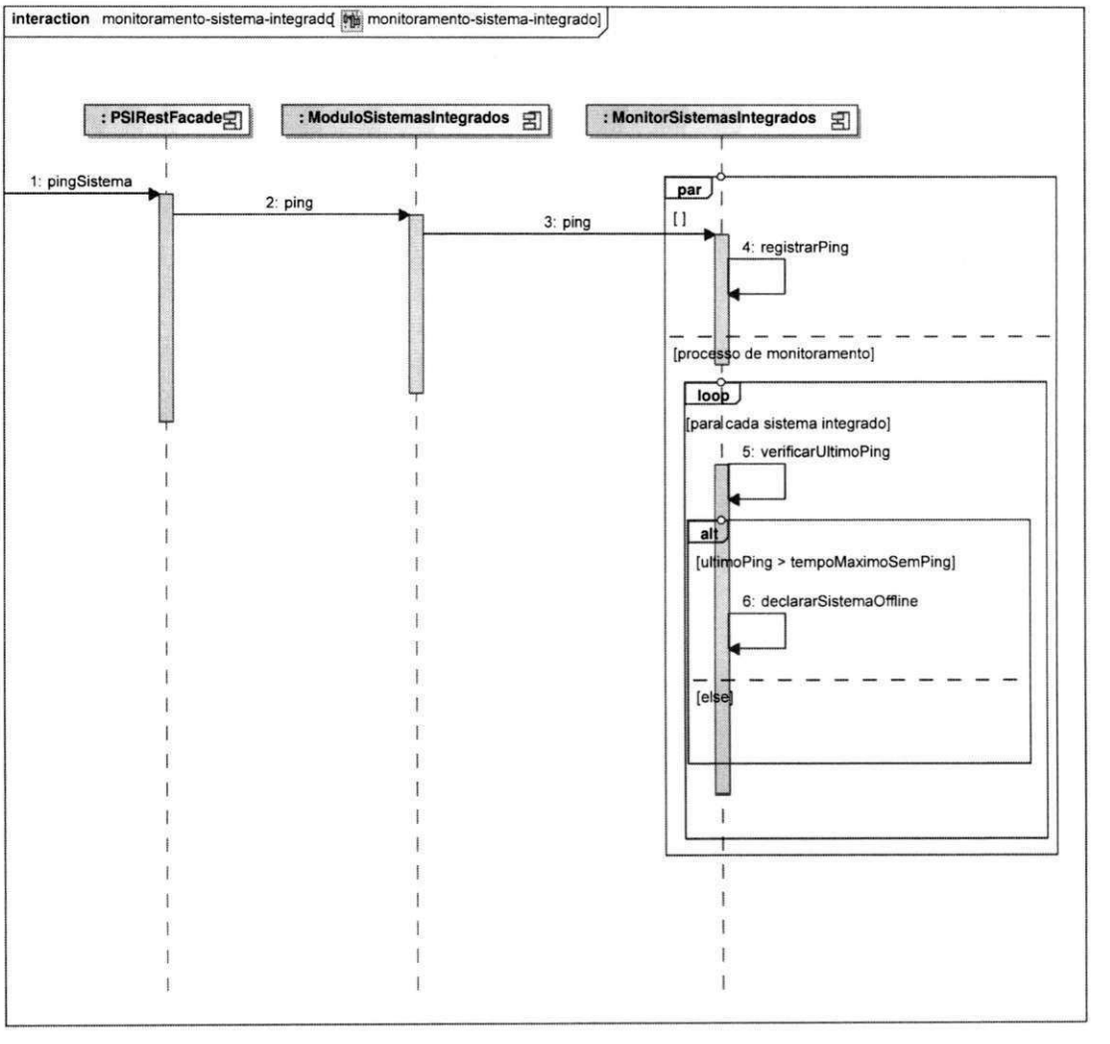
1. A *façade* REST do PSI, `PSIRestFacade`, é acessada via requisição HTTP e a operação `registrarSistema` é chamada;

2. A lógica de sistemas integrados é acessada e a operação registrar é chamada. Um novo sistema integrado é adicionado ao PSI;

3. O monitoramento do novo sistema integrado é iniciado.

Cenário: Monitoramento dos sistemas integrados Cada sistema integrado realiza uma operação de ping para informar sua disponibilidade ao PSI. Além disso, existe um processo

Figura C.10: Diagrama de sequência para o cenário "Monitoramento dos sistemas integrados"



em execução que checa se existe algum sistema integrado não disponível. Eis as etapas que ocorrem:

Thread* 1 - *thread iniciado a partir da requisição de um sistema integrado:

1. Um sistema integrado acessa a *façade* REST do PSI via requisição HTTP acessando a operação `pingSistema`;
2. A *façade* REST acessa a operação `ping` do componente

`ModuloSistemasIntegrados`;

3. O componente `ModuloSistemasIntegrados` acessa a operação `ping` do componente `MonitorSistemasIntegrados`;

4. A operação `registrarPing` atualiza o horário do último `ping` recebido pelo sistema integrado em questão;

***Thread* 2** - *thread* de monitoramento, em execução desde o início do PSI:

Para cada sistema integrado:

5. A operação `verificarUltimoPing` recupera o último ping enviado pelo sistema integrado;

6. Se o último *ping* enviado for maior que o tempo máximo permitido, ele é declarado como fora do ar.

Interfaces

Segue abaixo a lista de todas as interfaces a serem implementadas pelo sistema.

Nome:	`IAcao`
Descrição:	Qualquer requisição à *façade* do sistema é feita utilizando uma ação.
Serviços providos:	`getChave`: retorna o identificador da ação; `getChaveLogica`: retorna o identificador da lógica responsável por essa ação.

Nome:	`IDAO`
Descrição:	Provê várias operações para a persistência de dados.
Serviços providos:	`adicionar`: Adiciona um novo objeto na fonte de dados; `atualizar`: Atualiza um objeto existente na fonte de dados; `remover`: Remove um objeto da fonte de dados; `getObjetos`: Recupera um conjunto de objetos de acordo com um filtro de busca; `getNumeroObjetos`: Recuper o número de objetos de acordo com um filtro de busca.

Nome:	`IFachadaKsk`
Descrição:	*Façade* de acesso às lógicas do sistema.
Serviços providos:	`executar`: Executa uma ação através do repasse à lógica responsável; `verificarDadosAcesso`: Verifica os dados de acesso contidos na ação a ser executada, envolve a checagem da autenticação; permissões de acesso; etc; `registrarAcesso`: Registra o acesso para qualquer ação realizada.

Nome:	`IRegistroAcesso`
Descrição:	Provê informações sobre uma ação executada.
Serviços providos:	`getLogin`: Retorna o login do usuário que realizou a ação; `getChaveAcao`: Retorna o identificador da ação realizada; `getIp`: Retorna o IP de acesso ao sistema; `getDescricao`: Retorna uma descrição detalhada da ação realizada.

Nome:	`ILogica`
Descrição:	Representa uma lógica para um certo contexto.
Serviços providos:	`getChave`: Retorna o identficador da lógica; `executar`: Executa uma ação.

Nome:	`IDadosMonitoramento`
Descrição:	Provê informações sobre o estado do sistema em execução.
Serviços providos:	`getMemoriaConsumida`: Memória consumida na JVM; `getMemoriaTotal`: Memória total da JVM; `getQuantidadeThreads`: Quantidade de *threads* em execução; `getTempoMedioUltimasRequisicoes`: Tempo médio de resposta das últimas 100 requisições; `getEspacoLivreDisco`: Quantidade de espaço livre em disco. Retorna um valor em *bytes*.

Nome:	`IPropriedades`
Descrição:	Provê as operações de gerenciamento de propriedades.
Serviços providos:	`getValorPropriedade`: Retorna o valor para uma certa propriedade; `setValorPropriedade`: Atualiza o valor para uma certa propriedade.

Nome:	`IAutenticador`
Descrição:	Provê as operações para autenticação dos usuários.
Serviços providos:	`autenticar`: Recebe os dados de login e senha e retorna o usuário autenticado. Lança uma exceção caso os dados sejam inválidos.

Nome:	`IServicoKsk`
Descrição:	*Façade* acessada pelas requisições HTTP do GWT.
Serviços providos:	`executar`: Executa uma ação informada pelo usuário; `injetarInformacoesLogin`: Atualiza a ação informada pelo usuário com os dados de autenticação.

Nome:	`ILogicaLogin`
Descrição:	Lógica responsável pelas operações de login.
Serviços providos:	`executar`: Recebe uma ação e encaminha para a operação responsável; `autenticar`: Recebe os dados de login e senha e retorna o usuário autenticado. Lança uma exceção caso os dados sejam inválidos; `getUsuarioAutenticado`: Retorna o usuário autenticado; `alterarSenha`: Altera a senha de um usuário. Deve receber login, senha atual e a nova senha; `recuperarLogin`: Recupera o login de um usuário. Deve receber o e-mail do usuário e enviar o seu login para esse e-mail; `recuperarSenha`: Recupera a senha de um usuário. Uma nova senha deve ser gerada e enviada por e-mail.

Nome:	`ILogicaPapeis`
Descrição:	Lógica responsável pelas operações de gerenciamento de papéis.
Serviços providos:	`executar`: Recebe uma ação e encaminha para a operação responsável; `adicionar`: Adiciona um novo papel; `atualizar`: Atualiza um papel existente; `remover`: Remove um papel caso ele não seja utilizado por nenhum usuário; `getPapeis`: retorna um conjunto de papéis.

Nome:	`ILogicaRelatorios`
Descrição:	Lógica responsável pelas operações de geração de relatórios.
Serviços providos:	`executar`: Recebe uma ação e encaminha para a operação responsável; `recuperarRegistrosAcesso`: Retorna um conjunto de registros de acesso de acordo com o filtro de busca informado; `exportarRelatorio`: Exporta o relatório de acesso e retorna a URL de acesso.

Nome:	`IPSIRest`
Descrição:	*Façade* REST acessada por sistemas integrados.
Serviços providos:	`registrarSistema`: Adiciona um novo sistema à lista de sistemas integrados e começa o seu monitoramento; `pingSistema`: Atualiza o último tempo de ping enviado pelo sistema informado.

Nome:	`ILogicaServidorPublico`
Descrição:	Lógica responsável pelas operações de gerenciamento de servidores públicos.
Serviços providos:	`executar`: Recebe uma ação e encaminha para a operação responsável; `adicionar`: Adiciona um novo servidor público; `atualizar`: Atualiza um servidor público existente; `getServidoresPublicos`: retorna um conjunto de servidores públicos; `importar`: Importa os servidores públicos.

Nome:	`ILogicaSistemasIntegrados`
Descrição:	Lógica responsável pelas operações de gerenciamento de sistemas integrados.
Serviços providos:	`executar`: Recebe uma ação e encaminha para a operação responsável; `getSistemasIntegrados`: retorna um conjunto de sistemas integrados; `getUrlAcessoSistemaIntegrado`: retorna a URL de acesso para um certo sistema integrado; `verificarUltimoPing`: verifica o último ping de um certo sistema; `declararSistemaOffline`: altera o status de um sistema integrado para indisponível; `iniciarMonitoramento`: inicia o monitoramento de um sistema integrado.

Nome:	`IPSIEasyAcceptFacade`
Descrição:	Representa a *façade* que será acessada para os testes de aceitação.
Serviços providos:	Todos os serviços providos por `FachadaKsk` com valores de retorno e parâmetros alterados para se adaptar à solução de testes de aceitação.

Nome:	`ILogicaUsuario`
Descrição:	Lógica responsável pelas operações de gerenciamento de usuários.
Serviços providos:	`executar`: Recebe uma ação e encaminha para a operação responsável; `adicionar`: Adiciona um novo usuário; `atualizar`: Atualiza um usuário existente; `desativar`: Desativa um usuário impedindo que o mesmo se autentique no PSI; `getUsuarios`: retorna um conjunto de usuários.

Nome:	`IPapel`
Descrição:	Representa um papel para um certo sistema. É composto por um conjunto de permissões.
Serviços providos:	`getSistema`: retorna o sistema associado; `getIdentificador`: retorna o identificador do papel; `getDescricao`: retorna a descrição do papel; `getPermissoes`: retorna o conjunto de permissões associadas.

Nome:	`IPermissao`
Descrição:	Representa uma permissão para um certo sistema.
Serviços providos:	`getSistema`: retorna o sistema associado; `getIdentificador`: retorna o identificador da permissão; `getDescricao`: retorna a descrição da permissão.

Nome:	`IServidorPublico`
Descrição:	Representa um servidor público da instituição.
Serviços providos:	`getNome`: Retorna o nome do servidor; `getSetor`: Retorna o setor do servidor; `getCargo`: Retorna o cargo do servidor.

Nome:	`ISistemaIntegrado`
Descrição:	Representa um sistema integrado ao PSI.
Serviços providos:	`getIdentificador`: Retorna o identificador do sistema; `getDescricao`: Retorna a descrição do sistema; `getPermissoes`: Retorna as permissões utilizadas pelo sistema.

Nome:	`IUsuario`
Descrição:	Representa um usuário do PSI.
Serviços providos:	`getServidorPublico`: Retorna o servidor público associado; `getLogin`: Retorna o login do usuário; `getSenha`: Retorna a senha do usuário de forma criptografada; `getSistemasAcessiveis`: Retorna os sistemas acessíveis pelo usuário: `getPapeis`: Retorna os papéis do usuário em um certo sistema.

C.5.2 Visão de Desenvolvimento

Nesta seção serão apresentados diagramas com os pacotes a serem utilizados pelos desenvolvedores.

Modelos

Pacotes para a `Ksk`

- A `Ksk` deve ser empacotada em um arquivo de formato JAR.

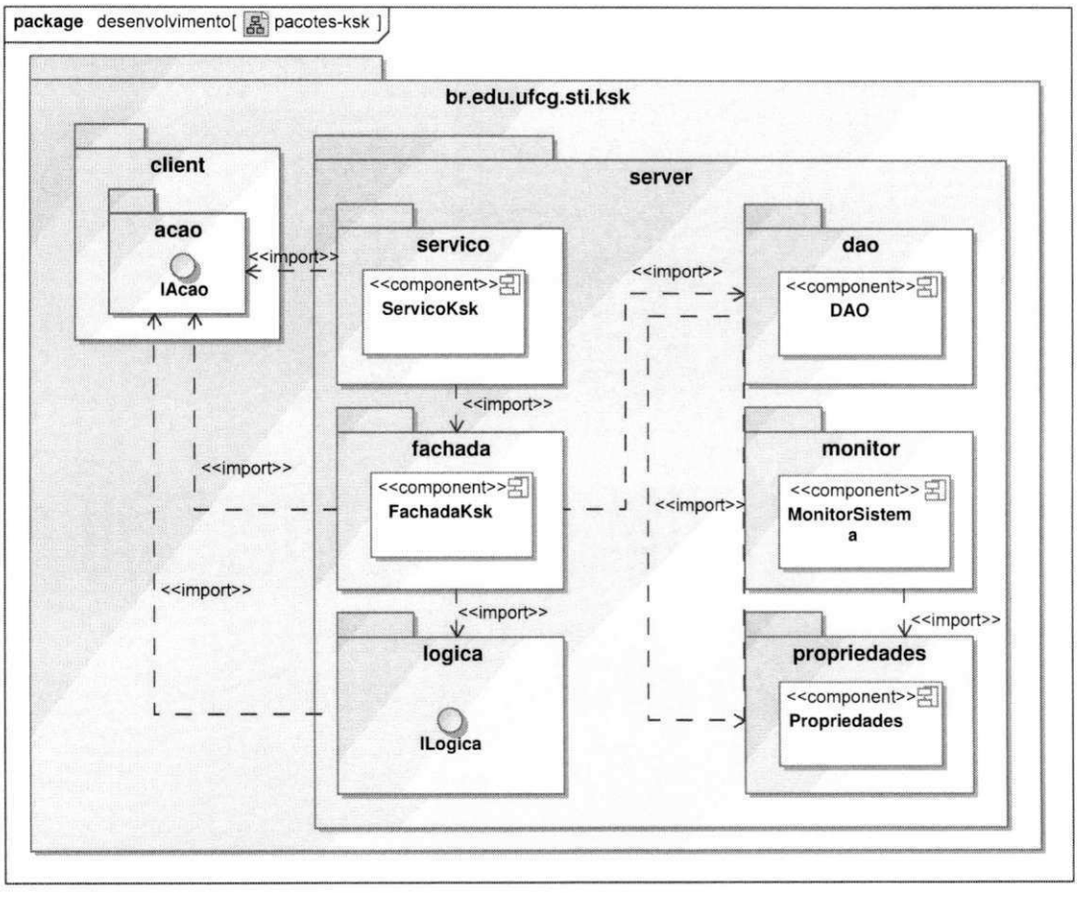
Pacotes para o PSI

- O PSI deve ser empacotado em um arquivo de formato WAR.

C.5.3 Visão de Processos

O funcionamento do PSI depende da execução dos seguintes processos:

Figura C.11: Diagrama de pacotes da `Ksk`



- **Balanceadores de carga**: Cada requisição HTTP do usuário deve passar inicialmente por um dos balanceadores de carga existentes. Mais detalhes podem ser vistos em C.5.3;

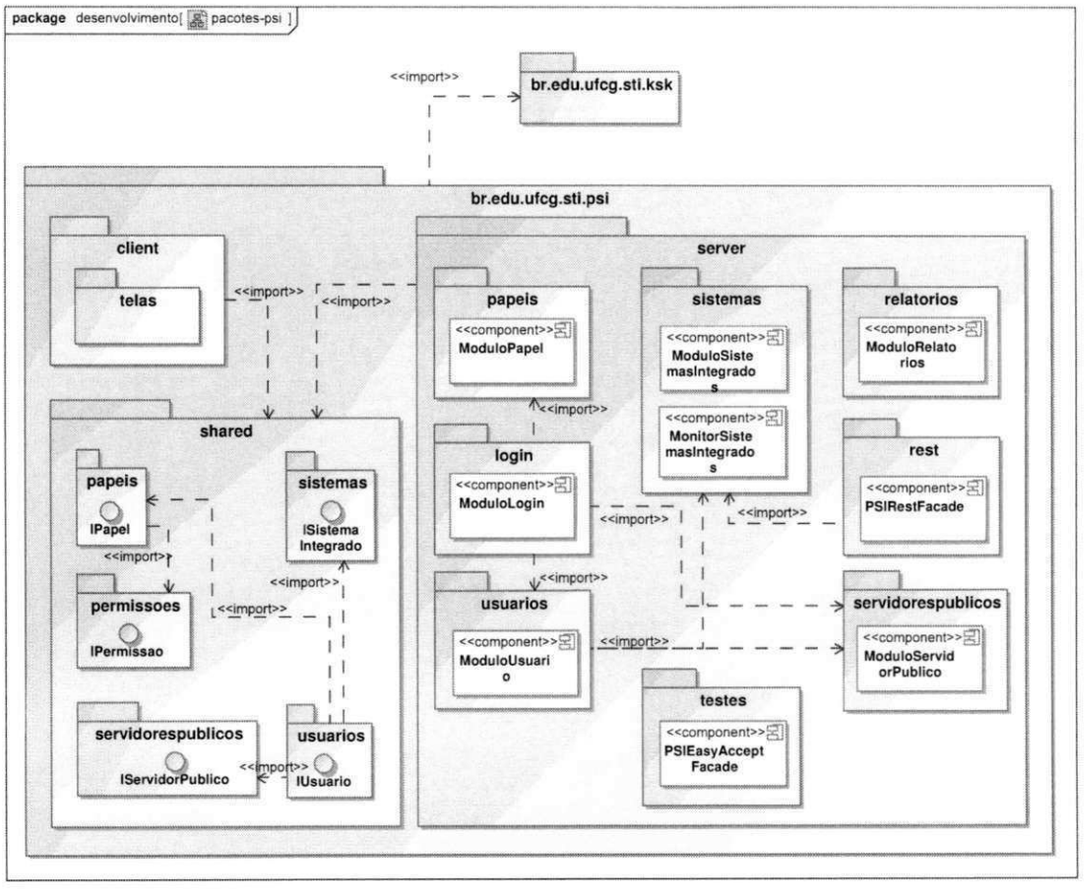
- **Servidores de aplicação**: cada servidor de aplicação deverá executar um JOSSO Agent, pelo menos um sistema integrado ou uma instância do PSI;

- **PSI**: para evitar problemas de disponibilidade e escalabilidade, deverá existir pelo menos duas instâncias do PSI em execução em dois servidores de aplicação diferentes;

  - **Sistemas integrados**: Dependendo do número de acessos e da carga gerada por cada sistema integrado, a quantidade de instâncias em execução deve variar;

  - **JOSSO Gateway**: Responsável pelo acesso aos dados de autenticação e pelo ar-

Figura C.12: Diagrama de pacotes do PSI



mazenamento das sessões existentes. Deve existir somente **um** JOSSO Gateway instalado em **um** servidor de aplicação. Inicialmente não será possível o uso do JOSSO Gateway em cluster por limitações da versão gratuita;

- **JOSSO Agent**: Responsável pela comunicação com o JOSSO Gateway para a verificação de uma sessão já existente. Todos os servidores de aplicação devem conter uma instalação do JOSSO Agent para que a autenticação do tipo SSO funcione;

- **Sistemas de gerenciamento de bancos de dados**: Deve existir pelo menos duas instâncias do SGBD em execução para garantir tempos de resposta aceitáveis para as aplicações. Caso a replicação do SGBD seja complexa, pode-se optar pelo armazenamento de diferentes bancos em diversas instâncias, sem que haja replicação entre

eles;

- **Servidores de diretório LDAP**: Deve haver pelo menos uma instância do servidor LDAP para que sistemas de terceiros utilizem os mesmos dados de autenticação do PSI.

Modelos

Requisição de um usuário

- Uma requisição é feita;
- É verificado se a requisição possui uma sessão associada a ela;
- **Se existir:**
  - A requisição do usuário é encaminhada para o servidor de aplicação utilizado anteriormente;
- **Se não existir:**
  - Um servidor de aplicação para tratar a requisição é escolhido;
  - O servidor de aplicação escolhido é armazenado para tratar futuras requisições;
- A requisição do usuário é atendida;
- A resposta à requisição é retornada ao usuário.

C.5.4 Visão Física

Segue abaixo a relação entre nodos de hardware e processos em execução e as capacidades necessárias de processamento, memória RAM e disco rígido.

Nodos de hardware	**Processos em execução**	**Capacidade de processamento/memória/disco**
Servidores para balanceamento	Balanceador de carga	Alta/Média/Baixa

Servidores do PSI	PSI, JOSSO Agent	Alta/Alta/Baixa
Servidores de sistemas integrados	Sistemas integrados, JOSSO Agent	Alta/Alta/Baixa
Servidores de autenticação	JOSSO Gateway, sistemas de diretório LDAP	Média/Baixa/Baixa
Servidores de bancos de dados	SGBD	Alta/Média/Alta
Servidores para sistemas de terceiros	Sistemas de terceiros	Alta/Alta/Média
Máquinas existentes	Uma máquina cliente para o *browser* de cada usuário.	Baixa/Baixa/Baixa

Figura C.13: Diagrama de atividades para uma requisição de um usuário qualquer

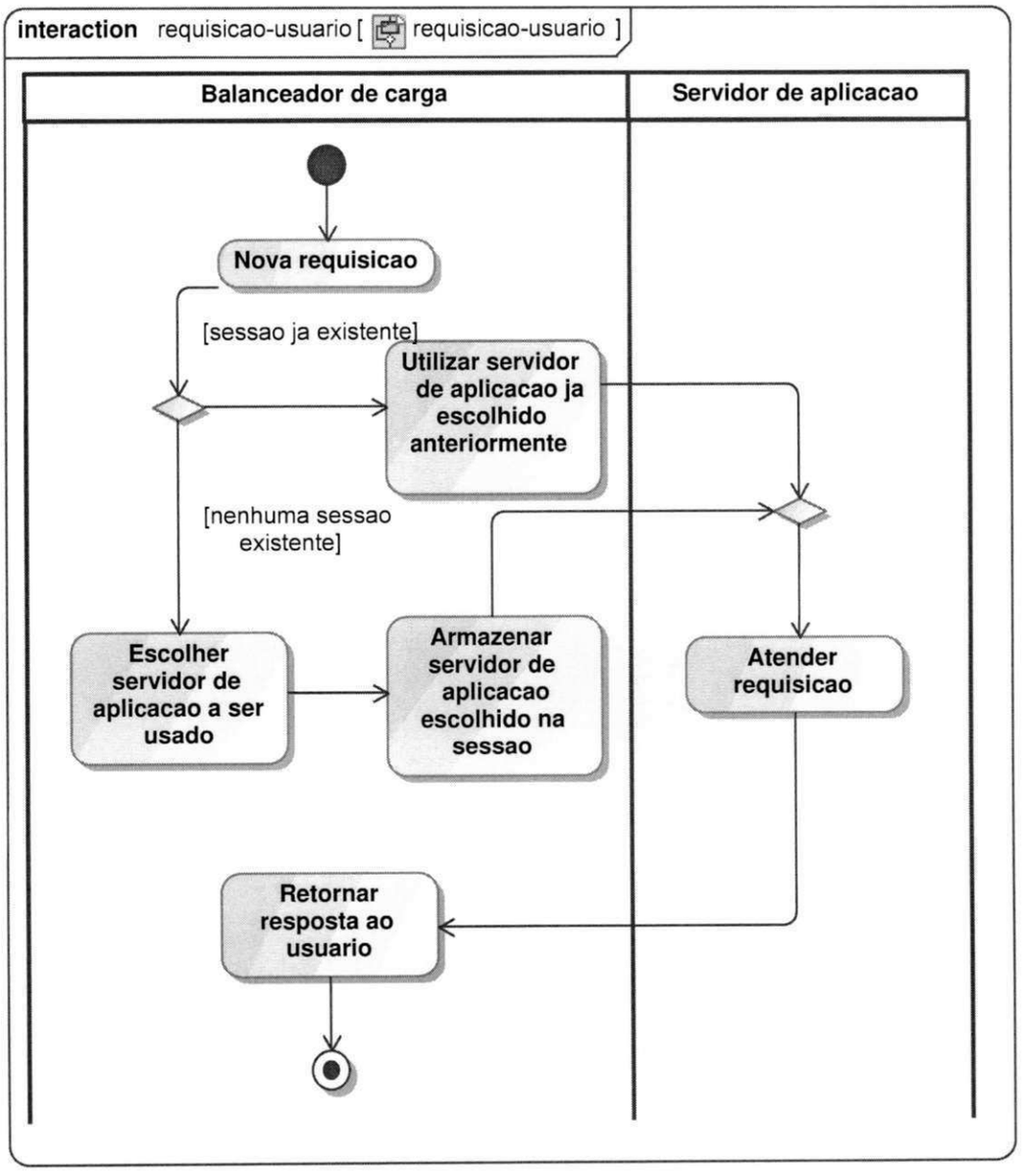
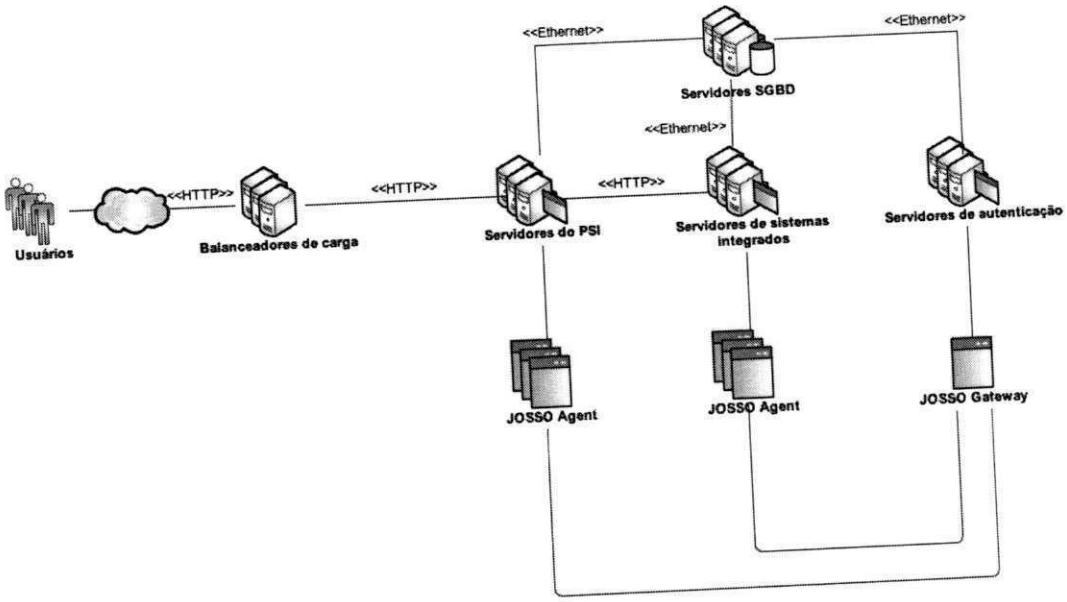


Figura C.14: Esquema resumido da visão física



Apêndice D

Relatório de Validação Arquitetural do PSI

Universidade Federal de Campina Grande

Secretaria de Planejamento e Orçamento

Serviço da Tecnologia da Informação

Portal de Sistemas Integrados - PSI

Relatório de Validação Arquitetural

Campina Grande, Paraíba, Brasil

Março, 2013

Revisões

Autores	Descrição	Versão	Data
Leandro José	Versão inicial do documento de validação arquitetural do Portal de Sistemas Integrados.	1.0	12/03/2013

D.1 Sistema de Interesse

A UFCG é composta cerca de milhares de servidores e alunos, mas poucos são os sistemas desenvolvidos e implantados para auxiliar os diversos processos existentes na instituição.

Os sistemas existentes sofrem de problemas como:

- Falta de um processo de desenvolvimento definido;
- Falta de um conjunto padrão de tecnologias utilizadas;
- Falta de um padrão de nomenclatura para bancos de dados;
- Falta de um arcabouço para o desenvolvimento de sistemas;
- Falta de um ponto único para acesso aos sistemas da instituição.

Os três primeiros problemas citados foram resolvidos para os sistemas desenvolvidos a partir de 2011, o penúltimo foi resolvido parcialmente e o último problema ainda existe. Com isso, o objetivo do PSI é fornecer um arcabouço para futuros sistemas a serem desenvolvidos e um meio de acesso a todos os sistemas da instituição.

O arcabouço será um conjunto de bibliotecas de software que serão utilizadas na produção de novos sistemas institucionais. O ponto único de acesso será um sistema *web* que permitirá o uso de diversos sistemas pelos alunos e servidores da instituição, de forma que a autenticação seja feita somente uma vez.

D.2 Requisitos Arquiteturais

Seguem abaixo os requisitos funcionais e não-funcionais que foram classificados como **arquiteturais**. Mais detalhes sobre os mesmos podem ser encontrados na seção C.3.2.

- Acessar um sistema;
- Monitorar sistemas integrados;
- Usuários simultâneos;
- Sistema portável;

- Interface *web*;
- Operações registradas;
- Testes automáticos;
- Acesso restrito;
- Login único;
- Várias instâncias em execução;
- API para integração;
- Fornecer dados de autenticação para outros sistemas;
- Comunicação segura;
- Permissões por sistema;
- Papéis por sistema.

D.3 Decisões-chave

Seguem abaixo os identificadores das decisões-chave associadas aos requisitos arquiteturais. As decisoões detalhadas podem ser lidas na seção C.5.1.

- Utilizar *Java Open Single Sign On* (JOSSO);
- Utilizar login definido pelo servidor público;
- Configurar replicação no SGBD;
- Utilizar um SGBD para armazenar dados;
- Utilizar uma implementação da *Java Persistence API* (JPA);
- Criar múltiplos usuários para casos em que o servidor público possui dois cargos na mesma instituição;

- Criar múltiplos usuários para casos em que o servidor público possui dois cargos na mesma instituição;
- Definir padrão seguro para senhas **(informação adicionada após iteração 01 da validação)**;
- Recusar senhas já utilizadas;
- Todo usuário deve possuir um e-mail institucional;
- Utilizar LDAP para fornecer informações de autenticação para sistemas de terceiros;
- Sistemas integrados podem estar implantados em diferentes servidores de aplicação;
- Bloquear acesso de usuários desativados **(informação adicionada após iteração 01 da validação)**;
- Exigir permissões de administrador para gerenciar usuários e papéis;
- Utilizar *JasperReports* na geração de relatórios;
- Registrar tempos de resposta para qualquer acesso à *façade*;
- Armazenar dados de monitoramento dos sistemas integrados **(informação adicionada após iteração 01 da validação)**;
- Realizar testes de carga;
- Utilizar balanceadores de carga;
- Utilizar *caches* distribuídos quando necessário;
- Utilizar Java;
- O sistema deve permitir implantação no servidor *Apache Tomcat*;
- Utilizar *Google Web Toolkit* (GWT);
- Utilizar uma *façade* para acessar a lógica de negócio;
- Usar o *EasyAccept* para testes de aceitação;

- Usar o *Selenium* para testes da interface gráfica;
- Registrar qualquer acesso à *façade*;
- Adaptar GWT para integrar com a biblioteca JOSSO;
- Criar *scripts* para backups dos bancos de dados do SGBD;
- Criar interfaces REST e bibliotecas Java para sistemas integrados;
- Utilizar *RestEasy* **(informação adicionada após iteração 01 da validação)**;
- Utilizar SSL.

D.4 Validação

Nessa seção se encontram informações sobre a forma de validação de cada requisito arquitetural e seus resultados detalhados. Inicialmente, segue abaixo um resumo dos resultados obtidos na validação:

Requisito Arquitetural	**Técnicas**	**Resultados**
RF - Monitorar sistemas integrados	Cenários	**Iteração 01:** • $m = 0$; • Nenhum cenário foi executado com sucesso; • **Situação:** falha. **Iteração 02:** • $m = 1, m \geq 0,5$; • Todos os cenários foram executados com sucesso; • **Situação:** OK.

RNF - Usuários simultâneos RNF - Várias instâncias em execução	Simulações	**Iteração 01:** • $t \leq 15$ segundos para uma taxa de entrada variando entre 1 e 19 sessões por segundo; • Cada sessão teve uma duração média de 78 segundos, logo, a exigência de 200 sessões simultâneas foi atingida a partir da taxa de uma entrada de 3 sessões por segundo; • Não é necessário que múltiplas instâncias sejam executadas para manter o desempenho desejável; • **Situação:** OK.
RNF - Sistema portável	*Checklists*	**Iteração 01:** • $m = 1, m \geq 0,3$; • **Situação:** OK.

RNF - Acesso restrito	Cenários	**Iteração 01:** • $m = 0,\overline{6}, m < 0,8$; • Falha na execução do cenário "Acessar um recurso com um usuário desativado"; • **Situação:** falha. **Iteração 02:** • $m = 1, m \geq 0,8$; • Todos os cenários foram executados com sucesso; • **Situação:** OK.
RNF - Login único	*Checklists*	**Iteração 01:** • $m = 0,8, m \geq 0,5$; • **Situação:** OK. **Iteração 02:** • Não houve alterações nos resultados.

RNF - API para integração	*Checklists*	**Iteração 01:** • $m = 0,125, m < 0,8$; • **Situação:** falha. **Iteração 02:** • $m = 0,625, m < 0,8$; • **Situação:** falha.
RNF - Fornecer dados de autenticação para outros sistemas	*Checklists*	**Iteração 01:** • $m = 0,\overline{3}, m \geq 0,3$; • **Situação:** OK. **Iteração 02:** • Não houve alterações nos resultados.
RNF - Comunicação segura	*Checklists*	**Iteração 01:** • $m = 0,\overline{72}, m < 0,8$; • **Situação:** falha. **Iteração 02:** • $m = 0,\overline{81}, m \geq 0,8$; • **Situação:** OK.

As próximas seções descrevem com detalhes a validação de cada requisito arquitetural.

D.4.1 Monitorar Sistemas Integrados

Configuração

Métricas:	Percentual de sucesso com $k = 1$
Limites:	$m \geq 0,5$
Técnicas:	Cenários

Cenários

Nome:	Visualizar indisponibilidades de um sistema integrado
Partes interessadas:	Implantador
Descrição:	1. O usuário X se autentica no PSI; 2. X acessa a tela de monitoramento de sistemas integrados; 3. O sistema integrado Y é selecionado; 4. X clica no botão de exibição de indisponibilidades; 5. Um período para exibição é selecionado; 6. X clica no botão de atualização de resultados.

Nome:	Exibir sistemas integrados disponíveis em tempo real
Partes interessadas:	Implantador
Descrição:	1. O usuário X se autentica no PSI; 2. X acessa a tela de monitoramento de sistemas integrados; 3. O modo de monitoramento em tempo real é ativado; 4. Os dados são atualizados a cada minuto.

Nome:	Checar valores mínimos, médios e máximos para as métricas monitoradas de um sistema integrado
Partes interessadas:	Implantador
Descrição:	1. O usuário X se autentica no PSI; 2. X acessa a tela de monitoramento de sistemas integrados; 3. O sistema integrado Y é selecionado; 4. X clica no botão de estatísticas detalhadas; 5. Um período para exibição é selecionado. 6. X clica no botão de atualização de resultados.

Resultados

Iteração 01 Não foi possível executar nenhum teste por falta de informações na arquitetura, logo, $m = 0$, o que não satisfaz o limite mínimo desejado.

Nome:	Visualizar indisponibilidades de um sistema integrado
Execução:	1. OK - `ModuloLogin` responsável pelo lógica de login. `TelaLogin` responsável pela interface gráfica; 2. Falha - `ModuloSistemasIntegrados` responsável pela lógica de sistemas integrados. Não é informada tela para monitoramento de sistemas.
Situação:	Falha: não foi possível executar o teste por falta de informações na arquitetura.

Nome:	Exibir sistemas integrados disponíveis em tempo real
Execução:	1. OK - `ModuloLogin` responsável pelo lógica de login. `TelaLogin` responsável pela interface gráfica; 2. Falha - `ModuloSistemasIntegrados` responsável pela lógica de sistemas integrados. Não é informada tela para monitoramento de sistemas.
Situação:	Falha: não foi possível executar o teste por falta de informações na arquitetura.

Nome:	Checar valores mínimos, médios e máximos para as métricas monitoradas de um sistema integrado
Execução:	1. OK - ModuloLogin responsável pelo lógica de login. `TelaLogin` responsável pela interface gráfica; 2. Falha - `ModuloSistemasIntegrados` responsável pela lógica de sistemas integrados. Não é informada tela para monitoramento de sistemas.
Situação:	Falha: não foi possível executar o teste por falta de informações na arquitetura.

Iteração 02 Todos os testes foram executados corretamente, logo, $m = 1$, o que satisfaz o limite mínimo desejado.

Nome:	Visualizar indisponibilidades de um sistema integrado
Execução:	1. OK - `ModuloLogin` responsável pelo lógica de login. `TelaLogin` responsável pela interface gráfica; 2. OK - `ModuloSistemasIntegrados` responsável pela lógica de sistemas integrados. `TelaSistemasIntegrados` responsável pela interface gráfica; 3. OK - Uma listagem de sistemas acessíveis é exibida ao usuário; 4. OK - Na tela de monitoramento é possível visualizar as informações em tempo real ou em períodos anteriores; 5. OK - Os períodos anteriores são armazenados em bancos de dados; 6. OK - O componente responsável pela interface gráfica deve implementar esse comportamento.
Situação:	OK

Nome:	Exibir sistemas integrados disponíveis em tempo real
Execução:	1. OK - `ModuloLogin` responsável pelo lógica de login. `TelaLogin` responsável pela interface gráfica; 2. OK - `ModuloSistemasIntegrados` responsável pela lógica de sistemas integrados. `TelaSistemasIntegrados` responsável pela interface gráfica; 3. OK - Na tela de monitoramento é possível visualizar as informações em tempo real ou em períodos anteriores; 4. OK - O componente responsável pela interface gráfica deve implementar esse comportamento.
Situação:	OK

Nome:	Checar valores mínimos, médios e máximos para as métricas monitoradas de um sistema integrado
Execução:	1. OK - `ModuloLogin` responsável pelo lógica de login. `TelaLogin` responsável pela interface gráfica; 2. OK - `ModuloSistemasIntegrados` responsável pela lógica de sistemas integrados. `TelaSistemasIntegrados` responsável pela interface gráfica; 3. OK - Uma listagem de sistemas acessíveis é exibida ao usuário; 4. OK - Na tela de monitoramento é possível exibir informações estatísticas para períodos anteriores; 5. OK - Os períodos anteriores são armazenados em bancos de dados; 6. OK - O componente responsável pela interface gráfica deve implementar esse comportamento.
Situação:	OK

Conclusão

Após alterações na arquitetura, o limite mínimo foi atingindo na segunda iteração.

D.4.2 Usuários Simultâneos e Várias Instâncias em Execução

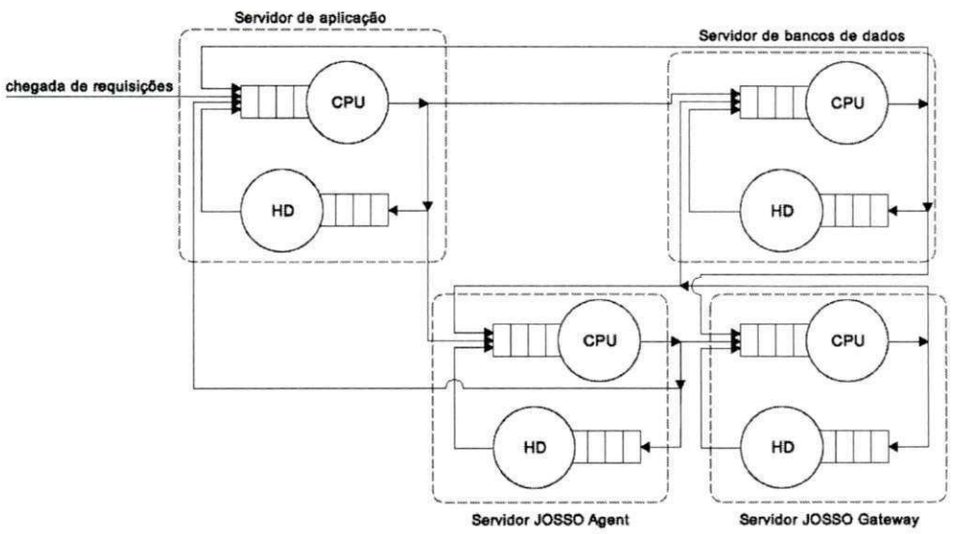
Configuração

Métricas:	Tempo de resposta
Limites:	$t \leq 15$ segundos
Técnicas:	Simulações

Simulador

Descrição Geral Um simulador será utilizado para validar os requisitos de desempenho definidos na arquitetura do PSI. O modelo em rede de filas abertas a ser utilizado pelo simulador pode ser visto na figura D.3.

Figura D.1: Modelo de filas para o simulador

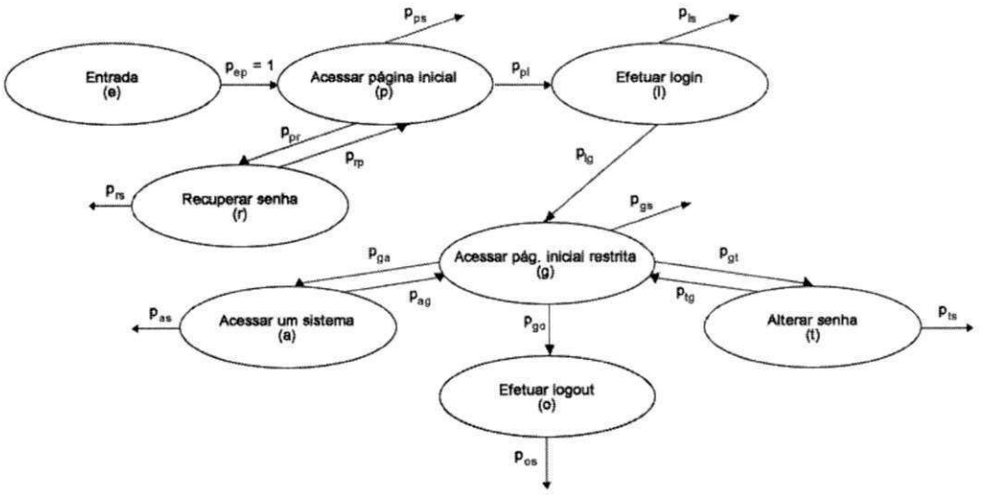


Algumas hipóteses foram elaboradas para guiar a construção do mesmo:

- O processo de chegada de fregueses segue uma distribuição de Poisson;
- Os tempos de serviço são distribuídos exponencialmente;
  - Essas distribuições serão utilizadas diante da falta de registros de acesso para determinar a distribuição correta e por serem bastante utilizadas para tais finalidades;
- Taxas de atraso causados pela rede não serão considerados;
  - A maioria das requisições serão feitas a partir da própria instituição, além disso, não existe nenhuma operação que demande um alto uso da rede;
- O *think time* dos fregueses são distribuídos exponencialmente.

Com as distribuições escolhidas e diante da simplicidade do modelo não seria necessário realizar simulações para conseguir determinar valores como tempo de resposta; vazão e tamanho da fila, mas um simulador foi construído a fim de demonstrar cada técnica possível na validação dos requisitos arquiteturais.

Figura D.2: Grafo que representa o comportamento dos fregueses



Entidades

Fregueses A carga de trabalho do sistema é caracterizada pela CBMG [63] exibida na figura D.2.

As transições entre os estados são rotulados com a probabilidade delas ocorrerem. Por exemplo, p_{pl} representa a probabilidade de um cliente acessar a página de login a partir da página inicial.

Servidores Cada servidor a seguir é composto de um processador (CPU) e um disco rígido (HD).

Tipo	Operações atendidas

Aplicação	Acessar página inicial (p) Recuperar senha (r) Efetuar login (l) Acessar página inicial restrita (g) Alterar senha (t) Acessar um sistema (a) Efetuar logout (o)
Bancos de dados	Efetuar login (l) Recuperar senha (r) Alterar senha (t)
JOSSO *Agent*	Efetuar login (l) Efetuar logout (o)
JOSSO *Gateway*	Efetuar login (l) Efetuar logout (o)

Tabela D.2: Operações atendidas por servidor

Casos de Teste O simulador será utilizado para gerar as seguintes métricas a partir das taxas de entrada variando de 1 a 30 sessões por segundo, com intervalos de $0,5$:

- Quantidade média de sessões;
- Quantidade máxima de sessões;
- Tempo médio por sessão (segundos);
- Utilização - Servidor de aplicação - CPU (%);
- Utilização - Servidor de aplicação - HD (%);
- Utilização - Servidor de bancos de dados - CPU (%);
- Utilização - Servidor de bancos de dados - HD (%);

- Utilização - Servidor JOSSO Agent - CPU (%);
- Utilização - Servidor JOSSO Agent - HD (%);
- Utilização - Servidor JOSSO Gateway - CPU (%);
- Utilização - Servidor JOSSO Gateway - HD (%);
- Tempo médio de resposta - Acessar página inicial (segundos);
- Tempo médio de resposta - Recuperar senha (segundos);
- Tempo médio de resposta - Efetuar login (segundos);
- Tempo médio de resposta - Acessar página inicial restrita (segundos);
- Tempo médio de resposta - Alterar senha (segundos);
- Tempo médio de resposta - Acessar um sistema (segundos);
- Tempo médio de resposta - Efetuar logout (segundos).

Dispositivo	**p**	**r**	**l**	**g**	**t**	**a**	**o**
Aplicação - CPU	200	75	50	150	100	200	125
Aplicação - HD	175	60	45	125	75	175	100
Bancos de dados - CPU	N/A	50	60	N/A	50	N/A	N/A
Bancos de dados - HD	N/A	30	40	N/A	30	N/A	N/A
JOSSO *Agent* - CPU	N/A	N/A	150	N/A	N/A	N/A	180
JOSSO *Agent* - HD	N/A	N/A	120	N/A	N/A	N/A	150
JOSSO *Gateway* - CPU	N/A	N/A	125	N/A	N/A	N/A	150
JOSSO *Gateway* - HD	N/A	N/A	100	N/A	N/A	N/A	120

Tabela D.3: Taxas de atendimento dos servidores

Operação	***Think time* médio (segundos)**

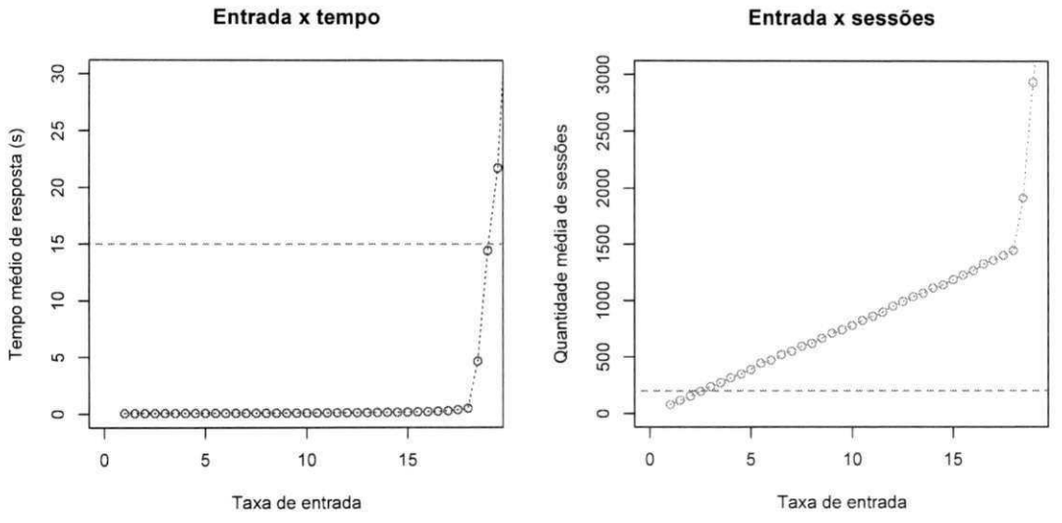
Acessar página inicial (p)	10,00
Recuperar senha (r)	25,00
Efetuar login (l)	40,00
Acessar página inicial restrita (g)	10,00
Alterar senha (t)	30,00
Acessar um sistema (a)	15,00
Efetuar logout (o)	0,00

Tabela D.4: *Think time* médio para cada operação

Estado	**e**	**p**	**r**	**l**	**g**	**t**	**a**	**o**	**s**
Entrada (e)	0,00	1,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Acessar página inicial (p)	0,00	0,00	0,20	0,70	0,00	0,00	0,00	0,00	0,10
Recuperar senha (r)	0,00	0,85	0,00	0,00	0,00	0,00	0,00	0,00	0,15
Efetuar login (l)	0,00	0,00	0,00	0,00	0,90	0,00	0,00	0,00	0,10
Acessar pág. inic. rest. (g)	0,00	0,00	0,00	0,00	0,00	0,10	0,80	0,05	0,05
Alterar senha (t)	0,00	0,00	0,00	0,00	0,80	0,00	0,00	0,00	0,20
Acessar um sistema (a)	0,00	0,00	0,00	0,00	0,60	0,00	0,00	0,00	0,40
Efetuar logout (o)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	1,00
Saída (s)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Tabela D.5: Taxas de atendimento dos servidores

Figura D.3: Gráficos obtidos a partir de dados gerados pelo simulador



Resultados

Iteração 01 Um simulador de acordo com o modelo em redes de filas foi desenvolvido na linguagem Java com o uso da biblioteca *SimJava* [64], a fim de facilitar e agilizar a construção do mesmo.

Conclusão

A partir da análise dos resultados obtidos foi possível concluir que:

- O tempo médio de resposta do sistema permaneceu abaixo do limite, 15 segundos, para uma taxa de entrada entre 1 e 19 sessões por segundo;
- Cada sessão teve uma duração média de 78 segundos, logo, a exigência de 200 sessões simultâneas foi atingida a partir da taxa de uma entrada de 3 sessões por segundo;
- O sistema permaneceu estável até um nível de milhares de sessões simultâneas, o que é muito acima dos limites exigidos, portanto, não é necessário que múltiplas instâncias sejam executadas para manter o desempenho desejável;
- Deve-se atentar para o uso de, por exemplo, uma instância reservada para ser executada caso a instância principal falhe por algum motivo inesperado.

D.4.3 Sistema Portável

Configuração

Métricas:	Pontuação com $m \in [0, 1]$
Limites:	$m \geq 0,3$
Técnicas:	*Checklists*

Checklist

Pergunta
1. A linguagem de programação escolhida pode ser compilada para uso em vários sistemas operacionais?
2. A linguagem de programação evita o uso de códigos específicos de um sistema operacional na linguagem de programação para tarefas triviais? Por exemplo, para criar janelas, botões, etc. de uma interface gráfica.
3. A linguagem pode ser utilizada em sistemas operacionais Windows?
4. A linguagem pode ser utilizada em sistemas operacionais Linux?
5. A linguagem pode ser utilizada em sistemas operacionais AIX?
6. A linguagem pode ser utilizada em sistemas operacionais Mac OS X?
7. Todas as funcionalidades do sistema não exigem o uso de um sistema operacional específico?
8. O servidor *web* escolhido pode ser utilizado em vários sistemas operacionais?
9. Grandes companhias utilizam a linguagem de programação escolhida?
10. A linguagem de programação é atualizada constantemente com novas versões e funcionalidades?

Resultados

Todas as perguntas foram respondidas positivamente, logo, $m = 1$, o que satisfaz o limite mínimo desejado.

Pergunta	Sim	Não	N/A
1. A linguagem de programação escolhida pode ser compilada para uso em vários sistemas operacionais? **Justificativa:** A partir de uma única compilação, é possível utilizar as classes na máquina virtual Java para o sistema operacional desejado.	X		
2. A linguagem de programação evita o uso de códigos específicos de um sistema operacional na linguagem de programação para tarefas triviais? Por exemplo, para criar janelas, botões, etc. de uma interface gráfica. **Justificativa:** Pode-se utilizar o *Swing* para produzir interfaces gráficas. Para outras tarefas, como manipulação de estruturas de dados, leitura/gravação de arquivos, uso de *threads*, etc. também não é necessário o uso de um código específico.	X		
3. A linguagem pode ser utilizada em sistemas operacionais Windows? **Justificativa:** A implementação da *Oracle* é compatível [65].	X		
4. A linguagem pode ser utilizada em sistemas operacionais Linux? **Justificativa:** A implementação da *Oracle* é compatível [65].	X		
5. A linguagem pode ser utilizada em sistemas operacionais AIX? Justificativa: A implementação da IBM é compatível [66].	X		
6. A linguagem pode ser utilizada em sistemas operacionais Mac OS X? **Justificativa:** A implementação da *Oracle* é compatível [65].	X		

7. Todas as funcionalidades do sistema não exigem o uso de um sistema operacional específico? **Justificativa:** Não há funcionalidades que exijam o uso de código nativo ou de uma biblioteca específica que só funcione em um sistema operacional.	X		
8. O servidor *web* escolhido pode ser utilizado em vários sistemas operacionais? **Justificativa:** O Apache Tomcat é desenvolvido em Java.	X		
9. Grandes companhias utilizam a linguagem de programação escolhida? **Justificativa: [67]** • 1,1 bilhão de desktops executam Java; • 930 milhões de download do Java Runtime Environment a cada ano; • 3 bilhões de telefones celulares executam Java; • Telefones Java são lançados a cada ano em um número 31 vezes maior que a *Apple* e a *Android* juntas; • 100% de *Blu-ray players* executam Java; • 1,4 bilhões de Placas Java são fabricadas a cada ano; • Decodificadores Java, impressoras, câmeras *web*, *games*, sistemas de navegação automotiva, casas lotéricas, dispositivos médicos, estações de pagamento de estacionamento e mais.	X		

10. A linguagem de programação é atualizada constantemente com novas versões e funcionalidades? Justificativa: Novas versões são lançadas constantemente [1].	X		

Conclusão

O limite mínimo foi atingido, portanto, não houve necessidade de uma nova iteração.

D.4.4 Acesso Restrito

Configuração

Métricas:	Percentual de sucesso com $k = 1$
Limites:	$m \geq 0,8$
Técnicas:	Cenários

Cenários

Nome:	Acessar um recurso não permitido ao usuário autenticado
Partes interessadas:	Analista de segurança
Descrição:	1. O usuário X se autentica no PSI; 2. X informa o endereço de um recurso não permitido a ele; 3. Uma tela ou mensagem de acesso negado é exibida ao usuário.

Nome:	Acessar um recurso sem nenhum usuário autenticado
Partes interessadas:	Analista de segurança
Descrição:	1. O usuário Y informa o endereço de um recurso restrito; 2. A tela de autenticação do PSI é exibida.

[1] http://en.wikipedia.org/wiki/Java_version_history

Nome:	Acessar um recurso com um usuário desativado
Partes interessadas:	Analista de segurança
Descrição:	1. O usuário X se autentica no PSI; 2. O usuário administrador Y desativa X; 3. X tenta acessar algum recurso; 4. Uma tela ou mensagem de acesso desativado é exibida ao usuário.

Resultados

Iteração 01 Um teste não foi executado corretamente por falta de informações na arquitetura, logo, $m = 0,\overline{6}$, o que não satisfaz o limite mínimo desejado.

Nome:	Acessar um recurso não permitido ao usuário autenticado
Execução:	1. OK - `ModuloLogin` responsável pelo lógica de login. `TelaLogin` responsável pela interface gráfica; 2. OK; 3. OK - A biblioteca do JOSSO é encarregada de encaminhar o usuário para uma página de erro.
Situação:	OK

Nome:	Acessar um recurso sem nenhum usuário autenticado
Execução:	1. OK. 2. OK - A biblioteca do JOSSO é encarregada de encaminhar o usuário para uma página de login caso ele não esteja autenticado.
Situação:	OK

Nome:	Acessar um recurso com um usuário desativado
Execução:	1. OK - `ModuloLogin` responsável pelo lógica de login. `TelaLogin` responsável pela interface gráfica. 2. OK - `ModuloUsuario` responsavél pela lógica de usuários. `TelaUsuarios` responsável pela interface gráfica; 3. OK; 4. Falha - Não há menção de como é feita a checagem a cada acesso para verificar se o usuário está desativado.
Situação:	Falha: não foi possível executar o teste por falta de informações na arquitetura.

Iteração 02 O teste que falhava foi corrigido e agora $m = 1$, satisfazendo o limite mínimo desejado.

Nome:	Acessar um recurso não permitido ao usuário autenticado
Execução:	1. OK - `ModuloLogin` responsável pelo lógica de login. `TelaLogin` responsável pela interface gráfica; 2. OK; 3. OK - A biblioteca do JOSSO é encarregada de encaminhar o usuário para uma página de erro.
Situação:	OK

Nome:	Acessar um recurso sem nenhum usuário autenticado
Execução:	1. OK. 2. OK - A biblioteca do JOSSO é encarregada de encaminhar o usuário para uma página de login caso ele não esteja autenticado.
Situação:	OK

Nome:	Acessar um recurso com um usuário desativado
Execução:	1. OK - `ModuloLogin` responsável pelo lógica de login. `TelaLogin` responsável pela interface gráfica. 2. OK - `ModuloUsuario` responsavél pela lógica de usuários. `TelaUsuarios` responsável pela interface gráfica; 3. OK; 4. OK - Uma mensagem de acesso desativado será exibida ao usuário. A decisão-chave **Bloquear acesso de usuários desativados** trata dessa situação.
Situação:	OK

Conclusão

Após alterações na arquitetura, o limite mínimo foi atingindo na segunda iteração.

D.4.5 Login Único

Configuração

Métricas:	Pontuação com $m \in [0, 1]$
Limites:	$m \geq 0,5$
Técnicas:	*Checklists*

Checklist

Pergunta
1. A implementação de SSO escolhida é *open source*?
2. A implementação de SSO escolhida pode ser utilizada em *clusters* computacionais?
3. Grandes companhias utilizam a implementação de SSO escolhida?
4. A implementação de SSO escolhida é compatível com diversas linguagens de programação?
5. A implementação de SSO escolhida é implantável em diferentes servidores *web*?

6. A implementação de SSO escolhida permite o uso de LDAP?
7. A implementação de SSO escolhida permite o uso da funcionalidade "Lembrar login"?
8. A interface de login utilizada é customizável?
9. A implementação de SSO escolhida é atualizada em intervalos aceitáveis?
10. A implementação de SSO escolhida possui uma boa documentação?

Resultados

Iteração 01 Oito das dez perguntas foram respondidas positivamente, logo, $m = 0,8$, o que satisfaz o limite mínimo desejado.

Pergunta	**Sim**	**Não**	**N/A**
1. A implementação de SSO escolhida é *open source*? **Justificativa:** Existe uma versão open source que atende aos requisitos.	X		
2. A implementação de SSO escolhida pode ser utilizada em *clusters* computacionais? **Justificativa:** A utilização em *clusters* computacionais só é possível com a versão paga.		X	
3. Grandes companhias utilizam a implementação de SSO escolhida? **Justificativa:** De acordo com [68], várias companhias utilizam o JOSSO.	X		
4. A implementação de SSO escolhida é compatível com diversas linguagens de programação? **Justificativa:** JOSSO é compatível com Java, PHP e ASP.	X		
5. A implementação de SSO escolhida é implantável em diferentes servidores *web*? **Justificativa:** JOSSO é compatível com *JBoss*, *Tomcat*, *Weblogic*, *Websphere CE*, *Geronimo*, *Jetty*, *Apache* e IIS.	X		

6. A implementação de SSO escolhida permite o uso de LDAP? **Justificativa:** É possível utilizar LDAP como fonte de dados de acesso.	X		
7. A implementação de SSO escolhida permite o uso da funcionalidade "Lembrar login"? **Justificativa:** Essa funcionalidade está listada em [69].	X		
8. A interface de login utilizada é customizável? **Justificativa:** Como pode ser visto em [70], é possível.	X		
9. A implementação de SSO escolhida é atualizada em intervalos aceitáveis? **Justificativa:** A versão mais recente foi liberada em 31/08/2012 e os intervalos são aceitáveis [71].	X		
10. A implementação de SSO escolhida possui uma boa documentação? **Justificativa:** Embora existam vários tutoriais no site do fabricante para as tarefas mais comuns, não há uma documentação centralizada e bem explicativa.		X	

Iteração 02 Não houve alterações que afetaram esse requisito arquitetural.

Conclusão

O limite mínimo foi atingido, portanto, não houve necessidade direta de uma nova iteração, mas outra foi realizada por causa de requisitos relacionados.

D.4.6 API para Integração

Configuração

Métricas:	Pontuação com $m \in [0, 1]$
Limites:	$m \geq 0,8$
Técnicas:	*Checklists*

Checklist

Pergunta
1. A API para integração ignora o uso de uma linguagem de programação específica?
2. A interface da API está definida?
3. A tecnologia de comunicação escolhida é bem documentada?
4. A tecnologia de comunicação escolhida é *open source*?
5. A tecnologia de comunicação escolhida é utilizada por grandes companhias?
6. A tecnologia de comunicação escolhida é atualizada em intervalos aceitáveis?
7. A tecnologia de comunicação escolhida pode ser utilizada em clusters computacionais?
8. A tecnologia de comunicação escolhida permite o uso de conexão segura?

Resultados

Iteração 01 Uma das oito perguntas foram respondidas positivamente, logo,m = $0,125$, o que **não** satisfaz o limite mínimo desejado.

Pergunta	**Sim**	**Não**	**N/A**
1. A API para integração ignora o uso de uma linguagem de programação específica? **Justificativa:** Existe uma interface REST que pode ser utilizada a partir de outras linguagens de programação, mas a biblioteca Java só pode ser utilizada na mesma linguagem.		X	

2. A interface da API está definida? **Justificativa:** Podem existir métodos que não foram previstos na etapa de preparação da arquitetura.		X	
3. A tecnologia de comunicação escolhida é bem documentada? **Justificativa:** A documentação do protocolo pode ser encontrada em [72].	X		
4. A implementação da tecnologia de comunicação escolhida é *open source*? **Justificativa:** A implementação de REST a ser utilizada não foi mencionada na arquitetura do PSI.			X
5. A tecnologia de comunicação escolhida é utilizada por grandes companhias? **Justificativa:** A implementação de REST a ser utilizada não foi mencionada na arquitetura do PSI.			X
6. A implementação da tecnologia de comunicação escolhida é atualizada em intervalos aceitáveis? **Justificativa:** A implementação de REST a ser utilizada não foi mencionada na arquitetura do PSI.			X
7. A implementação da tecnologia de comunicação escolhida pode ser utilizada em clusters computacionais? **Justificativa:** A implementação de REST a ser utilizada não foi mencionada na arquitetura do PSI.			X
8. A implementação da tecnologia de comunicação escolhida permite o uso de conexão segura? **Justificativa:** A implementação de REST a ser utilizada não foi mencionada na arquitetura do PSI.			X

Iteração 02 Cinco das oito perguntas foram respondidas positivamente, logo, $m = 0,625$, o que não satisfaz o limite mínimo desejado.

Pergunta	Sim	Não	N/A
1. A API para integração ignora o uso de uma linguagem de programação específica? **Justificativa:** Existe uma interface REST que pode ser utilizada a partir de outras linguagens de programação, mas a biblioteca Java só pode ser utilizada na mesma linguagem.		X	
2. A interface da API está definida? **Justificativa:** Podem existir métodos que não foram previstos na etapa de preparação da arquitetura.		X	
3. A tecnologia de comunicação escolhida é bem documentada? **Justificativa:** A documentação do protocolo pode ser encontrada em [72].	X		
4. A implementação da tecnologia de comunicação escolhida é *open source*? **Justificativa:** Como pode ser visto em [73], *RestEasy* é *open source*.	X		
5. A tecnologia de comunicação escolhida é utilizada por grandes companhias? **Justificativa:** Não foram encontradas informações sobre grandes companhias que usam *RestEasy*.			X
6. A implementação da tecnologia de comunicação escolhida é atualizada em intervalos aceitáveis? **Justificativa:** A versão mais recente foi liberada em 07/05/2013 e os intervalos são aceitáveis [74].	X		

7. A implementação da tecnologia de comunicação escolhida pode ser utilizada em clusters computacionais? **Justificativa:** A implementação de REST a ser utilizada não foi mencionada na arquitetura do PSI.	X		
8. A implementação da tecnologia de comunicação escolhida permite o uso de conexão segura? **Justificativa:** A implementação de REST a ser utilizada não foi mencionada na arquitetura do PSI.	X		

Conclusão

O limite mínimo não foi atingido mesmo após a segunda iteração, todavia, o esforço de definir a API a ser oferecida não compensa diante da dificuldade de identificar todos os métodos a serem implementados. Ainda, a obrigação do uso da linguagem de programação Java para utilizar a biblioteca oferecida não é problema pois existe a interface REST, que pode ser acessada através de outras linguagens.

D.4.7 Fornecer Dados de Autenticação para Outros Sistemas

Configuração

Métricas: Pontuação com $m \in [0, 1]$

Limites: $m \geq 0,3$

Técnicas: *Checklists*

Checklist

Pergunta
1. A tecnologia de autenticação ignora o uso de uma linguagem de programação específica?
2. A tecnologia de autenticação escolhida possui uma boa documentação?

3. A implementação da tecnologia de autenticação escolhida é *open source*?
4. A implementação da tecnologia de autenticação escolhida é utilizada por grandes companhias?
5. A implementação da tecnologia de autenticação escolhida é atualizada em intervalos aceitáveis?
6. A implementação da tecnologia de autenticação escolhida pode ser utilizada em clusters computacionais?

Resultados

Iteração 01 Duas das seis perguntas foram respondidas positivamente, logo, $m = 0,\overline{3}$, o que satisfaz o limite mínimo desejado.

Pergunta	**Sim**	**Não**	**N/A**
1. A tecnologia de autenticação ignora o uso de uma linguagem de programação específica? **Justificativa:** Várias linguagens como C, C++, Java, PHP, etc. possuem suporte a LDAP.	X		
2. A tecnologia de autenticação escolhida possui uma boa documentação? **Justificativa:** A documentação do protocolo pode ser encontrada em [75].	X		
3. A implementação da tecnologia de autenticação escolhida é open source? **Justificativa:** A implementação de LDAP a ser utilizada não foi mencionada na arquitetura do PSI.			X
4. A implementação da tecnologia de autenticação escolhida é utilizada por grandes companhias? **Justificativa:** A implementação de LDAP a ser utilizada não foi mencionada na arquitetura do PSI.			X

5. A implementação da tecnologia de autenticação escolhida é atualizada em intervalos aceitáveis? **Justificativa:** A implementação de LDAP a ser utilizada não foi mencionada na arquitetura do PSI.			X
6. A implementação da tecnologia de autenticação escolhida pode ser utilizada em clusters computacionais? **Justificativa:** A implementação de LDAP a ser utilizada não foi mencionada na arquitetura do PSI.			X

Iteração 02 Não houve alterações que afetaram esse requisito arquitetural.

Conclusão

O limite mínimo foi atingido, portanto, não houve necessidade direta de uma nova iteração, mas outra foi realizada por causa de requisitos relacionados.

D.4.8 Comunicação Segura

Configuração

Métricas: Pontuação com $m \in [0, 1]$

Limites: $m \geq 0,8$

Técnicas: *Checklists*

Checklist

Pergunta
1. A tecnologia de comunicação ignora o uso de uma linguagem de programação específica?
2. A tecnologia de comunicação escolhida é bem documentada?
3. A tecnologia de comunicação escolhida é *open source*?
4. A tecnologia de comunicação escolhida é utilizada por grandes companhias?

5. Todas as comunicações realizadas entre o sistema em questão e o cliente são de forma segura?
6. A tecnologia de comunicação escolhida é implantada de forma transparente para o sistema em questão?
7. A tecnologia de comunicação escolhida é compatível com o servidor *web* escolhido?
8. Existe um padrão a ser seguido pelas senhas utilizadas?
9. As senhas utilizadas são trocadas periodicamente?
10. O certificado de segurança é ou será emitido por uma empresa especializada?
11. O sistema em questão verifica se as senhas cadastradas são facilmente quebráveis via força bruta?

Resultados

Iteração 01 Oito das onze perguntas foram respondidas positivamente, logo, $m = 0,\overline{72}$, o que não satisfaz o limite mínimo desejado.

Pergunta	**Sim**	**Não**	**N/A**
1. A tecnologia de comunicação ignora o uso de uma linguagem de programação específica? **Justificativa:** Várias linguagens como C, C++, Java, PHP, etc. possuem suporte a SSL.	X		
2. A tecnologia de comunicação escolhida é bem documentada? **Justificativa:** A documentação do protocolo SSL pode ser encontrada em [76].	X		
3. A implementação da tecnologia de comunicação escolhida é open source? **Justificativa:** Com o uso de Java, a comunicação segura pode ser implantada sem nenhuma biblioteca adicional.	X		

4. A tecnologia de comunicação escolhida é utilizada por grandes companhias? **Justificativa:** *Google*, *Apple*, *Microsoft*, *Amazon*, dentre outras grandes companhias utilizam comunicação segura em seus sites.	X		
5. Todas as comunicações realizadas entre o sistema em questão e o cliente são de forma segura? **Justificativa:** Qualquer acesso *web* será possivel somente com o uso de conexão segura.	X		
6. A tecnologia de comunicação escolhida é implantada de forma transparente para o sistema em questão? **Justificativa:** A ativação do uso de SSL pode ser feita na configuração do *Apache Tomcat*.	X		
7. A tecnologia de comunicação escolhida é compatível com o servidor *web* escolhido? **Justificativa:** Existe uma parte específica da documentação do *Apache Tomcat* explicando a configuração da comunicação SSL.	X		
8. Existe um padrão a ser seguido pelas senhas utilizadas? **Justificativa:** Um padrão a ser seguido não foi mencionado na arquitetura do PSI.		X	
9. As senhas utilizadas são trocadas periodicamente? **Justificativa:** Existe um requisito não-funcional para isso.	X		
10. O certificado de segurança é ou será emitido por uma empresa especializada? **Justificativa:** Os certificados de segurança serão emitidos pelos próprios desenvolvedores do PSI.		X	

11. O sistema em questão verifica se as senhas cadastradas são facilmente quebráveis via força bruta? **Justificativa:** Essa informação não foi mencionada na arquitetura do PSI mas exige uma atenção especial.		X	

Iteração 02 Nove das onze perguntas foram respondidas positivamente, logo, $m = 0,\overline{81}$, o que satisfaz o limite mínimo desejado.

Pergunta	**Sim**	**Não**	**N/A**
1. A tecnologia de comunicação ignora o uso de uma linguagem de programação específica? **Justificativa:** Várias linguagens como C, C++, Java, PHP, etc. possuem suporte a SSL.	X		
2. A tecnologia de comunicação escolhida é bem documentada? **Justificativa:** A documentação do protocolo SSL pode ser encontrada em [76].	X		
3. A implementação da tecnologia de comunicação escolhida é open source? **Justificativa:** Com o uso de Java, a comunicação segura pode ser implantada sem nenhuma biblioteca adicional.	X		
4. A tecnologia de comunicação escolhida é utilizada por grandes companhias? **Justificativa:** *Google*, *Apple*, *Microsoft*, *Amazon*, dentre outras grandes companhias utilizam comunicação segura em seus sites.	X		

5. Todas as comunicações realizadas entre o sistema em questão e o cliente são de forma segura? **Justificativa:** Qualquer acesso *web* será possivel somente com o uso de conexão segura.	X		
6. A tecnologia de comunicação escolhida é implantada de forma transparente para o sistema em questão? **Justificativa:** A ativação do uso de SSL pode ser feita na configuração do *Apache Tomcat*.	X		
7. A tecnologia de comunicação escolhida é compatível com o servidor *web* escolhido? **Justificativa:** Existe uma parte específica da documentação do *Apache Tomcat* explicando a configuração da comunicação SSL.	X		
8. Existe um padrão a ser seguido pelas senhas utilizadas? **Justificativa:** Existe uma decisão-chave tratando da elaboração de um padrão.	X		
9. As senhas utilizadas são trocadas periodicamente? **Justificativa:** Existe um requisito não-funcional para isso.	X		
10. O certificado de segurança é ou será emitido por uma empresa especializada? **Justificativa:** Os certificados de segurança serão emitidos pelos próprios desenvolvedores do PSI.		X	
11. O sistema em questão verifica se as senhas cadastradas são facilmente quebráveis via força bruta? **Justificativa:** Essa informação não foi mencionada na arquitetura do PSI mas exige uma atenção especial.		X	

Conclusão

Após alterações na arquitetura, o limite mínimo foi atingindo na segunda iteração.

D.5 Conclusão Final

A validação do sistema PSI foi considerada satisfatória pois somente **um** dos **oito** requisitos arquiteturais não atingiu o limite mínimo exigido. Os requisitos não validados na primeira iteração foram melhores descritos, aumentando o nível de descrição da arquitetura e evitando decisões futuras que poderiam causar impactos inesperados.