UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA E
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA

MARCELO MASSAO KATAOKA HIGASKINO
THAYSE MARQUES SOLIS

ACESSO DE ÁREA DE TRABALHO REMOTA VIA APLICAÇÃO WEB PROTEGIDO POR CRIPTOGRAFIA

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA

Dezembro de 2014

MARCELO MASSAO KATAOKA HIGASKINO
THAYSE MARQUES SOLIS

ACESSO DE ÁREA DE TRABALHO REMOTA VIA APLICAÇÃO WEB PROTEGIDO POR CRIPTOGRAFIA

Relatório da disciplina de Trabalho de Conclusão de Curso 2 de Engenharia de Computação apresentado ao Departamento Acadêmico de Informática e Departamento Acadêmico de Eletrônica da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de "Engenheiro de Computação".

Orientador: Prof. Dr. Carlos Alberto Maziero

CURITIBA

Dezembro de 2014

AGRADECIMENTOS

A Universidade Tecnológica Federal do Paraná, seu corpo docente e direção que proporcionaram a oportunidade de concluir o curso de Engenharia de Computação.

A CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior), o DAAD (*Deutscher Akademischer Austausch Dienst*) e a *Universität Duisburg-Essen*, bem como seus respectivos colaboradores, permitiram a aquisição de conhecimento que se mostrou de grande importância para o desenvolvimento deste trabalho.

Dedicamos especial agradecimento ao Professor Carlos Alberto Maziero, orientador que soube guiar-nos para alcançarmos nossos objetivos.

Aos nossos pais, irmãos e a toda a família que, com muito carinho, nos apoiaram nas horas difíceis.

A todos que direta ou indiretamente fizeram parte da nossa formação, o nosso muito obrigado.

RESUMO

KATAOKA HIGASKINO, Marcelo Massao, MARQUES SOLIS, Thayse. ACESSO DE ÁREA DE TRABALHO REMOTA VIA APLICAÇÃO WEB PROTEGIDO POR CRIPTOGRAFIA. 133 f. Trabalho de Conclusão de Curso – Departamento Acadêmico de Informática e Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, Dezembro de 2014.

Fala-se muito em acesso à informação à distância e na necessidade de se utilizar um sistema remotamente. Os usuários precisam realizar operações em máquinas remotas, mas é indesejável, às vezes impossível, que o acesso implique na instalação de um novo aplicativo na máquina local. Dessa forma, torna-se interessante o desenvolvimento de uma forma segura para acesso a uma máquina remota pela internet, através de um navegador web. O objetivo deste trabalho é a implementação de um software de acesso de área de trabalho remota com criptografia na comunicação entre cliente e servidor. O cliente pode ser uma máquina simples, sem a necessidade de hardware dedicado, mas que possua um navegador web atualizado. Inicialmente foi desenvolvido um plano de projeto, que serviu como base à implementação. Nesta etapa, foi empregado um método de desenvolvimento em espiral, o qual partiu da concepção de pequenos protótipos, que evoluíram incrementalmente até o resultado esperado. O serviço fornecido permite que a área de trabalho do servidor seja exibida em um navegador executado na máquina cliente, cujos teclado e mouse são usados para controlar o servidor. É o navegador web que provê a conexão e a visualização. O resultado esperado é que o sistema desenvolvido facilite a interação com uma máquina remota e com uma garantia mínima de segurança sem a necessidade do cliente ter de instalar um novo aplicativo na máquina local. Essa ideia é útil no contexto empresarial ou de educação, quando nem sempre determinadas funcionalidades estão disponíveis na máquina local, como a de possuir privilégios elevados no sistema operacional.

Palavras-chave: Área de Trabalho remota, Segurança, Navegador Web, WebSockets

ABSTRACT

KATAOKA HIGASKINO, Marcelo Massao, MARQUES SOLIS, Thayse. CRYPTOGRAPHICALLY SECURE REMOTE DESKTOP WEB APPLICATION. 133 f. Trabalho de Conclusão de Curso – Departamento Acadêmico de Informática e Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, Dezembro de 2014.

Remote information access is usually cited and the necessity to access systems remotely is in common place nowadays. Users need to execute instructions in remote machines, however it is highly undesirable, and sometimes impossible, to setup a new program at the local machine. Due to this fact, it is interesting to develop a secure access to a remote machine through an internet connection, using a web browser. We aimed to implement a remote desktop sharing software with cryptographically secured communication between client and server. The client side machine can be simple, even without dedicated hardware. However it must have an updated web browser. Initially, a project plan was developed and it served as the basis for implementation. In this step, we employed a spiral development method which started from the design of small prototypes that evolved incrementally until the desired result. The provided service allows the server side desktop to be displayed in a web browser executed in the client machine, whose keyboard and mouse are used to control the server. It is the web browser that provides the connection and viewing. We expected to make it easier and more secure to access a remote desktop without setting up any new application at the local desktop. This idea is useful at workstations or educational facilities, where certain functionalities may not be available, e.g. administrative/root privileges in operating system.

Keywords: Remote Desktop, Security, Web Browser, WebSockets

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ADIT	*Analysis, Design, Implementation, Testing*
Ajax	*Asynchronous JavaScript and XML*
API	*Application Programming Interface*
BLOB	*Binary Large Object*
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
codec	Codificador/Decodificador
CPU	*Central Processing Unit*
CSS	*Cascading Style Sheets*
DAAD	*Deutscher Akademischer Austausch Dienst*
DES	*Data Encription Standard*
DOM	*Document Object Model*
ECF	*Environmental Complexity Factor*
EPG	*Electronic Program Guide*
FR	Requisito Funcional
GPU	*Graphics Processing Unit*
GUI	*Graphical User Interface*
HTML5	*Hypertext Markup Language 5*
IDE	Ambiente de Desenvolvimento Integrado
IP	*Internet Protocol*
JKS	*Java KeyStore*
JNI	*JAVA Native Interface*
JS	JavaScript
JSON	*Java Script Object Notation*
JVM	*Java Virtual Machine*
LVD	*Lightweight Virtual Desktop*
MJPEG	*Motion Joint Photographic Expert Group*
MPS.BR	Melhoria de Processo do Software Brasileiro
NFR	Requisito Não-Funcional
NPAPI	*Netscape Plugin Application Programming Interface*
OCL	*Object Constraint Language*
OpenGL ES	*Open Graphics Library for Embedded Systems*
PEM	Privacy Enhanced Mail
PNG	*Portable Network Graphics*
RDP	*Remote Desktop Protocol*
RFB	*Remote Framebuffer Protocol*
SO	Sistema Operacional
SSL	*Secure Socket Layer*
TCC	Trabalho de Conclusão de Curso
TCF	*Technical Complexity Factor*
TCP	*Transmission Control Protocol*
TLS	*Transport Layer Security*

UAW	*Unadjusted Actor Weight*
UCP	*Use Case Points*
UML	*Unified Modelling Language*
UUCP	*Unadjusted Use Case Point*
UUCW	*Unadjusted Use Case Weight*
VM	Máquina Virtual
VNC	*Virtual Network Computing*
WebGL	*Web-based Graphics Library*
WSS	WebSocket Secure
Xpra	*X Persistent Remote Applications*

SUMÁRIO

1 INTRODUÇÃO

Diante da ubiquidade do acesso à internet e do fato dos usuários possuírem mais de um computador (um *desktop* em casa e um *notebook* nos demais ambientes, por exemplo), o acesso remoto a outras máquinas tornou-se uma infraestrutura interessante. Ao mesmo tempo, pode ser necessário acessar um computador pessoal através de uma máquina disponível na empresa, escola ou universidade com a impossibilidade de instalação de um aplicativo específico, além dos pré-disponíveis no computador, devido a falta de privilégios necessários para executar as ações mencionadas. Por essas razões, é interessante o desenvolvimento de uma forma segura para acesso a uma máquina remota pela internet, usando um navegador, por exemplo.

Nem sempre é possível ou desejável instalar um aplicativo específico para acesso a um desktop remoto. Assim, justifica-se o tema escolhido para este trabalho: a elaboração de um sistema que permita o acesso a uma máquina remota utilizando um navegador web. Os conhecimentos envolvidos neste projeto estão relacionados a redes de computadores, sistemas distribuídos, segurança e auditoria de sistemas e fundamentos de programação, todos das áreas de estudo no curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná, campus Curitiba. Dessa forma, o trabalho também se justifica como uma aplicação dos métodos e teorias aprendidos no curso.

1.1 OBJETIVOS GERAL E ESPECÍFICOS

O objetivo geral deste trabalho de conclusão de curso é desenvolver um software de prova de conceito para acesso seguro à área de trabalho remota que não necessite de um aplicativo dedicado no lado cliente, além do navegador web. Nesse contexto, entende-se como "cliente" o computador que acessa a área de trabalho remota, e "servidor" o computador cuja área de trabalho é acessada.

Os objetivos específicos propostos para a execução são:

- Visualizar a área de trabalho do servidor a partir do cliente (via navegador).

- Controlar a área de trabalho do servidor (via teclado e mouse) a partir do cliente.
- Cifrar a comunicação entre cliente e servidor.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está dividido em sete capítulos. O capítulo presente elucida a motivação e objetivos do desenvolvimento. O segundo capítulo apresenta a fundamentação teórica que engloba protocolos, tecnologias e boas práticas afins à área deste trabalho, assim como um levantamento bibliográfico com vários trabalhos relacionados. O terceiro capítulo elabora um estudo do processo de desenvolvimento de software: as fases e a descrição do processo. O quarto capítulo apresenta uma análise do processo de desenvolvimento; compreende a metodologia e as tecnologias e recursos escolhidos. O quinto capítulo contém dados do projeto, implementação e testes do protótipo. O sexto capítulo apresenta a análise post-mortem do projeto, envolvendo o cronograma executado, análise do protótipo e sugestões de trabalhos futuros. Por último, o sétimo capítulo apresenta as considerações finais acerca deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Ao longo deste TCC (Trabalho de Conclusão de Curso) diversos conceitos técnicos e teóricos são mencionados com frequência, pois servem de fundamento para os objetivos e a solução delineados. Portanto surge a necessidade de aprofundar algumas definições antes de iniciar de fato a explanação dos procedimentos utilizados e resultados obtidos ao longo do projeto. Este capítulo serve justamente para esse fim, além de ser referência para partes posteriores do texto.

A estrutura do capítulo consiste na apresentação dos três pilares deste trabalho: a definição do protocolo de área de trabalho remota, dos protocolos e técnicas de segurança e das técnicas e tecnologias de desenvolvimento de aplicativos web. Outro recurso apresentado são os *Design Patterns*, cuja utilização faz parte das boas práticas de programação e facilita a reutilização do código. Por fim é apresentado um conjunto de resultados obtidos no levantamento de referências que possuem algum tipo de relação com as necessidades deste projeto, com a finalidade de comparação do uso de tecnologias, abordagens e resultados.

2.1 PROTOCOLOS DE ÁREA DE TRABALHO REMOTA

Um dos pilares deste TCC é o conceito de área de trabalho remota, cuja função principal é a de delegar controle da área de trabalho de um computador para outros dispositivos. Para alcançar as funcionalidades desejadas, entretanto, é necessário um formato comum de comunicação, convencionado pela utilização de um protocolo. Um dos mais simples e ao mesmo tempo aberto é o RFB (*Remote Framebuffer Protocol*), que é a base da solução comercial VNC (*Virtual Network Computing*).

Richardson (2010) especifica o protocolo *RFB*, que permite o acesso remoto a uma GUI (*Graphical User Interface*). O funcionamento do RFB é exposto na Figura 1. O servidor informa ao cliente qual a versão mais nova suportada do protocolo e o cliente responde determinando qual versão será utilizada. Em seguida são trocadas mensagens com o intuito de acordar que tipo de segurança será utilizado (também existe a opção de não utilizar segurança).

Em seguida o cliente manda ao servidor uma mensagem de inicialização onde define uma *flag* que determina se o acesso é exclusivo ou não. O servidor, por sua vez, envia uma mensagem de inicialização ao cliente onde determina a largura e altura do seu *framebuffer*, o formato dos *pixels* e o nome associado ao *desktop*. Feito isso, o cliente passa a controlar a demanda de atualizações, como resposta recebe retângulos de *pixels* para uma posição específica, ao mesmo tempo que eventos de *mouse* e teclado são diretamente repassados ao servidor.

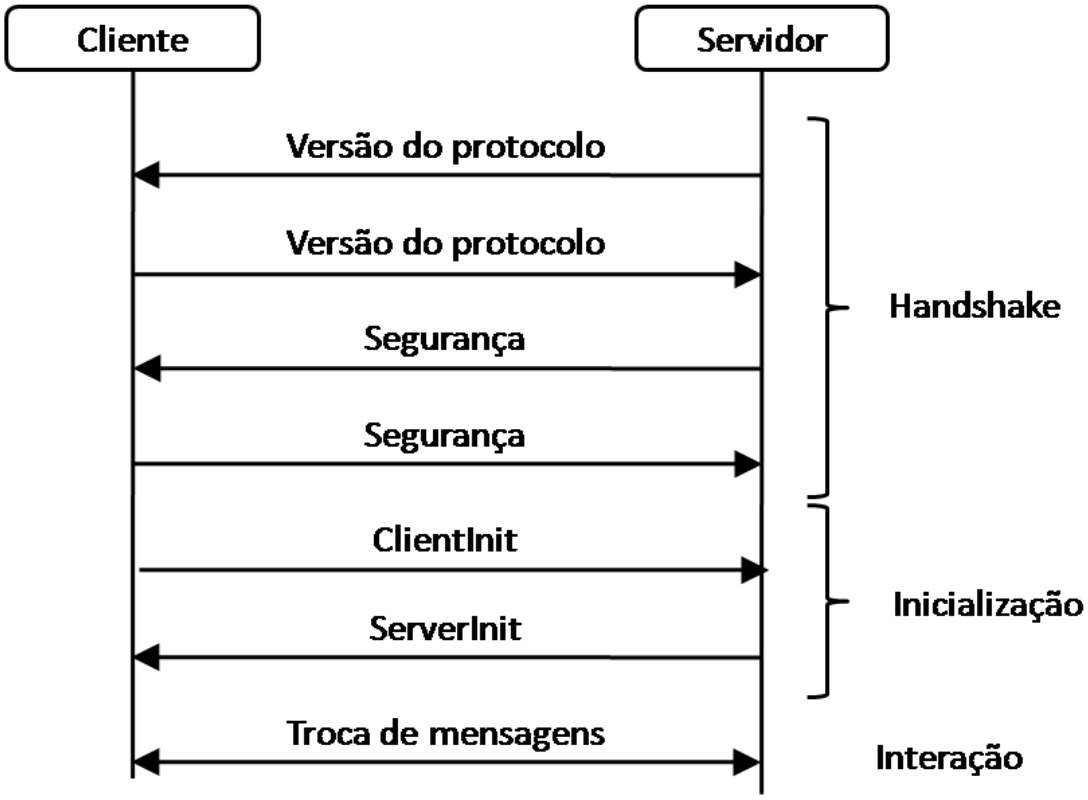

Figura 1: Diagrama ilustrando as fases/etapas do RFB.

O protocolo RFB pode ser estendido com medidas de segurança, tipos de mensagem e tipos de codificação de *pixels*. Esta flexibilidade é especialmente útil devida a certas limitações do protocolo, como, por exemplo, o fato da única medida de segurança definida na documentação do RFB ser a autenticação por *challenge-response* (desafio-resposta) cifrado em DES (*Data Encription Standard*). A técnica de desafio-resposta tem como base um segredo e um algoritmo de cifragem previamente definidos entre cliente e servidor. Uma das partes escolhe um valor aleatório (desafio) e o envia à outra. Essa, em posse do desafio, concatena-o com um segredo, calcula o resumo da concatenação e devolve à outra parte. Essa última, por sua vez, executa a mesma operação do seu lado e, utilizando o valor do segredo armazenado previamente, compara com o resultado obtido. Se os resultados forem iguais, a autenticação é efetivada.

Existem outras alternativas ao RFB. Entretanto, muitas delas são restritas em algum aspecto no qual o RFB dá plena liberdade, seja pela convenção do protocolo ou pela possibilidade

de estender novas funcionalidades. As implementações baseadas em Xpra (*X Persistent Remote Applications*) são restritas a aplicações gráficas baseadas em servidor gráfico X11, e limitam-se a exibição apenas das janelas das aplicações (DEVLOOP, 2014), enquanto a necessidade deste projeto é a de ter acesso a toda a área de trabalho do servidor. Por outro lado existem os protocolos que estão sob domínio proprietário e preso ao sistema operacional, como ocorre com o RDP (*Remote Desktop Protocol*) (MSDN, 2014). Cada protocolo possui seus méritos, mas o que melhor atende as necessidades do projeto é o RFB.

2.2 PROTOCOLOS DE SEGURANÇA

Outro pilar deste trabalho está na segurança de sistemas, definida pela garantia de propriedades específicas sobre informações e recursos dos sistemas. Segundo Maziero (2013) as propriedades fundamentais são: confidencialidade, integridade, disponibilidade, autenticidade e irretratabilidade. Apesar dos cinco aspectos serem importantes, o único que entra no escopo desse trabalho é a da confidencialidade - garantia de acesso a recursos do sistema somente por usuários devidamente autorizados - pois o protocolo de área de trabalho remota *VNC* não trabalha com criptografia alguma durante a comunicação, enviando dados na rede que podem ser interceptados e lidos facilmente por um atacante. Para garantir que isso não ocorra, técnicas de criptografia podem ser aplicadas, e as mais interessantes para este trabalho são baseadas em chaves. Podem ser utilizadas em algoritmos simétricos, que, através de uma função matemática, cifra e, com uma função inversa, decifra uma mensagem, nesse caso sequência de bits, com uma mesma chave, ou assimétricos, no qual as chaves de cifragem e decifragem são distintas, e geralmente esses algoritmos são bem mais lentos que os anteriores (MAZIERO, 2013).

Neste TCC o tópico de grande importância no quesito segurança reside no protocolo TLS/SSL (*Transport Layer Security*/*Secure Socket Layer*), que foi utilizado como meio de garantir a confidencialidade na aplicação desenvolvida.

O protocolo TLS/SSL é na prática a utilização do protocolo TLS. Ele é assim nomeado devido ao fato do desenvolvimento do TLS ter se baseado no antigo SSL 3.0. O principal objetivo do protocolo TLS/SSL é assegurar privacidade e integridade de dados entre aplicações, além da autenticidade (DIERKS; RESCORLA, 2008). Cada uma das propriedades de segurança é tratada em uma das duas camadas contidas no protocolo: o *TLS Handshake Protocol* e o *TLS Record Protocol*.

O *TLS Record Protocol* funciona sobre uma conexão confiável (checagem de integridade, recepção completa da mensagem, etc.) e é suportado por criptografia de chaves simétricas. Este é o nível mais baixo do TLS/SSL, servindo de base para outros protocolos e o próprio *TLS*

Handshake Protocol (DIERKS; RESCORLA, 2008).

O *TLS Handshake Protocol* é parte importante do protocolo, pois ele permite a autenticação por criptografia de chave pública, implementada na prática com base em certificados digitais, e a negociação confiável de um segredo compartilhado entre as partes, que é importante para o restante da execução sobre o *TLS Record Protocol*, que ocorre sob chaves simétricas (DIERKS; RESCORLA, 2008). Uma representação em diagrama pode ser visto na Figura 2.

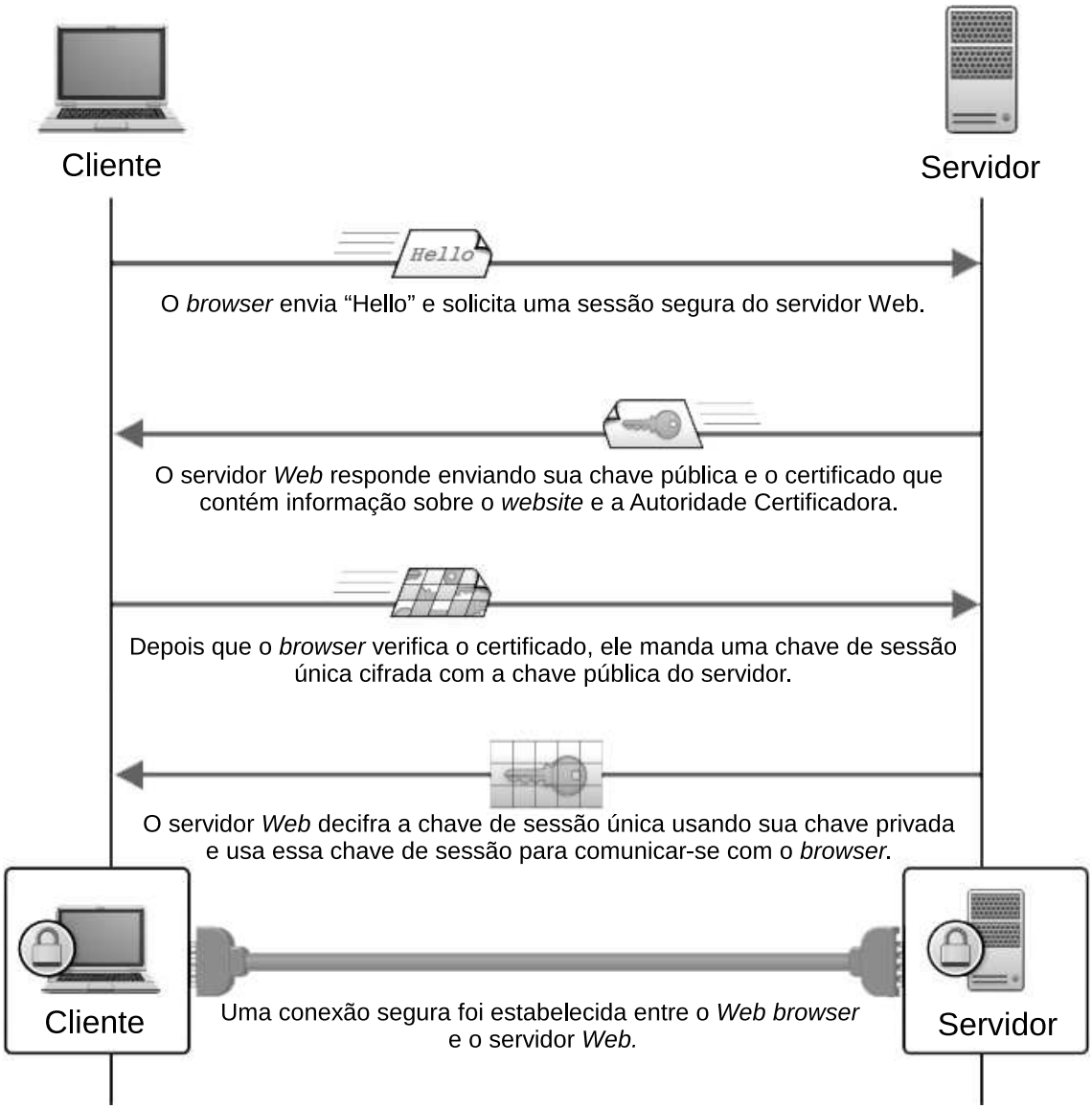

Figura 2: Diagrama ilustrando as fases/etapas do TSL. FONTE: (KAAZING, 2014).

A independência entre protocolo da camada de aplicação e o TLS/SSL (DIERKS; RESCORLA, 2008) aliada à implementação pronta para uso nos motores dos navegadores web mais populares fizeram desse protocolo a solução ideal para a garantia da segurança dos dados trocados pela aplicação desenvolvida neste trabalho. Por outro lado, caso uma das partes não utilize um navegador web como base, como ocorreu com o servidor da aplicação, torna-se

necessário dar atenção na existência de implementação compatível do protocolo para suportar a comunicação segura entre as partes.

2.3 DESENVOLVIMENTO DE APLICATIVOS WEB

O terceiro pilar deste TCC é o desenvolvimento de aplicações web. Com o avanço das linguagens de desenvolvimento de páginas web, que podem ter comportamento manipulado por scripts e estendidos por outras tecnologias, além do amadurecimento de navegadores web, a passagem de meras páginas para aplicativos web ocorreu. Com aumento de poder, consequentemente, técnicas de segurança foram implementadas como descrito pela Google (2013) no seu navegador: o Google Chrome, no qual uma página web acessada é confinada num ambiente isolado sem ter acesso a outras páginas ou a recursos do SO (Sistema Operacional). Essa característica de segurança dos navegadores web impactou as decisões de projeto, pois se tornou um empecilho ao acesso de recursos do SO, como a área de trabalho e a emulação de eventos nativos de periféricos.

Na sequência são apresentados o padrão HTML5, a linguagem JavaScript, a biblioteca jQuery, o DOM (*Document Object Model*) e o formato JSON (*Java Script Object Notation*).

2.3.1 HTML5

HTML 5 é o substituto do HTML 4.01, XHTML 1.0 e XHTML 1.1. Essa nova versão fornece novos recursos e padroniza várias características da plataforma web. Foi projetado para ser multi-plataforma e, na medida do possível, retrocompatível (PILGRIM, 2010).

Um dos recursos novos do HTML 5 é o elemento Canvas - uma área de *bitmap* dependente de resolução onde podem ser renderizadas imagens em “tempo real”. É basicamente um retângulo dentro da página que pode ser usado para desenhar qualquer coisa usando JavaScript (PILGRIM, 2010).

Há também suporte a um novo elemento - *video* - para embutir vídeo em páginas web sem o uso de *plugins*. Contudo há ainda uma questão em aberto no que diz respeito ao uso de codecs: cada navegador decide qual sua compatibilidade com cada tipo de codec (PILGRIM, 2010).

Outra novidade diz respeito à Web Workers: uma maneira padrão para navegadores rodarem código JavaScript em *threads* separadas. Isso permite que cada *thread* faça cálculos complexos, solicitações de rede e acesse armazenamento local enquanto a página principal continua respondendo ao mouse e teclado (PILGRIM, 2010). Contudo, Web Workers são bem

mais limitados, eles funcionam como se estivessem dentro de uma *sandbox* e só tem acesso a algumas APIs e funções (LAWSON; SHARP, 2011).

2.3.2 JAVASCRIPT

JavaScript é uma linguagem derivada do C e C++ (GOODMAN, 2001) e dá suporte a funções. Ela é fracamente tipada, possui objetos dinâmicos e uma notação literal de objetos que inspirou o JSON (CROCKFORD, 2008). A linguagem é usada principalmente em programação Web tendo acesso a objetos que representam a janela do navegador e seus conteúdos.

O JavaScript usado do lado cliente de uma aplicação combina um interpretador JavaScript com o DOM definido pelo navegador. Isso possibilita que conteúdo executável seja incluído em páginas web, permitindo controle sobre o navegador e o conteúdo HTML, interação com modelos HTML e com o usuário, leitura e escrita do estado do cliente com Cookies, interação com applets Java, entre outros (FLANAGAN, 1998).

O JavaScript também permite tornar páginas Web mais iterativas, fornecendo *feedback* ao usuário durante a navegação: validando o preenchimento de formulários, podendo exibir novas janelas, caixas de alerta e mensagens personalizadas na barra de status do navegador (NEGRINO; SMITH, 2009).

2.3.3 JQUERY

A biblioteca jQuery fornece uma camada de abstração para a escrita de código web. Um de seus recursos é a possibilidade de acessar elementos em um documento de maneira fácil, atravessando a árvore do DOM de forma a recuperar a parte exata do documento que deve ser manipulada. Permite modificar a aparência da página usando o mesmo padrão em todos os navegadores e apresentar mudanças animadas em um documento usando seletores CSS (*Cascading Style Sheets*). Também responde a interação com o usuário e simplifica tarefas comuns em JavaScript como manipulação de vetores (CHAFFER; SWEDBERG, 2009).

2.3.4 DOCUMENT OBJECT MODEL

No Javascript, enquanto o objeto Window representa uma janela, o objeto Document representa os conteúdos do Window (FLANAGAN, 1998).

O objeto Document tem propriedades que especificam as informações sobre o documento exibido no navegador, métodos que permitem que programas JavaScript produzam texto

dinamicamente em um documento e criem novos documentos do zero, e vetores que representam forms, imagens, links, âncoras e applets contidos no documento. Resumindo: é o objeto Document que dá acesso interativo aos conteúdos dos documentos que até então eram estáticos (FLANAGAN, 1998).

DOM surgiu da necessidade de padronizar o desenvolvimento. Pois havia um conflito entre os modelos de objeto dos navegadores web da Netscape e da Microsoft que dificultavam o desenvolvimento. Assim como foram criados HTML para padronizar conteúdo e CSS para estilos, o W3C assumiu a responsabilidade de criar um padrão de modelo de objeto do documento (GOODMAN, 2001). O DOM é um padrão que especifica como uma linguagem script pode acessar e manipular a estrutura detalhada de um documento HTML (FLANAGAN, 1998).

Como DOM deveria ser aplicado a XML e HTML, sua especificação foi dividida em duas seções. O Core DOM que define especificações para a estrutura básica do documento compartilhada entre HTML e XML e a segunda parte foca apenas em elementos do HTML. O DOM é uma especificação em evolução e dessa forma é comum que novas versões dos navegadores não incluam tudo que é proposto em uma nova versão da especificação (GOODMAN, 2001).

2.3.5 JAVA SCRIPT OBJECT NOTATION

O JSON é um formato de texto para a serialização de dados estruturados. Pode representar quatro tipos primitivos (strings, numbers, booleans e null) e dois tipos estruturados (objects e arrays) (CROCKFORD, 2006). O JSON é de fácil escrita e leitura para humanos e ao mesmo tempo é fácil para um máquina realizar uma análise sintática - isso o torna uma linguagem de troca de dados ideal (JSON, 2014).

Ele é baseado em um subconjunto da linguagem JavaScript - o padrão de linguagem de programação ECMAScript (CROCKFORD, 2006). É independente de linguagem, mas possui convenções que seguem o da família do C (JSON, 2014).

JSON é serializado de duas formas: como coleção de pares nome-valor (como em um objeto, struct, dicionário, tabela hash, etc) ou em forma de lista ordenada de valores (como em arrays, vetores, lista ou sequencia) (JSON, 2014).

Apesar deste TCC não ter utilizado JSON explicitamente, ele faz parte de muitas aplicações web, inclusive funciona como fundamento de várias bibliotecas.

2.4 PROTOCOLO DE COMUNICAÇÃO - PADRÃO WEBSOCKET

O protocolo WebSocket é simétrico depois que a conexão é estabelecida: tanto cliente como servidor podem trocar mensagens e fechar a conexão a qualquer momento. O cliente envia uma requisição a um servidor WebSocket iniciando o *handshake*, compatível com a infra-estrutura HTTP: os servidores web interpretam-no como uma requisição de *upgrade* em uma conexão HTTP. Terminado o *handshake*, pode iniciar-se a transferência de dados. Um diagrama esquematizando o padrão WebSocket está representado na Figura 3.

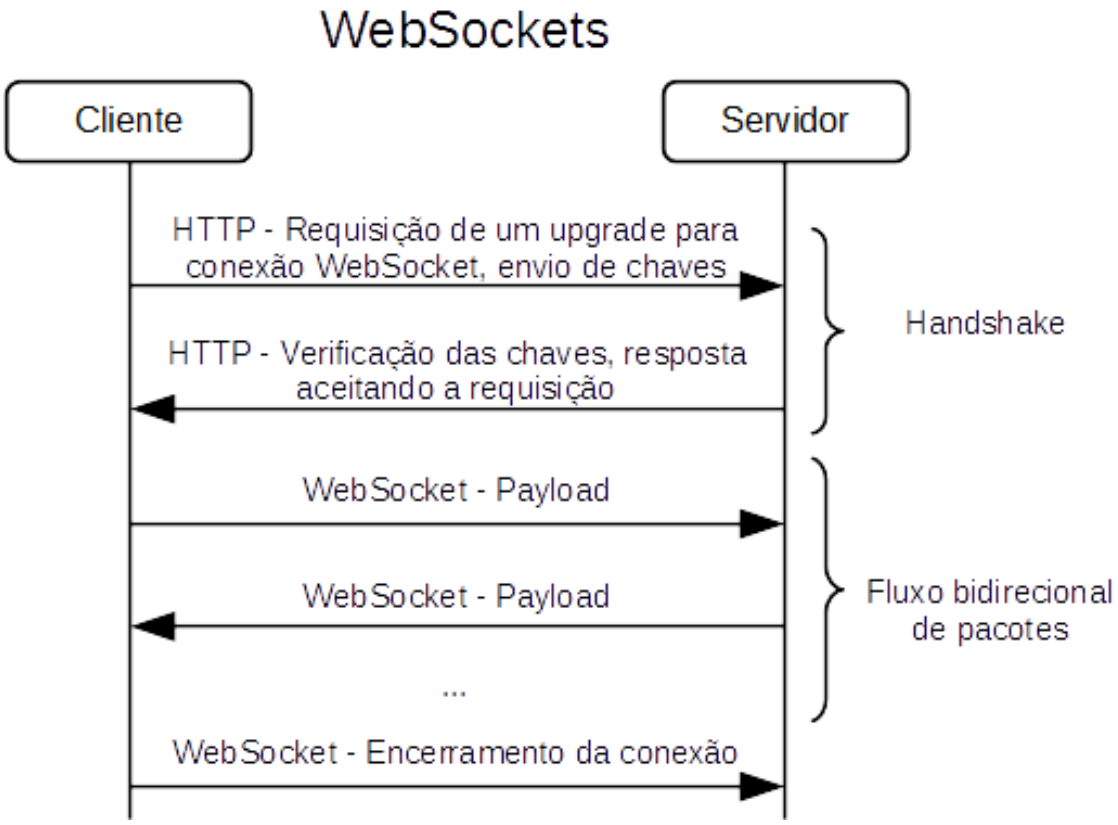

Figura 3: Diagrama ilustrando as fases/etapas do WebSocket.

2.5 BOAS PRÁTICAS E REUTILIZAÇÃO DE CÓDIGO – DESIGN PATTERNS

Esta parte do capítulo se dedica a apresentar um dos conceitos mais importantes para a fase de implementação deste trabalho, a de *Design Patterns*, que possibilitou realizar o projeto de tal forma que os produtos da fase de projeto sofressem o mínimo possível de alterações durante a implementação e testes. Esta seção é utilizada para fundamentação e faz referência em sua completude a obra de GAMMA (1995).

A ideia de usar padrões de desenvolvimento começou na área de arquitetura, com Christopher Alexander e alguns colegas que tiveram a ideia de usar uma linguagem padrão para

construir prédios e cidades. Essas ideias e outras contribuições acabaram chegando à área de software, especificamente ao Paradigma de Programação Orientado a Objetos.

Os *Design Patterns* surgiram da observação de que o cuidado com colaborações comuns entre objetos durante o desenvolvimento de um sistema gera uma arquitetura menor, simples e mais fácil de entender.

Um *Design Pattern* nomeia, abstrai e identifica os aspectos principais de uma estrutura comum que é útil na criação de projetos orientados a objeto reutilizáveis. A documentação e manutenção de um sistema também se torna mais fácil uma vez que os padrões fornecem uma especificação explícita das interações de classe e objeto.

Quando é feito um projeto, ao invés de ele ser específico para o problema ele pode ser suficientemente genérico para poder se encaixar em problemas e requisitos futuros, minimizando a necessidade de reprojetar e tornando as soluções mais flexíveis e elegantes.

Dos diversos padrões presentes no livro de GAMMA (1995) os que tiveram maior impressão neste TCC foram o *Singleton* e o *Strategy*.

2.5.1 SINGLETON

Singleton é um padrão relacionado à criação de objetos que objetiva assegurar que uma classe tenha apenas uma instância e provê um ponto global de acesso para ela. A solução é fazer com que a classe seja responsável por manter o controle da sua única instância, verificando as requisições para a criação de um novo objeto.

O uso desse padrão se aplica nos casos em que deve haver exatamente uma instância de uma classe e ela deve ser acessível por clientes de um ponto conhecido ou quando a instância única deve ser extensível por subclasses e os clientes devem ser capazes de usar uma instância estendida sem modificar seu próprio código.

Há vários benefícios em utilizar esse modelo. O acesso à única instância da classe é controlado, porque a instância é encapsulada e a classe *Singleton* tem controle sobre a forma como os clientes a acessam e quando isso ocorre. O *name space* é reduzido, uma vez que se evita o uso de variáveis globais que armazenam instâncias únicas. Esse modelo permite refinamento de operações e representação, uma vez que pode ter subclasses e é fácil de configurar uma aplicação com a classe estendida. Outra vantagem é que o programador pode alterar a quantidade de instâncias de forma simples, somente a operação que garante acesso à instância precisa ser alterada.

2.5.2 STRATEGY

Strategy ou *Policy* é um padrão relacionado ao comportamento de objetos e define uma família de algoritmos, encapsula cada um e os faz permutáveis.

O uso desse padrão é indicado no caso em que muitas classes relacionadas diferem apenas no seu comportamento: *Strategies* permitem que seja criada uma classe com um ou vários comportamentos. O mesmo se aplica se for necessário o uso de um algoritmo com algumas variantes hierárquicas ou quando o comportamento da classe é definido por declarações condicionais: uma classe *Strategy* simplifica o código. Outra aplicação é o caso em que um algoritmo usa dados sensíveis aos quais os clientes não devem ter acesso: *Strategy* evita a exposição de dados complexos ou específicos de um algoritmo.

Algumas vantagens do *Strategy* são a aplicação em famílias de algoritmos relacionados, uma alternativa a criação de subclasses - pois o encapsulamento do algoritmo em classes *Strategy* separadas torna a compreensão e manutenção do código mais fácil, é possível também eliminar declarações condicionais e permitir a escolha de diferentes implementações.

Algumas das desvantagens do *Strategy* são que o cliente tem que entender como as diferentes estratégias diferem antes de escolher a apropriada e o uso de *Strategy* aumenta o número de objetos na aplicação.

2.6 TRABALHOS RELACIONADOS

Um projeto relacionado à área de trabalho remota em navegador é o Guacamole, uma aplicação baseada em HTML5 (*Hypertext Markup Language 5*) e *VNC* que permite acesso a uma área de trabalho remota. É utilizado um servidor centralizado que age como túnel e proxy, permitindo acesso a múltiplos desktops através de um navegador web. O cliente, escrito em JavaScript, precisa apenas de um navegador com suporte a Ajax (*Asynchronous JavaScript and XML*) e *HTML5* e provê eventos de mouse e teclado via browser (JUMPER, 2013). Embora contemple conceitos importantes, o projeto Guacamole não possui uma camada de segurança.

O noVNC é outro projeto de cliente *VNC*, também baseado em *HTML5* que roda em qualquer navegador que suporte essa tecnologia, incluindo navegadores de *smartphones* (com sistemas operacionais Android ou iOS). Suporta cifragem por WebSocket TLS/SSL, permite o uso do cursor no sistema remoto ou local e cópia e colagem para a área de transferência (MARTIN; LEVY, 2013). O noVNC suporta, portanto, cifragem dos dados, se assemelhando à proposta deste trabalho e servindo de base para algumas ideias iniciais.

Tendo em vista a necessidade de acesso a sistemas remotos por meio de arquiteturas leves e com recursos que permitam personalização, Liao et al. (2010) expõe um sistema chamado LVD (*Lightweight Virtual Desktop*) que é constituído por um desktop virtual leve que suporta backup de dados, personalização, pode entrar e sair do estado suspenso, permite uso síncrono de aplicações em diferentes plataformas além de operar com baixo consumo de energia. O artigo expõe a arquitetura e comenta sobre algumas tecnologias chave. Uma delas é a possibilidade de usar modelos de imagens de disco com sistema operacional pré-instalado e o usuário escolhe a imagem que melhor lhe servir como base, criando uma cópia para si. A partir disso, ele pode acrescentar aplicativos e personalizar ao seu modo, o que gera uma imagem incremental para economizar espaço. Outra técnica interessante é o uso de múltiplos *checkpoints* que salvam todos os dados transientes e persistentes, compondo o estado de um ambiente em execução. O usuário pode acessar o sistema a partir de um *checkpoint* de acordo com seu trabalho realizado em diferentes locais, por exemplo.

Ainda na área de virtualização e acerca também da velocidade de atualização de tela durante uma transmissão, Ko et al. (2012) aplicam o protocolo *VNC* para acesso de *display* gráfico remoto. O artigo descreve a implementação de um protótipo de *VNC* para sistemas móveis baseado no "droid VNC", mas com algumas alterações para aumento da velocidade de atualização de tela. Na implementação do artigo codecs (codificador/decodificador) de vídeo foram integrados ao protótipo para verificar qual o mais adequado ao *VNC* móvel, pois técnicas de compressão de complexidade elevada não podem ser aplicadas devido as restrições de recursos. Ao mesmo tempo, a limitação de banda exige um mínimo de taxa de compressão. Na implementação, os *codecs* de áudio e vídeo do FFmpeg são portados e integrados via JNI (JAVA Native Interface) e cliente e servidor são conectados via TCP/IP (*Transmission Control Protocol/Internet Protocol*) sobre Wi-Fi. O *RFB* foi estendido para integrar os *codecs* de vídeo de tal forma que ainda haja retro compatibilidade. O protocolo é dividido em três fases: *handshaking*, inicialização e interação normal. No *VNC* o cliente é quem decide quando a tela é atualizada. Para melhorar o desempenho, nesta implementação o servidor faz o armazenamento de quadros mesmo sem a requisição do cliente, e quando ela ocorre, o servidor envia os quadros já disponíveis. O sistema foi modificado na região de codificação de forma que apenas as regiões modificadas entre uma imagem e outra são transmitidas. Verificou-se que o uso de MJPEG (*Motion Joint Photographic Expert Group*) é o mais adequado para o *VNC* móvel nos quesitos complexidade e taxa de compressão.

Com relação à transmissão de dados visuais de alta densidade pela Internet, algumas dificuldades surgem no que diz respeito a renderização de imagens e qualidade dos dados transmitidos devido à largura de banda. Wessels et al. (2011) tratam da necessidade de novas so-

luções para visualização baseada na web, como renderização remota. O artigo propõe uma arquitetura baseada em WebSockets e *HTML5* e delega o processamento para um servidor especializado. Este manda para o cliente as visualizações renderizadas de forma segura, e sem exigir um alto desempenho da máquina cliente. Citados anteriormente, o projeto é formado por duas partes: o servidor e o cliente. O servidor realiza o processamento e geração da imagem (usando GPU (*Graphics Processing Unit*)) e depois a envia para o cliente por um navegador. Dentro do servidor, o mecanismo de visualização é responsável pelo processamento das visualizações e o *daemon* pelas conexões dos clientes. Para otimizar a transmissão, a imagem só é mandada novamente ao cliente quando há algum evento que a modifica. O cliente precisa estabelecer uma conexão com o *daemon*, receber os dados da imagem e então mostrá-las para o usuário. Para tal é utilizado Canvas do *HTML5* (que permite que os dados de imagem sejam exibidos na tela), API (*Application Programming Interface*) do WebSocket (que permite transmissão de dados em tempo real) e codificação de imagem Base64 (que permite que o JavaScript use os dados como uma imagem no Canvas).

Também relacionado à transmissão de dados visuais via navegador, há o artigo de Chen e Xu (2011), que é um estudo de desempenho e factibilidade de um framework para jogos online multijogador baseados em *HTML5*, WebGL (*Web-based Graphics Library*), WebSocket e Three.js. O *WebGL* integra o OpenGL ES (*Open Graphics Library for Embedded Systems*) ao JavaScript, o que possibilita o uso de gráficos acelerados 3D nos navegadores compatíveis. WebSockets permitem comunicação bidirecional, *full-duplex* sobre um canal *TCP* entre cliente e servidor. O Three.js provê uma *API* simples para utilização de WebGL. É utilizado jWebSocket: a implementação WebSocket para Java e JavaScript. A arquitetura é baseada em cliente, *game server* (que é o servidor de jogo rodando em modo dedicado) e web server, que disponibiliza arquivos, páginas web, etc. O cliente utiliza *Web Workers* para otimizar a comunicação com o *web server*, enquanto jWebSocket é utilizado para comunicar-se com o *game server*.

Dentro do tema de transmissão de dados, Pimentel e Nickerson (2012) comparam WebSockets, *Polling* e *Long Polling* na transmissão de dados. São tratados temas como a diferença entre comunicação *hiperlink* e com troca de informações sem estado (*HTTP*) para uma com troca de informações em tempo real e com estado entre cliente e servidor; e a latência gerada pela sobrecarga na troca de informações em alguns tipos de aplicação. Tradicionalmente a técnica utilizada é *HTTP polling*, uma sequência de *request-response*. O cliente envia uma requisição ao servidor, caso haja uma nova mensagem ela é enviada, caso contrário uma mensagem vazia é enviada. Requisições são feitas com intervalos de tempo fixos. Pode ser útil para taxa de transmissão constante, mas conforme a taxa aumenta, a sobrecarga inerente ao *polling* implica num aumento de latência. Na variante *long polling*, ao invés do servidor enviar uma mensagem

vazia logo em seguida a chegada de uma requisição, o servidor espera por uma nova mensagem ou até um tempo limite expirar. Isso reduz o número desnecessário de requisições feitas pelo cliente, enquanto não existe mensagem nova, emulando o envio de informação direto do servidor para o cliente. A mais nova solução é WebSocket, tecnologia já citada anteriormente. Apesar de ser um *draft* em trabalho da W3C, é estimado que o protocolo reduzisse em 1/3 a latência contra o *HTTP Polling* em *half-duplex*. Para a comparação entre as técnicas, Pimentel e Nickerson (2012) configuram uma estação base que recebe dados de um sensor de vento por comunicação serial e os disponibiliza através de um servidor, que aceita comunicação nas três formas supracitadas. O que pode se perceber é que *Long Polling* algumas vezes atuou de forma quase tão eficiente quanto WebSocket. Por outro lado, excetuando-se uma vez, *Polling* sempre teve um desempenho pior.

FUKAI et al. (2012), por sua vez, introduziram uma *GUI* implementada em navegador web para TVs inteligentes com comunicação por WebSockets. Funções como obtenção do canal de TV atual e programação, obtenção de EPG (*Electronic Program Guide*) e adição de agendamento de gravação se tornam possíveis como serviços web. As alternativas empregadas envolvem o acesso do JavaScript a objetos especiais via *plug-in* NPAPI (*Netscape Plugin Application Programming Interface*), que faz chamadas a *getters* e *setters* do *middleware* da TV e *Extended Scripting Objects*: objetos especiais adicionados diretamente ao JavaScript, podendo ser melhor que o esquema de plug-in, mas limitado em alguns motores de JavaScript. Segundo os testes, a sobrecarga causada pelos WebSockets foi pequena, independente do poder de processamento da TV. Contudo, a acessibilidade pode ser um problema com o uso de WebSockets, devido a padronização ainda estar em andamento.

Zhu et al. (2012) fazem uma crítica a reprodutores de mídia baseados em tecnologias como *plug-ins* em Flash Player ou ActiveX devido a problemas de instalação, manutenção e falhas de segurança. O artigo descreve a implementação de um reprodutor de vídeo, que recebe dados por WebSockets e os mostra através do Canvas do *HTML5*, depois de decodificado nativamente por um *decoder* H.264 em JavaScript, sem a adição de *plug-ins*. No projeto há quatro módulos. No módulo de recepção está o WebSocket. No módulo de decodificação um codec H.264 é escrito em JavaScript, que permite a renderização pelo navegador. No módulo de renderização está o Canvas do *HTML5*, o que permite ao JavaScript pintar uma imagem no navegador sem a adição de plug-ins; o Canvas permite a alteração pixel a pixel de cores e se esse processo for rápido o suficiente, a exibição das imagens será semelhante à de um vídeo. No módulo de controle estão os *Web Workers*, que permitem a execução de scripts em plano de fundo sem serem interrompidos. Isso é necessário, pois o JavaScript tem uma limitação de que todo o processo é executado numa única thread. Resultados experimentais mostram que

é possível reproduzir vídeos de 640x360 a mais de 30 quadros por segundo. Nota-se que a implementação deste artigo sobrecarrega a CPU (*Central Processing Unit*). Isso poderia ser melhorado pelo estudo de WebGL, passando parte da carga para a *GPU*, como em Wessels et al. (2011).

Outro trabalho interessante é o de Pérez (2011) que objetiva criar um canal de comunicação sem a necessidade de instalação de um componente de software adicional, que exija a aceitação de um acordo ou compartilhamento de dados pessoais. É implementado um compartilhamento sem criptografia, sem interação remota, entre dois usuários: um compartilhando e outro vendo a transmissão, tudo sobre WebKit. Os protocolos de compartilhamento de área de trabalho apresentados são: *RFB*, NX, Xpra e RDP. Três *web standards* foram apresentados: *HTML5*, WebSocket e XMLHttpRequest. Foi escolhido o GStreamer, um *framework* multimídia baseado em *codecs* em *pipeline* que permite um fluxo multimídia, e uso do WebKit como motor de navegador web, utilizado em vários softwares, como Safari, Chrome e Android web browser. O aplicativo desenvolvido obtêm imagens da tela a partir do GStreamer, utiliza WebSockets para comunicação entre os *peers* e um servidor que envia dados de um usuário para outro. Foi escolhido trabalhar com envio de imagens ao invés de instruções de desenho devido à alta complexidade. Além disso, o envio de imagens é mais seguro em caso de perda, pois apenas parte da imagem não estará disponível, enquanto a perda de instruções impossibilita a exibição de qualquer imagem. Com relação a comunicação, o protocolo *VNC* foi escolhido, por estar bem documentado e atender os requisitos. Para atender à exigência de não utilizar *plug-ins* ou instaladores adicionais, o uso de WebSockets entrou como solução. Para o formato de imagens, o uso de JPEG ao invés de PNG (*Portable Network Graphics*) foi eleito baseado em problemas de ordem prática com o GStreamer. Para codificação foi utilizado Base64, que embora "aumente" o tamanho da imagem, causando diminuição da velocidade com que elas são mostradas no cliente, foi utilizada sem problemas no GTK browser em detrimento da codificação BLOBs (*Binary Large Object*). Um servidor é utilizado apenas para fins de autenticação, que depois de bem sucedida permite a conexão entre as duas pontas, associadas a um identificador. A sequência de imagens é disponibilizada como uma imagem HTML normal, possível pelo JavaScript, que suporta a imagem em Base64. O programa mostra imagens a cada 1 segundo, devido ao principal uso ser para mostrar slides ou semelhantes, sem interação do cliente.

A Tabela 1 apresenta um comparativo entre os trabalhos relacionados que foram levantados nesta fase do projeto. Ela evidencia quais os trabalhos relacionados a cada assunto de interesse:

Desktop Remoto : há alguma referência a protocolos de área de trabalho remota.

Codecs : há alguma referência a técnicas de codificação e decodificação de dados.

Comunicação : há alguma referência a protocolos de comunicação.

WebGL : há referência à utilização de WebGL para uso de GPU em código JavaScript.

Web Workers : há referência à utilização de *Web Workers* para paralelização de código JavaScript.

Cliente-Servidor : há alguma referência à arquitetura Cliente-Servidor.

Plug-ins : há referência à implementação de plug-ins para navegadores web.

Imagem : há referência sobre compressão ou formato de imagem.

Video : há referência sobre compressão ou formato de vídeo.

Captura de Imagem : há referência à captura de imagem da área de trabalho.

HTML5 : há referência à utilização de HTML5.

Tabela 1: Tabela comparativa dos trabalhos relacionados levantados.

Autores	Liao et al. (2010)	Wessels et al. (2011)	Chen e Xu (2011)	Pimentel e Nickerson (2012)
Desktop Remoto	✓	✓		
Codecs				
Comunicação		✓	✓	✓
WebGL			✓	
Web Workers			✓	
Cliente-Servidor		✓	✓	
Plug-ins				
Imagem		✓		
Video				
Captura de Imagem				
HTML5		✓	✓	
Autores	FUKAI et al. (2012)	Ko et al. (2012)	Zhu et al. (2012)	Pérez (2011)
Desktop Remoto		✓		
Codecs		✓	✓	✓
Comunicação	✓	✓	✓	✓
WebGL			✓	
Web Workers			✓	
Cliente-Servidor		✓		✓
Plug-ins	✓		✓	
Imagem		✓		✓
Video		✓	✓	
Captura de Imagem				✓
HTML5			✓	✓

2.7 CONSIDERAÇÕES FINAIS DO CAPÍTULO

A fundamentação apresentada neste capítulo dá alguns detalhes referentes a decisões de abordagem do problema proposto pelos objetivos deste trabalho, principalmente no que tange a escolha de protocolos bem definidos, incentivado pela arquitetura de cliente-servidor da aplicação, o que obriga a existência de uma padronização da comunicação entre as partes. Além disso, eles também trazem uma grande vantagem à implementação do sistema, nos quais as partes se tornam independentes da implementação uns dos outros, como ocorreu neste projeto, no qual cliente e servidor puderam ser implementados de forma independente devido a existência de um "contrato" bem definido entre si, além da independência de todo o protótipo com a estratégia de segurança.

A escolha do protocolo de área de trabalho remota foi guiada pela simplicidade e também pela existência de ampla documentação em detrimento de funcionalidades, o que levou a escolha do protocolo RFB. Na área de segurança a decisão foi feita com base no ambiente no qual a aplicação foi implantada, a de navegadores web, nos quais domina o uso de TLS/SSL.

O uso de navegadores web também implica em decisões de abordagem do desenvolvimento da aplicação. As funcionalidades fornecidas para as aplicações web foram avaliadas como suficientes para o lado cliente deste trabalho. Entretanto, elas eram muito restritivas para o servidor, que necessita comandos diretos com o sistema operacional da máquina em que está implantado, o que obrigou uma abordagem mais tradicional para o desenvolvimento, resultando num programa a parte, dedicado ao papel de servidor de área de trabalho remota.

Ao final do levantamento de trabalhos relacionados algumas importantes estratégias de abordagem do problema foram obtidas, e impactaram nas decisões de projeto tomadas ao longo do TCC.

A escolha de uma sequência de imagens com compressão em detrimento de um algoritmo de compressão de vídeo ocorreu com base nos seguintes argumentos: o consumo de processamento por um *codec* de vídeo, como MPEG ou H.264, tradicionais em vídeos de alta definição, é maior e mais desvantajoso do que um algoritmo mais simples aplicado sobre imagens individuais (KO et al., 2012), além da complexidade de implementação de um desses *codecs*. Além disso, o uso de uma compressão de imagem mínima já é um ponto de partida para redução da carga de banda. O uso de imagens também abre a possibilidade de aplicar codificação de região como feito por Ko et al. (2012) e diminuir a banda utilizada.

A aplicação foi desenvolvida para ser executada sobre navegadores web, sem a necessidade de o lado cliente ter que instalar componentes adicionais em sua máquina. Esse fato

foi determinante na escolha das linguagens de programação: HTML5 e JavaScript. Apesar deste conjunto de linguagens ser utilizado para a criação de páginas web, cada uma delas apresentou características de interesse para os objetivos do trabalho. O HTML5, em comparação com as suas versões anteriores, trouxe diversas melhorias com relação a conteúdo multimídia, mas essas mudanças ainda não foram suficientes para alcançar o dinamismo necessário para o compartilhamento e visualização de área de trabalho. Para tal, a adição de JavaScript para a manipulação das páginas HTML5, principalmente do componente Canvas, e de captura de eventos de mouse e teclado, foi necessária. Entretanto essas duas linguagens não são suficientes para garantir uma comunicação eficiente, por isso a necessidade do uso de WebSockets, dadas as vantagens do protocolo e das limitações do HTTP. Para garantir a segurança, o uso de TLS/SSL com o protocolo WebSocket é aplicável em vários navegadores web. Isso tirou a necessidade de implementar um módulo de criptografia extra.

3 ESTUDO DO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

O método de escolha para o levantamento de requisitos para o plano de projeto de TCC do curso de Bacharelado em Engenharia de Computação da UTFPR é por Casos de Uso. Apesar da profunda utilidade e poder do UML (*Unified Modelling Language*), que inclusive foi de grande importância para o projeto, como pode se constatar pelos apêndices deste relatório, foi percebida a necessidade de uma metodologia consistente para alcançar uma listagem de requisitos que melhor satisfizessem os objetivos propostos para este TCC. Este capítulo é dedicado para apresentar em detalhes a solução escolhida como procedimento de desenvolvimento de software numa tentativa de melhor aproveitar o potencial das ferramentas fornecidas pelo UML.

3.1 AS FASES DO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

Em geral, um projeto de software pode ser dividido em uma sequência de fases. A devida execução de cada etapa é vital para garantir que o problema, definido na fase inicial da análise, seja devidamente resolvido pela solução e validado ao final do projeto. A lista a seguir descreve brevemente cada uma das partes de um projeto genérico, que correspondem em grande parte às do processo de desenvolvimento utilizado neste trabalho. São as partes:

- Análise: definição de requisitos - descreve o que o sistema deve fazer. Cooperação entre desenvolvedores e usuários.
- Projeto: define a arquitetura do sistema. Representa as funções do sistema de uma forma que elas possam ser transformadas em programas.
- Implementação: transforma o projeto em unidades de programa.
- Validação: consiste em testes e revisões. Demonstrações também são possíveis.
- Serviço e manutenção: remoção de erros, melhoria do produto.

Esta divisão em fases é simples e abstrai muitos dos detalhes e peculiaridades de cada trabalho, mas coincide em boa parte com muitos dos projetos de software. Essa é, inclusive, uma estrutura muito semelhante ao processo de desenvolvimento ADIT, que é apresentado com mais detalhes ao longo deste capítulo.

3.2 PROCESSO DE DESENVOLVIMENTO ADIT

O processo de desenvolvimento ADIT (*Analysis, Design, Implementation, Testing*) foi proposto por Winston W. Royce (1929–1995). É um método iterativo e evolutivo, e decorre da constatação de que todos os artefatos de análise e projeto são parciais e evoluem em resposta a novas descobertas (LARMAN, 2007).

A fase que exige mais atenção é a análise, pois ela trata da descoberta de conceitos (ou objetos) do domínio do problema que definem as decisões durante as outras fases do projeto, gerando documentos que possibilitam a compreensão do comportamento do software. Alguns dos artefatos gerados durante a fase de análise são: casos de uso, modelos conceituais, glossários, diagramas de sequência, contratos de operação e diagramas de estado.

Apesar de cada fase apresentar suas peculiaridades, o processo todo é acompanhado pela elaboração de uma lista de termos relevantes e suas definições, chamada de glossário ou dicionário de dados, que serve como complemento dos modelos construídos pelo ADIT.

No decorrer desta parte do capítulo cada uma das fases é detalhada e acompanhada da descrição de cada um dos passos que as compõem.

3.2.1 *ANALYSIS* – A FASE DE ANÁLISE

Sob o ponto de vista do ADIT, o objetivo de um projeto é a construção de uma máquina (*machine*) – software executado sobre um hardware – com características específicas. Esta máquina é integrada a um ambiente (*environment*), parte do mundo real relevante para o problema, normalmente com propriedades fixas.

Máquina e ambiente podem interagir por meio de fenômenos (*phenomena*) compartilhados – ações ou operações que ocorrem no ambiente. Uma máquina interage com o ambiente observando certos fenômenos na entrada (*input*) e causando outros na saída (*output*). Eles são importantes para expressar declarações (*statements*).

As características do ambiente compõem o conhecimento do domínio e as características desejadas do sistema são os requisitos. Em outras palavras, a forma como o ambiente deve

ser depois da máquina ter sido integrada ao sistema.

Todos os conceitos levantados anteriormente são fundamentais para o desenvolvimento de um projeto em ADIT, principalmente na fase de análise, que pode ser dividida em seis passos: o passo A1 engloba a levantamento e descrição do problema; o A2, a decomposição e classificação do problema; A3, a especificação abstrata de software; A4, a especificação técnica de software; A5, especificação de dados de controle e operações; e A6, o ciclo de vida do software.

No Passo A1, chamado de levantamento e descrição do problema, os objetivos do projeto são declarados de forma textual, a partir dos quais são gerados diagramas de contexto. Sobre os diagramas são feitas as distinções entre ambiente e máquina, a representação das conexões entre eles, identificação dos domínios relevantes e de seus fenômenos compartilhados. Os domínios são representações de classes conceituais do mundo real, não de objetos de software (LARMAN, 2007). Resumindo: é representado o mundo quando a máquina está em operação.

Nos diagramas de contexto são usados domínios de quatro tipos:

- *Biddable domain*: geralmente é a representação de uma pessoa – é um domínio físico mas não possui causalidade interna e é sempre ativo;
- *Causal domain*: aquele cujas relações entre seus fenômenos são causais, normalmente representa um equipamento elétrico ou mecânico no mundo real;
- *Lexical domain*: representação física de dados, geralmente passivo;
- *Display domain*: domínio causal, serve como um dispositivo de saída para a máquina e provê informação aos outros domínios.

No passo A2, chamado de decomposição e classificação do problema, é tomado um problema e dividido em subproblemas mais simples. Uma forma prática de decomposição é a projeção, na qual se ignoram aspectos parciais e o resultado é composto por subproblemas paralelos não hierárquicos, com o conhecimento de classes de problemas e suas soluções. Essas classes são caracterizadas através de esqueletos de problemas (*problem patterns*), que contém padrões para os domínios envolvidos e seus fenômenos compartilhados.

Os cinco tipos mais comuns de esqueletos são:

- Comportamento requerido: construção de uma máquina que irá impor um controle a alguma parte do mundo físico para que satisfaça certas condições.

- Comportamento comandado: construção de uma máquina que irá aceitar comandos do operador e impor um controle a outra parte do sistema físico que deve ser controlada por esses comandos.
- Exibição de informação: construção de uma máquina que obtém informação do mundo (estados e comportamento de alguma parte do sistema físico), apresentando-a no local adequado e na forma requerida.
- Peças simples: construção de uma máquina que pode atuar em uma ferramenta que permite ao usuário criar e editar certas estruturas para que possam ser usadas de várias maneiras.
- Transformação: construção de uma máquina que irá produzir as saídas requeridas de acordo com entradas.

Para resolver um subproblema é desejável que ele se encaixe no esqueleto, ou seja, represente uma instância do diagrama modelo. Esse método contribui com outros tipos de decomposição, como *top-down* (arranjo de funções hierarquicamente) ou Casos de Uso (análise orientada a objeto).

A partir dos diagramas de contexto elaborados na fase anterior são feitos diagramas de problema. Eles são projeções dos diagramas de contexto e contém requisitos que se referem (*refer to*) e restringem (*constrain*) os domínios do problema.

No passo A3, chamado de especificação abstrata de software, ocorre a derivação de especificações – descrições suficientes para construir a máquina, também caracterizadas como requisitos implementáveis. Antes de dar continuidade vale ressaltar a definição de quatro conceitos até aqui apresentados:

- Requisitos são as características desejáveis do sistema (máquina e ambiente).
- Especificações são requisitos implementáveis, ou seja, fazem relação à máquina.
- Fatos são características intrínsecas do ambiente.
- Suposições são as características que o projeto supõe que o ambiente tenha para que os requisitos do projeto sejam alcançáveis.

Por consequência, o requisito nada mais é do que a junção das características da máquina com as do ambiente, ou seja, o conjunto de especificações, fatos e suposições. A partir do momentos que estas três características são atendidas, o requisito está cumprido.

De forma a representar as especificações, podem ser usados diagramas de sequência – eles organizam interações (mensagens trocadas) entre objetos ao longo do tempo e são usados para modelar comunicação do ambiente com a máquina.

Diagramas de sequência são usados em um cenário específico de um caso de uso (LARMAN, 2007). As dependências de subproblemas são explicitadas via predicados de estado – determinam como os subproblemas interagem e como eles devem ser compostos para constituir o problema completo. Os diagramas de sequência que representam especificações são usados posteriormente para a composição de testes que, quando atendidos, demonstram o atendimento das especificações pela máquina.

No passo A4, chamado de especificação técnica de software, o objetivo é a descrição da infraestrutura técnica na qual a máquina é executada. As tecnologias utilizadas são introduzidas como domínios de conexão e os fenômenos técnicos são usados para refinar os fenômenos do diagrama de contexto abstrato da fase A1.

Inicia-se o processo coletando os resultados da decomposição em subproblemas (novos domínios e fenômenos). Em seguida introduzem-se os domínios de conexão necessários e identificam-se as interfaces técnicas que precisam ser consideradas – fenômenos técnicos como, por exemplo, pacotes TCP/IP e comandos SMTP devem ser explicitados. Por fim é descrito o fenômeno técnico em detalhes ou é feita referência a uma especificação técnica existente.

No passo A5, chamado de especificação de dados de controle e operações, o objetivo é dar descrições detalhadas dos serviços que a máquina, como caixa preta, deve oferecer na interface pública (LARMAN, 2007). Esses serviços (operações) correspondem a mensagens direcionadas para domínios de máquina nas especificações estabelecidas na fase A3. A descrição de uma operação consiste em uma pré-condição e uma pós-condição. É necessário também modelar os dados que devem ser mantidos pela máquina para realizar as operações – para isso utilizam-se modelos de classe.

Um modelo de classes de máquina de estados mostra as classes que devem ser implementadas. Utiliza-se UML, que é um padrão industrial que pode ser enriquecido com especificações de restrições de estado e semântica de operação com a utilização da linguagem formal OCL (*Object Constraint Language*) (LARMAN, 2007).

O princípio de *design by contract*, utilizado nesta etapa, torna explícitas as obrigações e direitos dos usuários e provedores de serviços: basicamente o "contratante" de um serviço (método/procedimento) precisa garantir que a pré-condição seja satisfeita, o "prestador de serviço", por sua vez, precisa garantir que a pós-condição seja satisfeita.

A especificação de operações é usada para detalhar os fenômenos identificados na fase A3, de acordo com o *design by contract*. Os modelos de classes são desenvolvidos em paralelo com as especificações de operação.

No passo A6, chamado de ciclo de vida do software, o objetivo é a descrição do comportamento geral da máquina (protocolo de uso) e a restrição da ordem em que as operações podem ser executadas – utilizam-se expressões de ciclo de vida. É estabelecido um ciclo de vida para cada *biddable domain*. Essas expressões são utilizadas para expressar relações entre diagramas de sequência – tanto sequencialmente quanto em paralelo.

Os ciclos de vida para cada *biddable domain* (atores) são estabelecidos para prover uma visão da máquina para cada ator. Os modelos de ciclo de vida também são importantes para projetar e testar a interface de usuário.

3.2.2 *DESIGN* – A FASE DE PROJETO DA MÁQUINA

Até então foram trabalhadas a apresentação, estrutura e classificação de problemas. A partir deste ponto, inicia-se a estruturação e classificação das soluções de subproblemas. Assim como a Análise, o Projeto é dividido em passos, quatro mais especificamente: o passo D1 se dedica a definição da arquitetura de software; o D2, das interações inter-componente; D3, das interações intra-componente; e D4, do componente completo ou comportamento de classe.

No passo D1, chamado de arquitetura de software, as máquinas que serão construídas para resolver um problema de desenvolvimento de software precisam ser estruturadas. Essa estrutura chama-se arquitetura lógica e é resultado do projeto. A arquitetura estrutura a máquina em termos de componentes (que realizam cálculos) e conectores (a interação entre componentes).

Os estilos de arquitetura classificam sistemas de software e a arquitetura de qualquer máquina deve ser uma instância de um desses estilos. Geralmente vários estilos de arquitetura são possíveis e todos podem ser usados para implementar os requisitos funcionais. Sua seleção pode ser feita de acordo com critério não funcional. Alguns exemplos de estilos de arquitetura são:

- *Pipes-and-filter*: os *pipes* (tubos) movem dados de um filtro de saída para um filtro de entrada e os *filters* (filtros) transformam fluxos de dados de entrada em fluxos de dados de saída de maneira incremental – podem ser usados para resolver problemas que encaixam no esqueleto de problema de transformação;
- Repositório: o objetivo é a integração dos dados, a máquina pode ser descrita por um

repositório que é utilizado e alterado pelas diversas partes – podem ser usados em problemas que encaixam no esqueleto de busca e atualização.

- Em camadas: organização hierárquica de software – camadas mais baixas provem serviços para camadas mais altas (LARMAN, 2007) e geralmente apenas camadas adjacentes são conectadas – referência ao protocolo de comunicação ISO/OSI.

No passo D2, chamado de interação inter-componente, o objetivo é a descrição do comportamento dos componentes de software contidos na arquitetura do software, o estabelecimento do fluxo de comunicação entre componentes de software para realizarem uma operação e a especificação dos casos de teste. A ordem das operações definidas no passo A5 é representada por diagramas de sequência, conforme os componentes levantados pelo passo D1. Isso define a sequência de declarações que envolvem chamadas de métodos em outros componentes para a implementação da máquina.

No passo D3, chamado de interação intra-componente, as operações mais complexas da fase A5 são mapeadas em interações entre subcomponentes da arquitetura interna do componente. As interações entre os subcomponentes precisam realizar o comportamento interno dos componentes descrito no passo D2. Este processo, assim como a fase anterior, resulta em novos diagramas de sequência.

No passo D4, chamado de componente completo ou comportamento de classe, o objetivo é completar a descrição comportamental dos componentes de software ou classes contendo mais que dois estados. É um projeto detalhado que pode ser transformado em implementação diretamente e também pode ser usado para geração de casos de teste automáticos.

As máquinas de estado provêm uma notação rica para especificar comportamento. Elas precisam representar o comportando descrito nas fases anteriores de projeto e suas transições devem ser consistentes com o modelo de ciclo de vida.

3.2.3 *IMPLEMENTATION AND TESTING* – AS FASES DE IMPLEMENTAÇÃO E TESTES DE SOFTWARE

O propósito desta etapa é transformar o projeto em uma implementação que respeita às especificações. É relativamente simples, uma vez que a maioria das decisões mais complexas do projeto ocorreu na fase de Projeto. As bases para a implementação e testes são as etapas A5, A6, D1, D2 e D4.

A fase de implementação é seguida pela de testes, na qual são realizadas a validação e verificação do software com o objetivo de detectar defeitos. É importante ressaltar que esta

etapa possibilita a confirmação da presença de defeitos, mas não da sua ausência.

Os testes de software procedem em várias fases. No teste unitário, uma possível entrada é dividida entre classes de equivalência, das quais os casos de teste são tomados. É realizado um teste de caixa preta (ou teste funcional), cujos casos de testes são gerados unicamente baseados na especificação. Pode ser realizado também um teste de caixa branca (ou teste estrutural), cujos casos de teste são gerados também com base na estrutura do componente a ser testado.

Os testes de integração examinam se os componentes cooperam de acordo com a arquitetura e reduzem o número de *stubs* a serem implementados. Podem ser realizados o método de integração *top-down*, no qual os componentes são testados do nível mais alto ao mais baixo, ou *bottom-up*, do nível mais baixo ao mais alto.

No ADIT os procedimentos das fases de Teste são derivados das fases de Análise e Projeto e executados sobre os resultados da fase de Implementação. Portanto, os passos de teste são diretos e relativamente simples, sempre retornando a fase de Implementação quando necessário. Os passos da fase de teste são:

- T1 – Testes de componente: devem ser criados para cada componente da máquina. Um ambiente de componente é simulado com a utilização de *stubs*, ou uma base de dados preparada com dados de teste. O componente é estimulado por chamadas de operações, com parâmetros concretos e de acordo com as classes de equivalência.
- T2 – Testes de sistema: devem ser criados para a máquina integrada. Eles simulam as interfaces externas da máquina.
- T3 – Testes de aceitação: compreendem a implantação, o sistema em uso, usabilidade, e testes de desempenho. Testes de aceitação devem ser realizados no domínio da aplicação e podem ser usadas *checklists* para tornar os testes replicáveis.

3.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Um dos pontos importantes deste capítulo é a compatibilidade do ADIT com as necessidades encontradas para o desenvolvimento do trabalho, pois o ADIT é trabalhado de maneira incremental e iterativa: a cada iteração alcançada durante o desenvolvimento de software adquire-se mais conhecimento relacionado ao domínio.

O problema nas fases iniciais do trabalho eram compreender o problema, sua inserção no ambiente e como proceder para o desenvolvimento.

Além disso, a escolha desse processo em especial não foi por acaso, ele é especialmente indicado no desenvolvimento de sistemas embarcados. Pode não ser o caso deste TCC, mas é uma necessidade recorrente ao longo do curso de Engenharia de Computação da UTFPR Campus Curitiba.

4 METODOLOGIA

O presente capítulo apresenta uma visão geral do processo de desenvolvimento deste TCC, com a sequência do desenvolvimento do trabalho, descrita pela metodologia de trabalho, e os fundamentos necessários, descritos nos recursos de hardware e software utilizados no desenvolvimento da aplicação.

O projeto como um todo seguiu o processo de desenvolvimento de software ADIT, que está descrito em mais detalhes no decorrer do capítulo 3. O processo foi auxiliado com a produção de pequenas provas de conceito para avaliar a viabilidade das tecnologias escolhidas, executadas numa ordem em que possibilitou a junção gradual de módulos e a verificação do funcionamento das tecnologias em paralelo. Isso ajudou no ganho de familiaridade com as tecnologias e no levantamento de especificações do ADIT.

A sequência de desenvolvimento adotada foi: estudo detalhado dos fundamentos e das opções de tecnologia, descrito no capítulo 2, seguido do desenvolvimento de provas de conceito em variadas áreas, sendo as principais: verificação do funcionamento de WebSockets sobre TLS/SSL, utilização do Canvas do HTML5 e transmissão de eventos de mouse e teclado pelo canal de comunicação. Após esses passos, foi concebido um meio de obtenção de imagens da área de trabalho. Este passo foi importante, pois, dados os trabalhos relacionados encontrados, o desenvolvimento de um *plug-in* ou uso de um programa a parte para a captura de tela foi cogitado, o que seria desnecessário para o cliente, mas essencial para o servidor. A decisão final foi a de retirar o papel do servidor de dentro do navegador web e colocá-lo num programa a parte implementado em Java. Após a realização dos módulos, restou a integração do cliente numa página web única, que contém a aplicação web. Um último passo necessário para a integração foi a verificação da necessidade de otimização do desempenho da aplicação, possível no cliente através da utilização de *Web Workers*. Essa alteração removeu a limitação do código JavaScript ser executado sobre uma única thread disponibilizada pela interface gráfica do navegador web (ZHU et al., 2012), o que causava travamento da GUI em caso de longos processamentos feitos pelo script. Adquiridos os conhecimentos suficientes em torno das tecnologias, o trabalho entrou no processo ADIT, ao final do qual foi obtido o protótipo do sistema proposto como

objetivo.

A validação do ponto de vista de segurança da aplicação fica por conta do uso de uma ferramenta de *benchmarks* de redes de computadores capaz de obter pacotes TCP/IP que correm pela rede independentemente do seu destino. Isso serve para mostrar que, mesmo em mãos dos pacotes de dados trocados pela aplicação através da rede, eles não são de uso imediato para o atacante, uma vez que os dados por eles transportados estão cifrados.

Como forma de visualização geral do projeto, a Figura 4 dá uma ideia dos blocos que compõe o sistema e de algumas das tecnologias utilizadas. O diagrama de blocos mostra, de forma resumida e em alto nível, como o sistema se estrutura. Basicamente o cliente executa uma aplicação web, baseada em HTML5 e código JS (JavaScript), através do navegador web. A aplicação, por sua vez, envia para o servidor, através de uma conexão WebSocket, mensagens codificando eventos de mouse/teclado. O servidor emula esses eventos e retorna o resultado através da captura de tela da área de trabalho.

4.1 TECNOLOGIAS E RECURSOS DE HARDWARE E SOFTWARE

Esta seção está focada na listagem e aprofundamento dos recursos necessários para a execução do projeto sob os aspectos de recursos e tecnologias.

Os recursos de hardware não se aplicam, uma vez que este trabalho não envolveu o projeto e desenvolvimento de um protótipo de hardware; foram utilizados equipamentos apenas para implementação, execução e teste do software gerado. Por outro lado, nos recursos de software haviam necessidades em diversos pontos: implementação e utilização dos protocolos RFB, WebSocket e SSL/TLS, implementação de Servidor Web HTTP, comunicação de eventos de periféricos com o SO e utilização de técnicas de programação específicas para o JavaScript.

4.1.1 O AMBIENTE DE EXECUÇÃO E TESTES

A arquitetura do sistema deste TCC necessita de dois computadores conectados em rede: um deles para o cliente - que execute um navegador com suporte a HTML5, JavaScript, WebSockets e SSL/TLS; e o outro para o servidor - que possua uma JVM (*Java Virtual Machine*) atualizada.

Para fins de desenvolvimento e simplificação da implantação dos testes, VMs (Máquinas Virtuais) podem ser utilizadas para executar todo o protótipo dentro de uma única máquina física. Existem diversas soluções que atendem essas necessidades, que vão desde soluções gratuítas, como as fornecidas pela Oracle (2014e) através do VirtualBox, ou produtos pagos, como

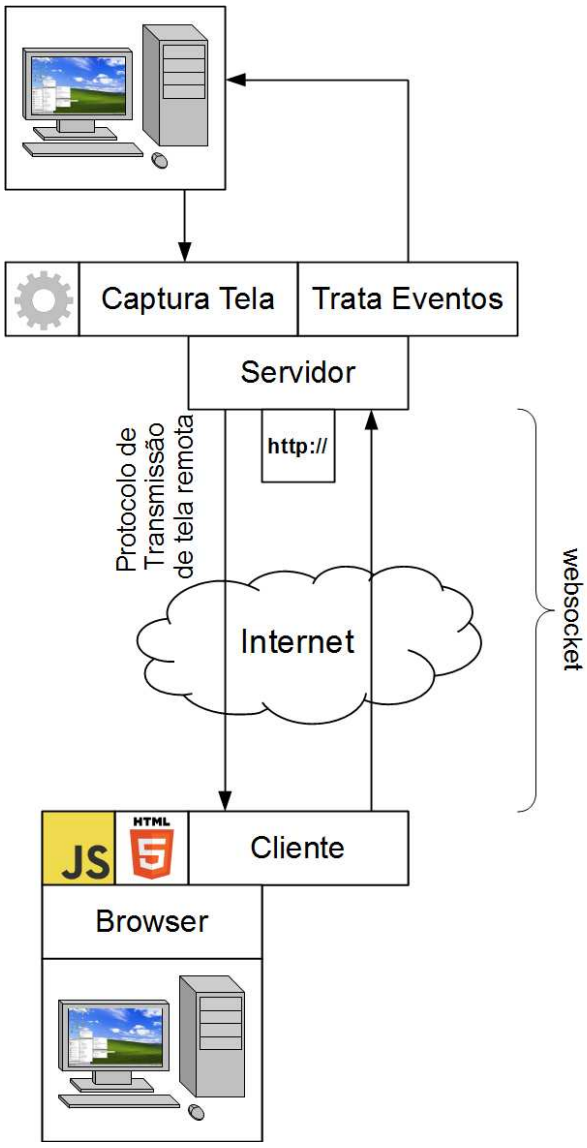

Figura 4: Diagrama simplificado de funcionamento do projeto.

o VMWare Fusion (VMWARE, 2014).

4.1.2 O AMBIENTE DE DESENVOLVIMENTO

Com o intuito de facilitar a implementação, a utilização de um IDE (Ambiente de Desenvolvimento Integrado) para páginas web foi cogitado, levando em conta a capacidade de trabalhar com HTML5, JavaScript e CSS. Nesse quesito Damm (2013) fez uma tabela comparativa entre vários IDEs e, excluídos os produtos comerciais, o Netbeans foi o que apresentou mais funcionalidades e melhor desempenho. Por outro lado, a leveza e simplicidade do Eclipse, que é suportado por diversos plug-ins que cobrem funcionalidades não implementadas originalmente, toranaram-o o IDE candidato para o desenvolvimento do trabalho.

4.1.3 A IMPLEMENTAÇÃO DE UMA BIBLIOTECA RFB

O RFB é um protocolo com uma boa documentação e ampla utilização por parte das mais diversas implementações (REALVNC, 2014)(TIGHTVNC, 2014). Entretanto, mesmo com essas características a seu favor, as buscas por uma solução pronta para uso não foram satisfatórias. Para o servidor, o melhor resultado encontrado foi uma implementação de uma classe Java (OLIVETTI; ORACLE, 1998) responsável pela codificação de mensagens do protocolo *RFB* em *byte arrays* que não pode ser aproveitado, pois a solução não era flexível o suficiente para extensão de codificações ou tratamento de *endianess* das mensagens.

Como forma de contornar a falta de uma biblioteca Java com a implementação foi cogitada a utilização de um programa *VNC* interfaceado por um socket Java, que traduziria as mensagens do programa para o WebSocket. Entretanto, a demanda de tempo para entender o funcionamento de um provável software proprietário e implementar uma interface socket dentro do servidor eram desvantagens que pesavam demasiadamente contra a utilização desta abordagem apenas para evitar a implementação da lógica e da codificação de mensagens em formato *RFB*. Ao final uma solução própria da biblioteca *RFB* teve de ser implementada.

4.1.4 IMPLEMENTAÇÃO WEBSOCKET NO SERVIDOR

O WebSocket é um padrão de conexão perene entre servidor e aplicativo cliente já implementado por vários navegadores web populares (MOZILLA, 2014), o que retira o ônus da busca por uma biblioteca funcional para o cliente. Mas a situação é contrária no servidor, pois o Java não fornece em sua API uma classe para atender conexões WebSocket. Conhecida essa deficiência de funcionalidade do servidor foi encontrada uma solução para o lado servidor da aplicação no que tange a implementação de uma conexão WebSocket em Java.

Desenvolvido por Rajlich e Rohmer (2014), o projeto Java-WebSocket contém uma biblioteca que possibilita a configuração e execução de um servidor WebSocket e fornece métodos com assinaturas semelhantes aos das funções definidas pelo padrão WebSocket (FETTE; MELNIKOV, 2011) (RAJLICH; ROHMER, 2014), o que facilita o entendimento do funcionamento da biblioteca e o projeto de uma solução de comportamento semelhante em ambos os lados, cliente e servidor. Além disso, a biblioteca também suporta o uso de certificados digitais, o que possibilita o uso da tecnologia sobre as camadas de segurança TLS/SSL, com um exemplo de utilização sucinta e de fácil compreensão no seu conjunto de códigos fonte (RAJLICH; ROHMER, 2014).

4.1.5 BIBLIOTECA DE SERVIDOR WEB EM JAVA

Um dos obstáculos para a realização do projeto foi a concepção de uma maneira pela qual a máquina cliente recebesse o aplicativo para acesso do servidor. Como já estava assumida uma conexão de rede entre as duas máquinas, a forma mais simples encontrada foi a de embarcar na aplicação do servidor um servidor web, que atenda a requisições HTTP com a entrega da página HTML e os scripts como resposta.

Num esforço de evitar a implementação de uma solução do zero, foi encontrada uma solução em Java que atendia perfeitamente as exigências do projeto: o projeto Jetty, que, ao invés de ser um servidor que comporta o programa desenvolvido (como ocorre no desenvolvimento e implantação de *Web Applications* em Java), ele é um servidor que é embarcado na aplicação desenvolvida (ECLIPSE, 2014).

De forma sucinta, uma das páginas de documentação do Jetty (WILKINS et al., 2014) demonstra como escrever o código para utilizar a biblioteca de forma a criar, configurar e inicializar um servidor que responda a requisições HTTP, seja pela geração dinâmica de páginas web ou na entrega de arquivos HTML com a página web já definida. Além disso, o Jetty também suporta uma funcionalidade que é essencial para este projeto: executar um servidor HTTPS, o que implica na utilização de TLS/SSL, camada de segurança que garante a confidencialidade de dados para a aplicação.

Outro ponto forte do Jetty é a de possuir a implementação de um driver WebSocket que atende a definição JSR 356 de interface comum para implementações de bibliotecas WebSocket para aplicações Java. Entretanto, a aplicação deste TCC já estava a utilizar a implementação WebSocket de Rajlich e Rohmer (2014), que já estava acoplada a aplicação num estado funcional e atendia as necessidades do projeto. Como forma de minimizar a adição de bibliotecas de diferentes fontes e utilizar uma implementação que atenda a definição JSR 356 da Oracle (2014a), a migração para a implementação WebSocket do Jetty é um potencial candidato para trabalhos futuros.

4.1.6 COMUNICAÇÃO COM MOUSE, TECLADO E ÁREA DE TRABALHO

Algumas das funcionalidades de vital importância deste projeto estavam na capacidade da aplicação interagir com o Sistema Operacional para a emulação de eventos de periféricos e a captura de imagem de toda a área de trabalho. Devido ao fato do Java trabalhar sob o conceito de código semi-interpretado dentro de uma máquina virtual (ORACLE, 2014c), o acesso direto de funções do sistema operacional sem o uso de programas ou bibliotecas ligadas pelo JNI (,

que geralmente implica na quebra de portabilidade da aplicação, não é possível. Felizmente a API do Java fornece uma classe que contém uma série de métodos relacionados à interface gráfica do Sistema Operacional: a classe Robot.

A classe Robot é em teoria dirigida para a implementação de testes automatizados de interface gráfica (ORACLE, 2014d). Entretanto, suas funcionalidades de gerar eventos de entrada nativos do sistema de mouse e teclado e de fornecer capturas de tela atendiam perfeitamente às necessidades de projeto (ORACLE, 2014d).

4.1.7 EXECUÇÃO *MULTI-THREAD* DE CÓDIGO JAVASCRIPT

JavaScript é uma linguagem definida para ser executada em *single-thread*, ou seja, ela não permite que diferentes *threads* rodem ao mesmo tempo. Isso acaba gerando um gargalo que piora o desempenho de muitas aplicações. Este trabalho apresenta um exemplo dessa situação: o cliente precisa receber uma série de mensagens ao mesmo tempo que decodifica imagens e as imprime no Canvas da aplicação web. Uma solução para esse problema é o uso de um recurso disponível pelo padrão HTML5: o Web Worker, cuja especificação define uma API para rodar scripts em segundo plano na aplicação web. Ele permite executar scripts externos sem bloquear a interface do usuário ou outros scripts (BIDELMAN, 2014).

Como forma de garantir a segurança da aplicação web e do sistema no qual ela é executada, os scripts a serem executados pelos Web Workers são colocados em arquivos separados. Isso isola o Web Worker do resto da aplicação, garantindo que ele apenas tenha acesso a recursos explicitamente permitidos pelo script pai, que lançou o Web Worker, através da comunicação via mensagens, com argumento composto por strings ou objetos JSON (ROUSSET, 2014). Isso evita uma série de problemas, como acesso concorrente a região crítica ou ações maliciosas sobre o DOM.

Embora Web Workers não tenham acesso ao DOM e aos objetos window, document e parent, eles podem carregar scripts externos e gerar Web Workers filhos, habilidade que implica na capacidade de disparar outras *threads*. Não são todos os navegadores que permitem que Web Workers rodem com arquivos em armazenamento local. A política de segurança do navegador web Google Chrome, por exemplo, não permite tal operação (BIDELMAN, 2014). Já no Internet Explorer, essa tecnologia recebeu suporte na décima iteração do navegador web da Microsoft. Outros navegadores, a saber: Firefox (desde a versão 3.6), Safari (desde a versão 4.0) e Opera 11, também suportam Web Workers (ROUSSET, 2014).

Outros exemplos de aplicação dos Web Workers são: transferência e captura de dados para uso posterior, análise de texto em segundo plano (iluminação de sintaxe em códigos e

checagem de grafia), análise de vídeo ou áudio, entrada e saída realizadas em segundo plano, processamento de grandes quantidade de dados e requisições concorrentes a um banco de dados local (BIDELMAN, 2014)(ROUSSET, 2014).

4.1.8 OBTENÇÃO DE UM CERTIFICADO DIGITAL

A obtenção de um certificado válido para utilização do TLS/SSL, no caso do "Secure Site Pro SSL Certificate" provido pela Symantec (2014), custa na ordem dos mil euros. Por esse motivo, optou-se pela utilização de um certificado auto-assinado, sem passar pela cadeia de certificação. Nesse caso, é esperado que o navegador exiba uma mensagem alertando que o certificado não pôde ser verificado. Nesse quesito, as pesquisas revelaram uma particularidade no Firefox. Quando uma conexão via WebSocket segura é criada e o servidor estiver usando um certificado auto-assinado, não há como forçar o navegador a aceitar uma exceção de segurança, levando o cliente direto a uma mensagem de erro (MARTIN; AL., 2014). Isso pode ser contornado no caso da aplicação deste TCC através da negociação do mesmo certificado, antes da conexão WebSocket, através de uma conexão HTTPS, no qual o navegador pedirá a permissão do usuário para adicionar o certificado às exceções de segurança.

Como forma de gerar certificados auto-assinados, o OpenSSL se apresenta como uma boa solução, aberta e gratuita para uso, apresentando, inclusive, um pequeno passo a passo bem explicado para a geração de certificados auto assinados (LEVITTE, 2014). Entretanto, o OpenSSL não gera certificados no formato JKS, uma crucial necessidade do projeto. Por esta razão a geração de certificados ficou a cargo da ferramenta Keytool (DARMSTADT, 2014).

4.2 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Um dos pontos mais importantes deste capítulo se concentrou no desenvolvimento incremental devido a algumas das tecnologias escolhidas serem relativamente recentes e à falta de familiaridade dos integrantes do grupo com alguns dos recursos e conceitos necessários para projeto de implementação da solução. Outro ponto foi o uso do procedimento de desenvolvimento ADIT na execução, uma tentativa de complementar as ferramentas no entendimento do problema e no projeto de uma solução através da abordagem de desenvolvimento baseado em Casos de Uso.

Outro aspecto importante foi a escolha de algumas tecnologias em fases prévias ao planejamento da implementação, forçada devido à limitação oferecida pelo sistema, presente desde a definição dos objetivos deste trabalho, como o uso de navegadores web.

Com relação às linguagens escolhidas, o desenvolvimento foi baseado no uso de HTML5 para a estruturação das páginas web e JavaScript para as funcionalidades adicionais no cliente da aplicação. Para a comunicação entre cliente e servidor, a tecnologia WebSocket foi a escolhida conforme as vantagens mostradas no capítulo 2 entre as possíveis comunicações cliente-servidor a partir de navegadores web. Na parte de segurança, o uso de TLS/SSL foi preferido devido à possibilidade de integrar a comunicação WebSocket e ser suportado por navegadores web. O protocolo de compartilhamento de área de trabalho é baseado no VNC por ser bem documentado, largamente utilizado e fácil de estender funcionalidades. Com o intuito de auxiliar no desenvolvimento, o IDE Eclipse foi adotado devido à sua leveza e ao suporte a HTML5 e JavaScript. Por fim, foram escolhidos navegadores web convencionais com suporte às tecnologias expostas anteriormente, pois dessa forma é possível realizar os testes e exibir as páginas web desenvolvidas para o acesso remoto.

5 DESENVOLVIMENTO DO PROTÓTIPO

Neste capítulo está detalhada a execução do processo de desenvolvimento do trabalho, conforme delineado pela metodologia adotada, descrita no capítulo 4. O início do projeto consistiu no contato inicial com as tecnologias escolhidas, através do desenvolvimento dos *toyprojects* propostos, seguidos pela apresentação dos resultados obtidos. As fases posteriores foram compostas pelo projeto/modelagem da aplicação desenvolvida, comparados os resultados dos processos baseados no levantamento de Casos de Uso, utilizado no início do projeto, e ADIT, processo adotado posteriormente como forma de superar algumas das dificuldades da fase de projeto do software. Como descrito no capítulo 3, dedicado para o processo de desenvolvimento de software, o ADIT inclui as fases de implementação e testes do software.

5.1 *TOYPROJECTS*

Esta parte do capítulo é dedicada para a apresentação da fase inicial de desenvolvimento do TCC, no qual, a partir das tecnologias escolhidas, algumas provas de conceito foram criadas para demonstrar as suas funcionalidades, adquirir conhecimento de cunho prático e verificar o atendimento das necessidades do projeto. Como não são componentes completos e não exigem um processo aprofundado de projeto de software, essas provas de conceito são nomeadas *toyprojects*. Como forma de melhor aproveitar o recurso de tempo para desenvolvimento do TCC, o desenvolvimento de *toyprojects* foi limitado a três pontos essenciais do trabalho: a primeira envolveu a verificação da capacidade de WebSockets proverem uma conexão bidirecional, capaz de trocar dados em forma textual ou de octetos binários, entre o servidor em Java e o cliente em JavaScript. O segundo *toyproject* foi a respeito do HTML5 e o seu componente mais importante para o projeto, o Canvas, o qual permite a manipulação gráfica de uma área da página web através de um programa JavaScript. O terceiro *toyproject* envolveu a integração dos dois primeiros, com a adição da utilização de um certificado digital para verificar a capacidade do WebSocket trabalhar sobre TLS/SSL, que garante para a aplicação o transporte de dados pela rede de forma segura, cifrando os dados de forma que apenas o destinatário dos pacotes possa decifrá-los e lê-los.

5.1.1 TOYPROJECT 1 – WEBSOCKETS

Para fins de atender a necessidade do primeiro *toyproject*, um programa simples, mas muito utilizado como verificação primitiva de sistemas de comunicações, foi desenvolvido, conhecido como um servidor de eco, que apenas reenvia para os clientes conectados os dados que são recebidos.

WebSockets disponibilizam duas formas de passagem dos dados, uma é textual, outra é binária (FETTE; MELNIKOV, 2011), ambas as formas foram testadas na prova de conceito produzida. O servidor é simples e apenas disponibiliza uma conexão numa porta predefinida. Ao receber uma mensagem o servidor apenas a reenvia pelo mesmo WebSocket. O cliente, por sua vez, é composto por uma simples interface gráfica: uma caixa de texto, um botão de envio e o restante da página dedicada para listar as mensagens recebidas do servidor. Ao ser pressionado o botão, o conteúdo da caixa de texto é enviado para o servidor. Ao receber uma mensagem, um parágrafo contendo o texto da mensagem é adicionado à página.

Ao final do desenvolvimento deste *toyproject* o resultado demonstrou que as tecnologias escolhidas para utilização do WebSocket na aplicação funcionaram com sucesso. Adicionalmente, alguns conhecimentos interessantes foram adquiridos ao longo desta parte:

- Nenhuma configuração adicional é necessária para a transmissão textual. A transmissão binária de dados pode resultar em duas representações distintas: uma em Blob, notação útil para um fluxo de dados, como arquivos; outra em ArrayBuffer, direcionada para o transporte de mensagens em formato binário, situação que descreve a necessidade deste trabalho.

- Para escolher entre uma das duas representações o parâmetro binaryType da conexão WebSocket no JavaScript deve ser atribuído com as strings “blob” ou “arraybuffer”.

- A forma mais conveniente de identificar se a mensagem é textual ou binária é verificar se os dados da mensagem estão representadas por uma string (forma textual) ou por um ArrayBuffer (forma binária).

- Objetos ArrayBuffer não podem ser manipulados diretamente. Para isso o JavaScript fornece objetos DataView (ou FileReader no caso de Blob).

- A biblioteca de WebSocket utilizada no servidor já possui métodos separados para se trabalhar com uma string ou ByteBuffer ou um vetor de bytes.

5.1.2 TOYPROJECT 2 – CANVAS

Depois de obtido um *toyproject* de conexão WebSocket bem sucedido, uma segunda prova de conceito foi produzida, na qual o servidor, a cada evento de recepção de mensagem, envia uma imagem da sua Área de Trabalho pelo WebSocket.

Desta vez a página web do cliente contém um grande Canvas, no qual é desenhada a imagem recebida pelo WebSocket. Durante o processo, os maiores problemas percebidos foram:

- A frequência de atualizações do Canvas: a atualização lógica do elemento demora a refletir na GUI. Como a concepção de estratégias de compressão de imagem e codificação de *pixel data* estavam fora do escopo do projeto, a solução encontrada foi a de utilizar uma resolução limitada na área de trabalho do servidor, diminuindo assim o tamanho das imagens a serem transportadas pela rede.
- Formato de codificação da imagem: ao enviar arquivos de imagem, não há uma maneira direta de mostrá-la no elemento Canvas da página web, a não ser que ela esteja codificada em base64, o que aumenta a quantidade de bytes necessária para representar o arquivo de imagem. Isso é um problema que foi verificado em um dos trabalhos relacionados, levantado no capítulo 2.

5.1.3 TOYPROJECT 3 – WSS

A produção deste último *toyproject* foi uma simples alteração do código dos anteriores, no qual a implementação do gerenciador de conexões WebSocket no servidor Java foi alterada de *default* para um SSLContext. Com o fornecimento de um certificado, chaves e senhas válidas, a produção de uma conexão WSS (WebSocket Secure) passa a ser possível, e a comunicação passa a ser suportado pelo protocolo TLS/SSL, fornecendo a segurança desejada.

O maior problema encontrado durante a implementação residiu não na obtenção do certificado, mas sim na forma como os certificados e as chaves são armazenadas. Como, por exemplo, dentro do próprio Java o formato de escolha é o JKS (*Java KeyStore*), entretanto há um sistema, desenvolvido para Java implementado para fornecer um servidor de Web Applications chamado Apache Tomcat, que por padrão aceita certificados armazenados em formato PEM (*Privacy Enhanced Mail*). Existem, além desses, muitos outros formatos, e essa indefinição pode gerar uma dificuldade para o usuário do sistema, que passa a ter de se preocupar com detalhes técnicos acerca de certificados digitais. Como este não era um problema previsto no

projeto do sistema e a sua aceitação também não geraria impacto negativo no comportamento do sistema, a tradução dos possíveis formatos de armazenamento de certificados digitais foi considerada fora de abordagem neste TCC. Entretanto, é um problema interessante para ser atacado com a finalidade de tornar o software um produto comercial.

5.2 A ABORDAGEM DO PROJETO DE SOFTWARE POR CASOS DE USO

Durante a formulação do plano de projeto o levantamento de requisitos foi feito através da abordagem por Casos de Uso. Esta parte do capítulo é dedicada para mostrar o resultado obtido pela utilização deste procedimento durante a fase inicial do projeto e as dificuldades enfrentadas, que culminaram na introdução do processo de desenvolvimento ADIT. Nesta seção são apresentados apenas alguns resultados do processo por Casos de Uso, mais detalhes estão no apêndice A.

Devido a falta de procedimentos bem definidos, os requisitos levantados durante essa fase assumiram uma visão muito distante das necessidades técnicas do desenvolvimento de software. Isso totalizou, num projeto com tantos objetivos específicos, um levantamento de meros 5 requisitos funcionais e 4 requisitos não-funcionais. Neste nível havia uma grande confusão entre as características do ambiente e as características desejadas do sistema a ser implementado, o que acaba por invalidar os já poucos requisitos levantados. Visivelmente o projeto estava incompleto.

De forma geral, pode se dizer que a representação por casos de uso é de extrema valia em casos quando os atores são entidades reais, e normalmente identificados como *Biddable Domains* pelo ADIT, e quando a notação de orientação a objetos mapeia exatamente a estrutura pertinente ao ambiente que ela automatiza/representa. Entretanto, no caso de uma aplicação que se comunica com outros componentes de hardware e de software e possui comportamento não algorítmico, a abstração de casos de uso apenas obscurece necessidades que devem ser abordadas durante o desenvolvimento do programa e atingir os objetivos do projeto.

A necessidade de um processo bem definido para se alcançar os requisitos necessários para o alcance dos objetivos do trabalho levaram a redefinição do projeto de software através do ADIT.

5.3 A ABORDAGEM DO PROJETO DE SOFTWARE POR ADIT

Durante a execução do projeto, sentiu-se a necessidade de um procedimento bem definido para aproveitar todo o potencial do UML como linguagem de modelagem e representação

de sistemas. A partir desta necessidade, o ADIT se destacou como um complemento interessante para o projeto: ele apresenta um procedimento iterativo e claro de modelagem para o desenvolvimento de software.

No capítulo 3, foi evidenciado que o ADIT fornece um processo de desenvolvimento e gera modelagens em UML. Também foi dito que os projetos de desenvolvimento de software possuem particularidades, mas seguem fases semelhantes durante a execução. Portanto, ao longo da explanação da abordagem do problema via ADIT, é feita uma equivalência entre o processo e a representação do sistema por Casos de Uso, demonstrando que o ADIT pode ser utilizado de forma consistente para a geração de um levantamento do sistema com resultados semelhantes a abordagem por Casos de Uso.

Ao longo desta seção estão os problemas encontrados, as soluções concebidas e os resultados obtidos em cada uma das fases do ADIT, conforme definidas no capítulo 3, bem como a equivalência do processo com os conceitos de Casos de Uso.

5.3.1 FASE A1 – LEVANTAMENTO E DESCRIÇÃO DE PROBLEMAS

A fase A1 começa com a definição de uma descrição informal do sistema a ser construído. Esta descrição é facilmente obtida através dos objetivos estabelecidos para o projeto (no caso deste TCC eles estão descritos no capítulo 1). A descrição resultante é compreendida no seguinte parágrafo:

O SRDWeB deve permitir o usuário visualizar e comandar a área de trabalho de seu computador remotamente, de forma segura e através de uma aplicação web. O protocolo do sistema será baseado no RFB (VNC) estendido. Uma parte do sistema consiste num programa servidor instalado na área de trabalho alvo, o qual disponibilizará uma conexão pela rede. O servidor em execução fica a espera de uma conexão, a partir do qual inicializará a execução do protocolo RFB com o cliente. Ao abrir a conexão: o servidor disponibiliza um simples servidor HTTP, com o qual o cliente poderá obter a aplicação web para controle da máquina remota; a partir da conexão HTTP o cliente não deve mais precisar entrar com o endereço de destino. O cliente entra com o endereço do servidor e recebe a aplicação web no seu navegador. A partir deste momento este estará conectado, enviando eventos de mouse e teclado e recebendo as imagens da máquina remota. A comunicação deve ser segura.

É importante ressaltar que todos os procedimentos da fase partem da descrição informal. Portanto, a definição clara dos objetivos do trabalho é essencial para o sucesso do projeto. Depois de definido o objetivo, restam as definições das declarações a respeito do ambiente e a concepção do diagrama de contexto, que coloca a máquina no ambiente que se deseja modificar.

A partir da descrição informal são separadas as características que pertencem ao ambiente, identificado no ADIT por fatos, e quais as características que descrevem o sistema quando os objetivos do projeto são atendidos, identificadas pelos requisitos. Eventualmente, algumas características variáveis ou imprecisas do ambiente são encontradas, e nem todas elas podem e nem devem ser tratadas na implementação da máquina. É neste ponto que entra a limitação de escopo do projeto, definido por características que o ambiente deva ter para que os requisitos sejam alcançáveis, identificadas pelas suposições. Ao final do trabalho, o sistema foi definido por 15 suposições, 20 fatos e 17 requisitos. Todos eles estão listados no apêndice B.

Ao ter uma descrição formal, topificada e clara do sistema, resta para o final da fase o diagrama de contexto, representado na Figura 5. Neste ponto começa a identificação das partes ativas do sistema e com quais partes do ambiente a máquina interage. Os elementos, novamente, são levantados a partir da descrição informal, complementados pelas características levantadas. O importante aqui é identificar os domínios e suas relações, representadas através de fenômenos. Isso fornece uma ideia clara da composição do sistema.

Os resultados desta fase do ADIT correspondem diretamente aos requisitos da abordagem por Casos de Uso. Esse procedimento facilita a visualização do sistema e pode ser utilizado para se ter um conjunto claro de requisitos para o desenvolvimento dos diagramas UML partindo da análise de Casos de Uso.

Durante a execução não houve grandes problemas com relação à fase A1. Entretanto, a lista de declarações e o diagrama de contexto passaram por constantes reformulações à medida que outras fases eram concluídas, ou seja, conforme era adquirido mais conhecimento a respeito do sistema ao longo das iterações, refletindo exatamente o que foi dito sobre o processo no capítulo 3.

Apesar dos requisitos funcionais serem importantes para guiarem o desenvolvimento de uma solução que atinja os objetivos do projeto, foram os requisitos não-funcionais que de fato impactaram a progressão do trabalho e geraram revisões mais profundas no projeto.

5.3.2 FASE A2 – DECOMPOSIÇÃO DE PROBLEMAS

Na fase A2, os requisitos foram ligados ao diagrama de contexto, através da concepção dos diagramas de problema. O intuito desta fase é a de compor o problema mais complexo do sistema em pequenos problemas simples, mais fáceis de serem abordados de maneira independente, o que vem a facilitar a paralelização da implementação do sistema em fases futuras.

Os diagramas de problemas mostram quais partes do ambiente os requisitos restringem

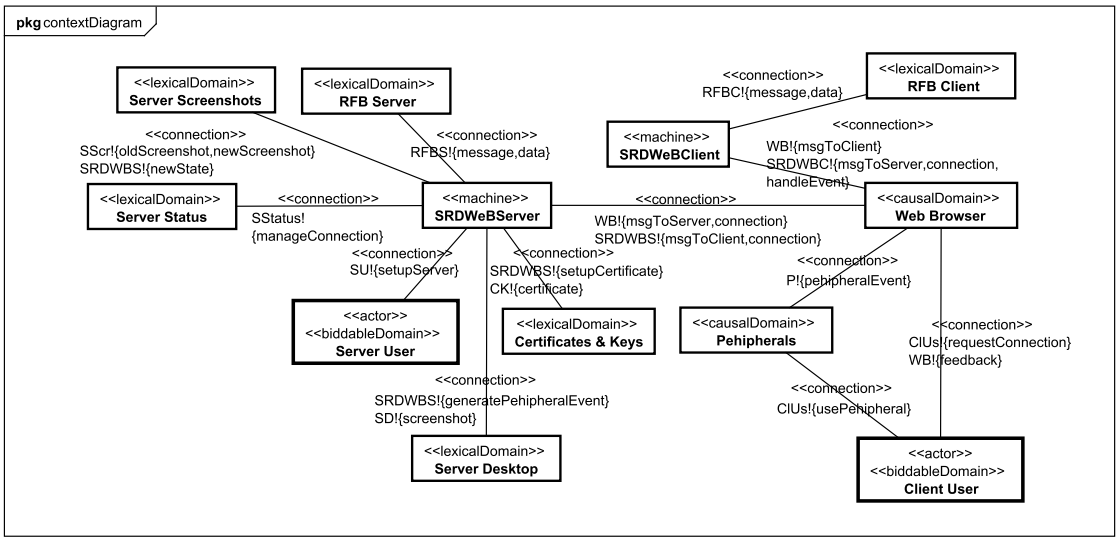

Figura 5: Diagrama de Contexto do projeto, levantado na fase A1.

ou fazem referência, o que indica para fases futuras sobre quais partes do ambiente a máquina deve trabalhar para alcançar os objetivos. Depois de levantados os diagramas, os problemas podem ser divididos em partes mais específicas ao serem encaixados em *problem patterns*. Isso pode parecer pouco interessante à primeira vista, mas, ao respeitar esse passo, dividem-se os requisitos e os domínios em classes, sendo cada classe um subproblema do projeto, e cada subproblema do ADIT pode ser traduzido a um caso de uso no diagrama de Casos de Uso do UML. Os diagramas resultantes da fase A2 estão no apêndice C.

Durante esta fase percebeu-se a incompletude do diagrama de contexto da fase A1, que antes estava visualizado sob apenas uma única máquina, identificada por “SRDWeB”. Entretanto, encarar o sistema como uma única peça começou a dificultar a abordagem, e o problema ainda estava complexo demais para ser instanciado dentro de *problem patterns*. A fase A2 mostrou ser mais sensato encarar o projeto como o desenvolvimento de dois sistemas independentes, o que resultou na quebra de “SRDWeB” em duas “machines” distintas: “SRDWB_Server” e “SRDWeB_Client”, o que implicou na divisão do *Biddable Domain* “User” em “Server User” e “Client User”.

5.3.3 FASE A3 – DERIVAÇÃO DE ESPECIFICAÇÕES

A fase A3 se dedica a traduzir as características desejadas do sistema nas características da máquina, o que resulta nas declarações chamadas de especificações, que, como apresentado no capítulo 3, são requisitos implementáveis. De forma simples, as declarações até aqui levantadas podem ser relacionadas por uma simples equação, como mostrada na Fórmula 5.1.

$$Requirements = Facts \cup Assumptions \cup Specifications \tag{5.1}$$

Dessa forma, garante-se que o cumprimento das especificações, dentro do ambiente definido pelo escopo do trabalho, implica no atendimento dos requisitos. Essas equivalências estão descritas no apêndice D.

O detalhamento das especificações fica por conta da sequencialização da ocorrência dos fenômenos ocorridos entre os domínios, mostrado pelos diagramas abstratos de especificação, também inclusas no apêndice D. As especificações do ADIT são equivalências diretas dos fluxos básico e alternativo dos casos de uso no UML.

A fase A3 mostrou a impossibilidade de alguns requisitos serem encaixados em especificações, devido ao fato de estarem descritos incorretamente, ou não serem de fato requisitos. Portanto, a fase A3 eliminou as incertezas acerca das declarações levantadas na fase A1 e as relacionou diretamente com a implementação da máquina.

5.3.4 FASE A4 – ESPECIFICAÇÃO TÉCNICA DE SOFTWARE

A fase A4 introduz domínios de conexão ao diagrama de contexto. Isso é feito para indicar entre quais domínios os recursos e tecnologias se encaixam no ambiente, em sua maioria levantados no capítulo 4. A figura 6 mostra o diagrama resultante da fase, e os detalhes estão apresentados no apêndice E.

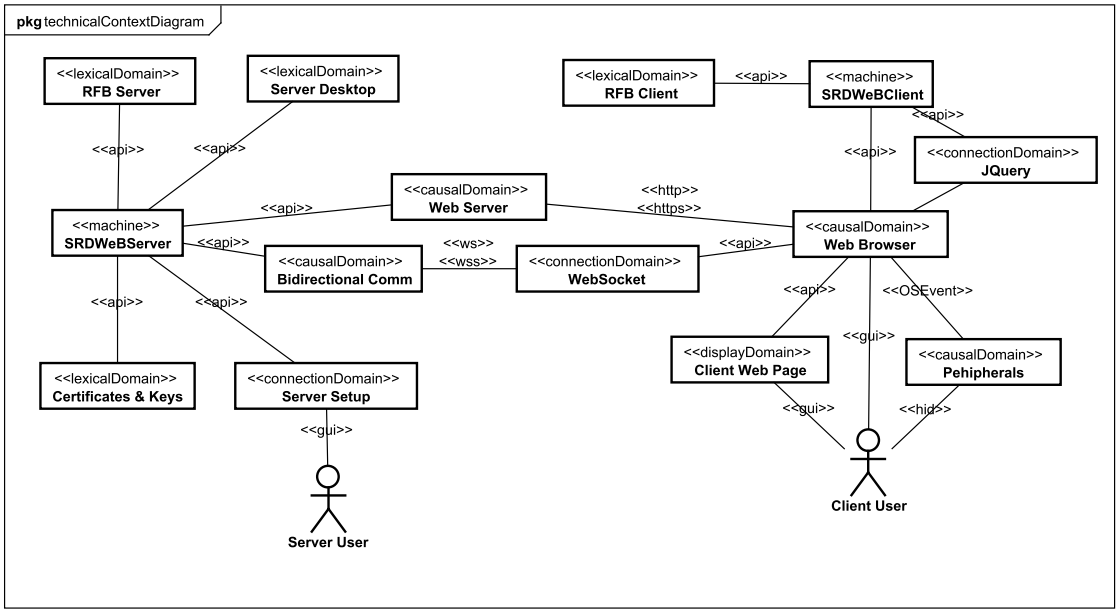

Figura 6: Diagrama de Contexto Técnico, levantado na fase A4.

O relacionamento do contexto com detalhes técnicos revelou a necessidade de rever alguns fenômenos levantados até esta fase. O comportamento das tecnologias escolhidas para o projeto implicou na mudança de alguns fenômenos, como ocorreu no caso dos WebSockets do navegador, a partir dos quais surgiu a necessidade de representar a instância de uma conexão para o SRDWB_Client devido a forma padronizada para recepção e envio das mensagens, traduzidos para os fenômenos “onMsgToClient”, “sendMsgToServer”, “onOpenToClient” e “onCloseToClient”. De forma semelhante “onMsgToServer”, “sendMsgToClient”, “onOpenToServer” e “onCloseToServer” foram introduzidos, juntos ao “Bidirectional Comm” para o sistema “SRDWB_Server”. Além disso, nesta fase também foi percebida a necessidade de um servidor web para o fornecimento das páginas web do servidor para o cliente. Devido à forma como o Jetty funciona, o domínio de Web Pages & Scripts perdeu sentido, pois o Jetty é capaz de buscar por conta própria os arquivos a serem enviados para o cliente, tirando o ônus da aplicação.

5.3.5 FASE A5 – ESPECIFICAÇÃO DE ATRIBUTOS E OPERAÇÕES

Esta fase é a que definiu as mensagens trocadas pela máquina através do *Design by Contract*. Basicamente são definidas as pré e pós-condições de cada uma das mensagens delineadas até o momento. O que possibilita, através do encadeamento das mensagens, validar os predicados de estado e definir o ciclo de vida do software. A fase é apresentada com mais detalhes no apêndice F.

Comparado a exemplos de programas, em que o comportamento das funções depende de um range de valores, as trocas de mensagem que ocorrem no caso deste projeto são, em geral, booleanos, ou então baseados em máquinas de estado. Isso faz com que as restrições de cada método sejam simples. Nesta fase houve alterações no que tange os predicados de estado levantados nas especificações, principalmente os relacionados ao Server User, os quais impossibilitavam a livre transição das fases de configuração do servidor. Os predicados de estado foram alterados de forma a deixarem as transições entre Port Setup, Mode Setup e Finish Setup mais flexíveis (possibilitando prosseguir e retornar entre sequências), e também facilitar a introdução de fases intermediárias, caso fossem necessárias na configuração do servidor.

5.3.6 FASE A6 – CICLO DE VIDA DO SOFTWARE

Ao final da análise, é levantado o ciclo de vida do software, útil para a definição das GUIs e ter uma visão geral do comportamento do sistema como uma unidade formada pelos diversos subproblemas levantados.

Ao encadear as diversas especificações numa sequência se percebeu a incerteza cau-

sada por alguns predicados de estado mal formulados. Por ser um sistema fortemente baseado em protocolos, a execução da aplicação é, até certo ponto, bem linear, não permitindo muitos desvios na maior parte das especificações, exceto na fase que implementa o RFBInteraction do servidor e do cliente.

5.3.7 FASES DE PROJETO

Ao término da fase de análise pode se dizer que o problema do projeto foi dividido em partes simples, a serem abordadas individualmente. A partir dos resultados obtidos, como a estrutura dos domínios e as mensagens trocadas entre eles, desenvolveu-se o projeto da solução.

A primeira fase de projeto, D1, consistiu no detalhamento da arquitetura do sistema, que foi baseada numa arquitetura em camadas e traduzida direta da especificação de operações da fase A5. Do ponto de vista qualitativo não houveram alterações na estrutura da aplicação até então levantada, o que resulta numa arquitetura, mostrada nas Figuras 7, para o cliente e 8, para o servidor. A fase está apresentada em detalhes no apêndice G.

As fases D2 e D3 seguiram as especificações levantadas em A3, baseadas na estrutura de software definida na fase D1. Elas ordenam as trocas de mensagens entre os componentes definidos na arquitetura de D1 e estão apresentadas no Apêndice H.

Para fins de implementação, uma equivalência da fase de projeto foi desenhada em forma de diagrama de classes, mostradas nas Figuras 9 e 10. A partir do diagrama em 10 foi feita uma tradução direta para código Java, a partir do qual os métodos foram preenchidos com base nas trocas de mensagens previstas em D2 e D3.

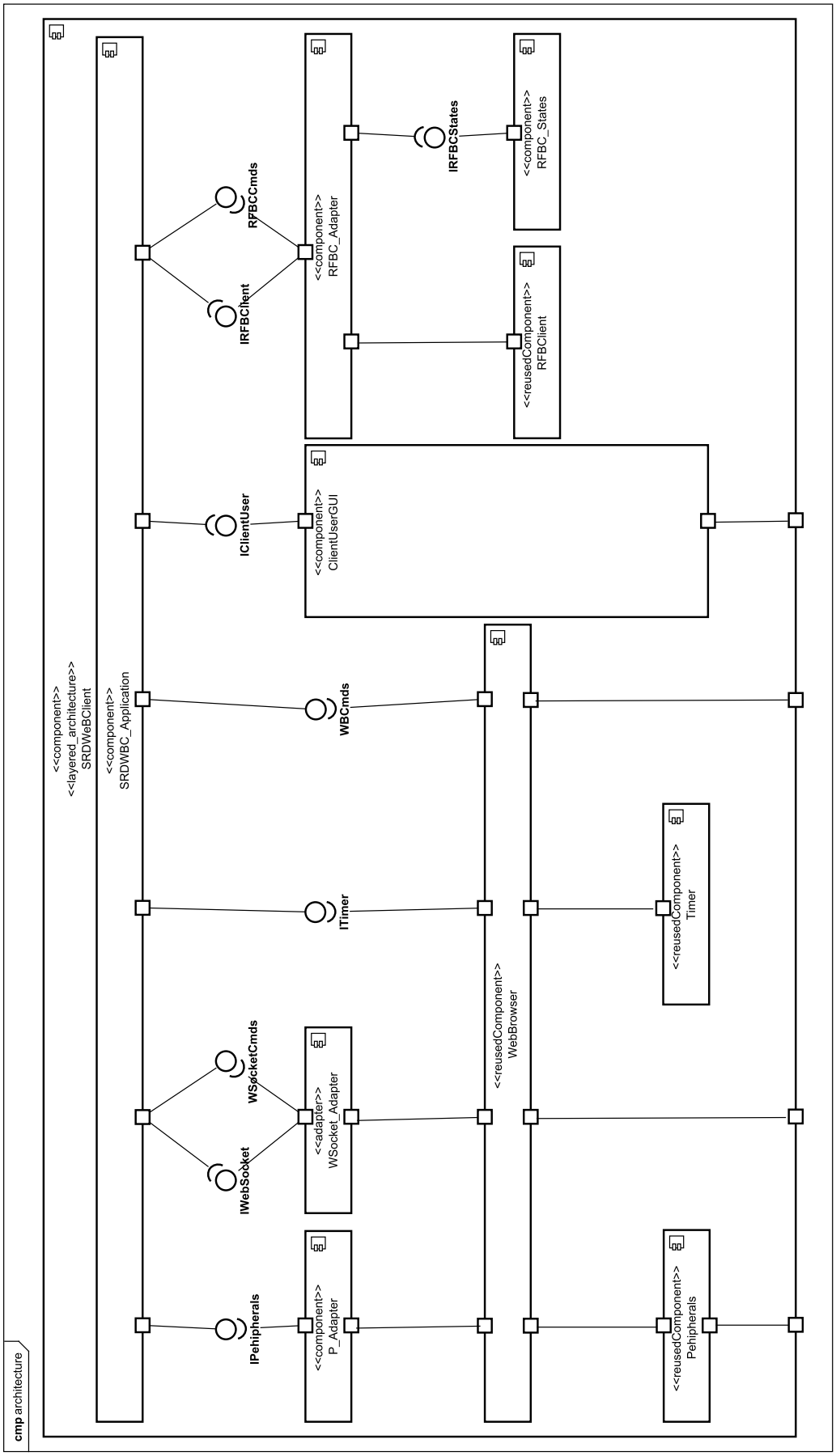

Figura 7: Diagrama de Arquitetura para o SRDWeBClient, levantado na fase D1.

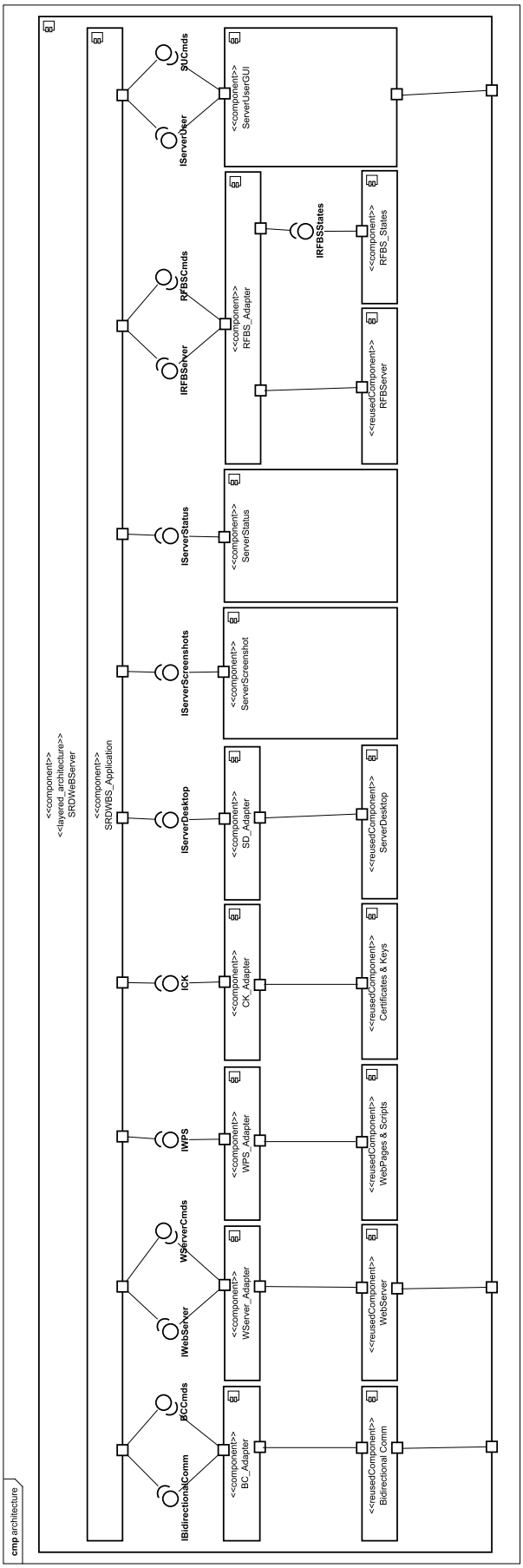

Figura 8: Diagrama de Arquitetura para o SRDWeBServer, levantado na fase D1.

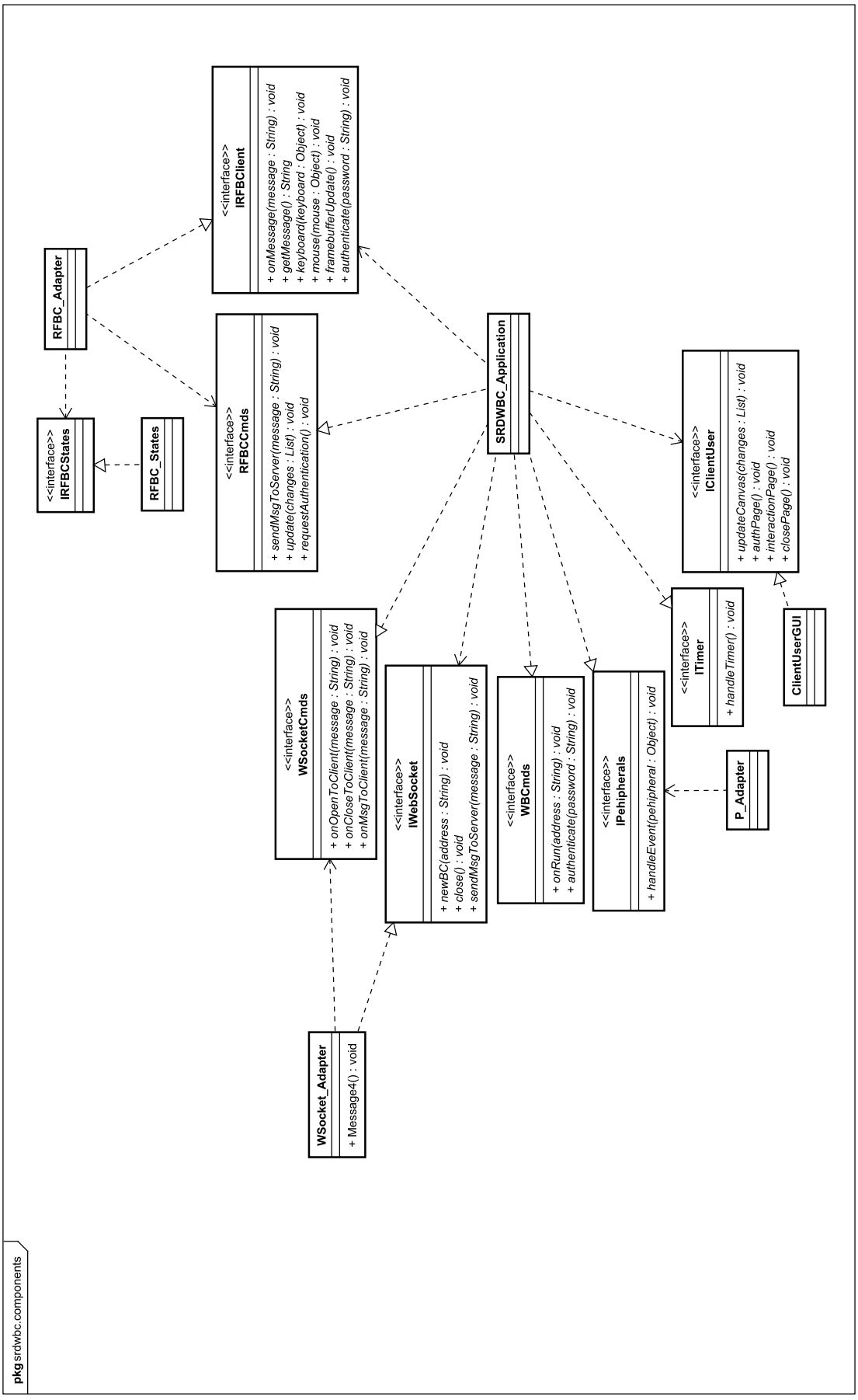

Figura 9: Diagrama de Classes para SRDWeBClient.

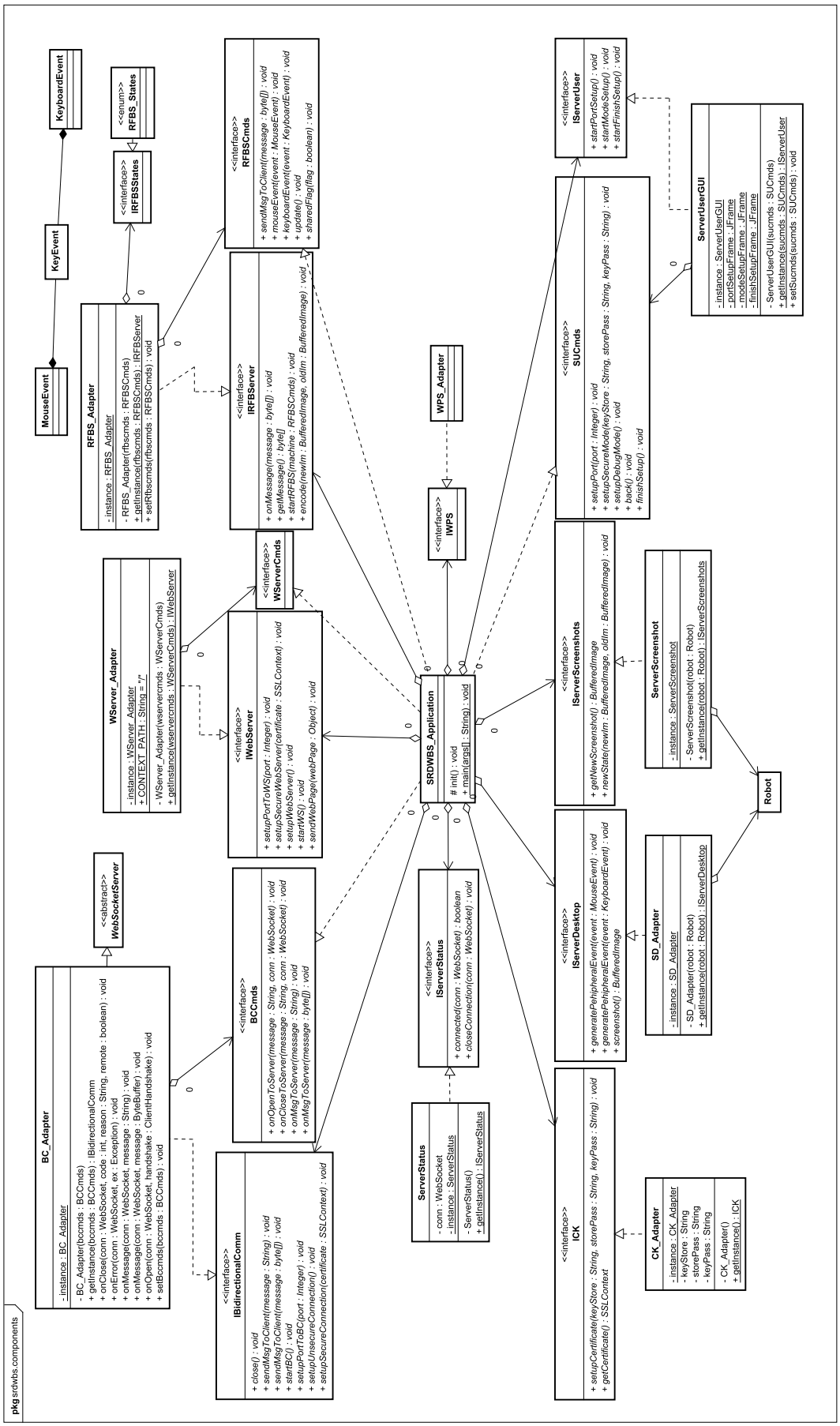

Figura 10: Diagrama de Classes para SRDWeBServer.

5.3.8 FASES DE IMPLEMENTAÇÃO E TESTES

Devido ao processo cuidadoso pelo qual o projeto da máquina passou, a implementação dos protótipos não sofreu atrasos, e passaram pelos testes dos componentes sem alterações significativas. Ou seja, do ponto de vista funcional a aplicação foi obtida sem problemas. Entretanto, a natureza *bottom-up* da fase de implementação sofreu com problemas de natureza não-funcional, características que poderiam ser avaliadas apenas quando todos os componentes estivessem encaixados e os últimos testes de aceitação fossem executados. Detalhes destas fases estão no apêndice I.

Vários problemas foram encontrados durante as fases de implementação e testes:

- A codificação Raw (com a imagem em bitmap) funcionou. Entretanto, o desempenho deixou a desejar, o que implicou na utilização de uma forma semelhante à codificação por arquivo de imagem utilizada no *toyproject* 1.
- Mesmo com a alteração de codificação, a aplicação ainda apresentava lentidão. Isso implicou na utilização de métodos vistos na seção 2.2.6: *framebuffer* dividido em vários pequenos quadros, dentre os quais apenas os quadros com alteração são enviados.
- Quadros retangulares que tenham largura menor do que a largura do *framebuffer* aparentemente apresentam lentidão na atualização do Canvas.
- As imagens eram enviadas sem muito atraso, mas mesmo assim o cliente apresentou atualizações muito lentas, devido à provável latência da interpretação do código JavaScript numa única *thread*, o que implicou na utilização inevitável de Web Workers.
- Atualização entre o buffer e o Cavas é feita de forma *interlaced* com a função requestFrameAnimation.
- O jQuery facilitou a implementação sobre diferentes navegadores e simplificou o código, mas apresenta alguns problemas, como na busca e manipulação de elementos exclusivos do HTML5 e a detecção da posição do mouse, que é feita com referência à área de exibição da página (o *viewport*), e não da página em si. Eventos de *mouse scrolling* não são suportados, e o suporte à algumas funcionalidades no Internet Explorer não é garantida.

Importante ressaltar que estas mudanças não impactaram de forma alguma a estruturação original do software. O projeto foi feito de tal forma que o planejamento não fosse impactado gravemente por pequenas mudanças de natureza não-funcional, principalmente graças

a aplicação de *Design Patterns* no projeto. Essa característica foi de extrema importância para o desenvolvimento, a ponto do programa poder apresentar resultados visíveis mesmo enquanto vários dos componentes ainda não estavam implementados, como os módulos de segurança do TLS/SSL, a autenticação do cliente descrita pelo protocolo RFB ou os processos de Encoding para acelerarem a apresentação de quadros por segundo no cliente.

Outro ponto importante do desenvolvimento é a natureza multi-thread da aplicação, o que dificulta a busca por erros de aspecto funcional através de depuração da execução do programa. Por esta razão o programa deveria ser feito corretamente desde o princípio. Esse problema foi evitado devido a definição do processo de projeto e desenvolvimento, garantida pela escolha do ADIT aliado ao UML, nas fases iniciais do TCC.

A única alteração de natureza funcional que ocorreu durante a implementação foi a introdução de uma nova funcionalidade na máquina: paralelizar o armazenamento de mudanças no *framebuffer* ao invés de executar apenas sob requisição de atualização do cliente. É uma sugestão proposta em um dos trabalhos relacionados levantados anteriormente e que pode ser implementada sem profundas alterações ao adicionar os fenômenos de obtenção e armazenamento de uma imagem da Área de Trabalho independentes do FramebufferUpdateRequest, e propagar essa alteração até a fase de implementação.

Uma particularidade que merece destaque é a forma como a conexão entre cliente e servidor é estabelecida. A Figura 11 mostra detalhes da sequência de passos necessários. Primeiramente é realizada uma conexão HTTPS entre cliente e Servidor Web (na porta 8443) para a obtenção dos arquivos HTML e JavaScript. O segundo passo é um conexão HTTPS entre cliente e Servidor WebSocket (na porta 8080) para que o navegador forneça a opção de adicionar uma exceção de segurança ao certificado: isso ocorre porque o certificado utilizado é auto-assinado e por padrão o navegador rejeita esse tipo de conexão. O último passo é a conexão via WebSocket seguro ao servidor WebSocket, dando então início ao protocolo RFB.

5.4 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Apesar das ferramentas desenhadas pelo padrão UML serem de grande valia para o processo de desenvolvimento de software, elas não definem um processo a ser seguido para que a modelagem do sistema seja concebida. Essa lacuna foi preenchida pela escolha do ADIT. O ADIT define precisamente o que são requisitos, especificações, domínios, por exemplo, e isso facilita a separação das ideias e definições do sistema.

A natural ligação das fases do ADIT faz com que um processo de desenvolvimento em

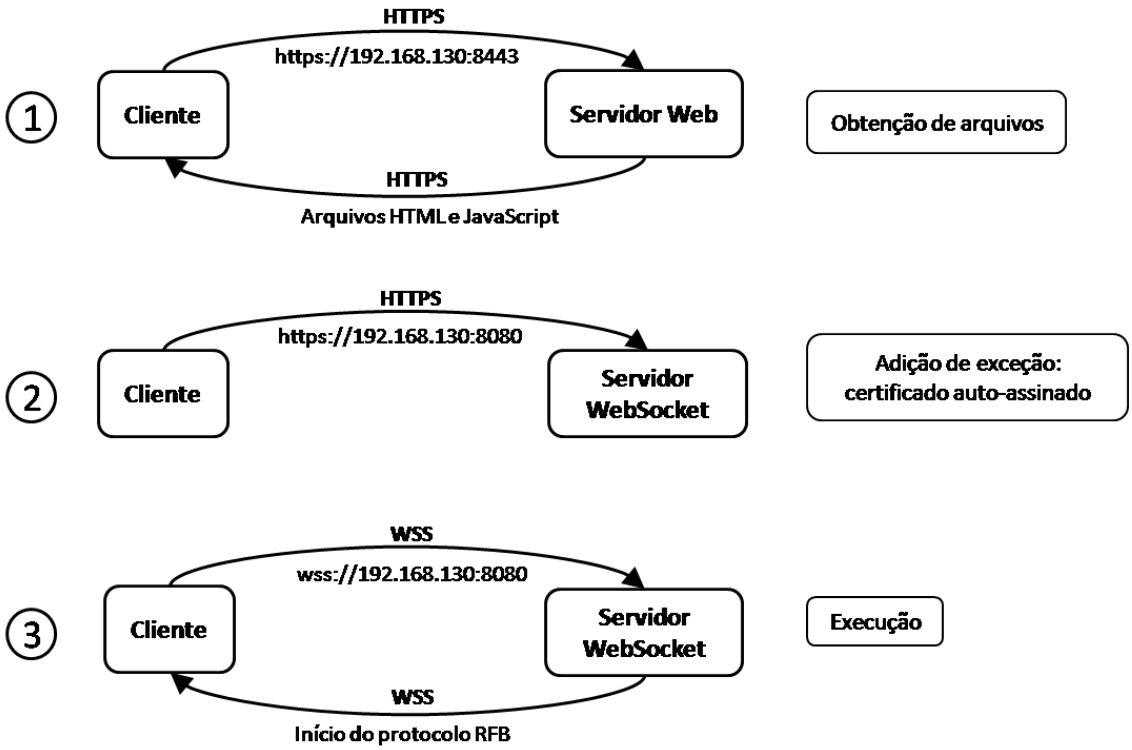

Figura 11: Diagrama ilustrando as fases/etapas da conexão.

V valide o resultado do projeto com os objetivos propostos.

A fase de testes não implicou em mudanças na aplicação até alcançar os testes de aceitação, no qual os aspectos não funcionais da aplicação começaram a ser avaliados. Os aspectos que mais interferiram foram o envio excessivo de eventos de mouse e teclado, por vezes sob codificação incorreta, devido a especificidades de plataforma, e a atualização excessivamente lenta do Canvas no cliente. Além do suporte a múltiplas conexões WebSocket ao invés de uma única no servidor, o que dificultava o tratamento de conexão.

A utilização de *Design Patterns* garantiu que o código seja reutilizável, e facilitou a revisão e correções definidas pela fase de testes. Pois as mudanças não foram estruturalmente profundas, apenas uma troca de componentes.

A revisão bibliográfica se mostrou importante, pois muitas das alterações ocorreram sob luz de sugestões presentes nos artigos, teses e documentos levantados durante a fundamentação teórica.

6 ANÁLISE POST-MORTEM DO PROJETO

Independente do sucesso ou falha de um projeto, muito pode ser aprendido com uma simples análise após a sua conclusão. Avaliar se os resultados obtidos e recursos consumidos condizem às projeções levantadas ao início do trabalho, ou buscar a razão da ocorrência de falhas durante o processo de desenvolvimento, podem levar a conhecimentos valiosos para a execução de projetos futuros. Para isso, este capítulo foca nos aspectos de maior importância para este TCC: comparação do cronograma proposto com o executado, discussão acerca das escolhas metodológicas do trabalho, análise da viabilidade de levar o resultado a se tornar um produto comercial e a proposta de funcionalidades não alcançadas neste trabalho, mas que poderiam beneficiar a qualidade da aplicação.

6.1 ANÁLISE DO CRONOGRAMA EXECUTADO

Durante a fase de planejamento foi criado um cronograma com as atividades básicas a serem desempenhadas. Na Figura 12 é apresentado um gráfico com o nome das principais atividades e seu desenvolvimento desde que foi dado início ao planejamento. Os quadros marcados em azul representam as atividades planejadas em cada mês e os quadros hachurados representam a realização efetiva da tarefa em questão.

Nota-se que a atividade relativa ao desenvolvimento do *Toy Project* Editor Gráfico Colaborativo não foi realizada por escolha da equipe. Os outros projetos foram suficientes para testar as tecnologias que a equipe não conhecia ou nas quais tinha pouca experiência.

Terminada a fase de criação e extraídas as conclusões dos toy projects, notou-se que os requisitos careciam de mais refinamento. O método de obtenção dos requisitos por Caso de Uso possui seu mérito, mas faltaram detalhes de componentes de software para guiar o desenvolvimento. Por essa razão foi realizada uma reformulação que levou ao levantamento de novos requisitos pela técnica ADIT. Essa fase é delicada pois nela são tomadas várias decisões importantes para a implementação, por isso decidiu-se atrasar o cronograma (em 2 meses) e focar em um bom planejamento. As demais atividades ocuparam as janelas de tempo pre-

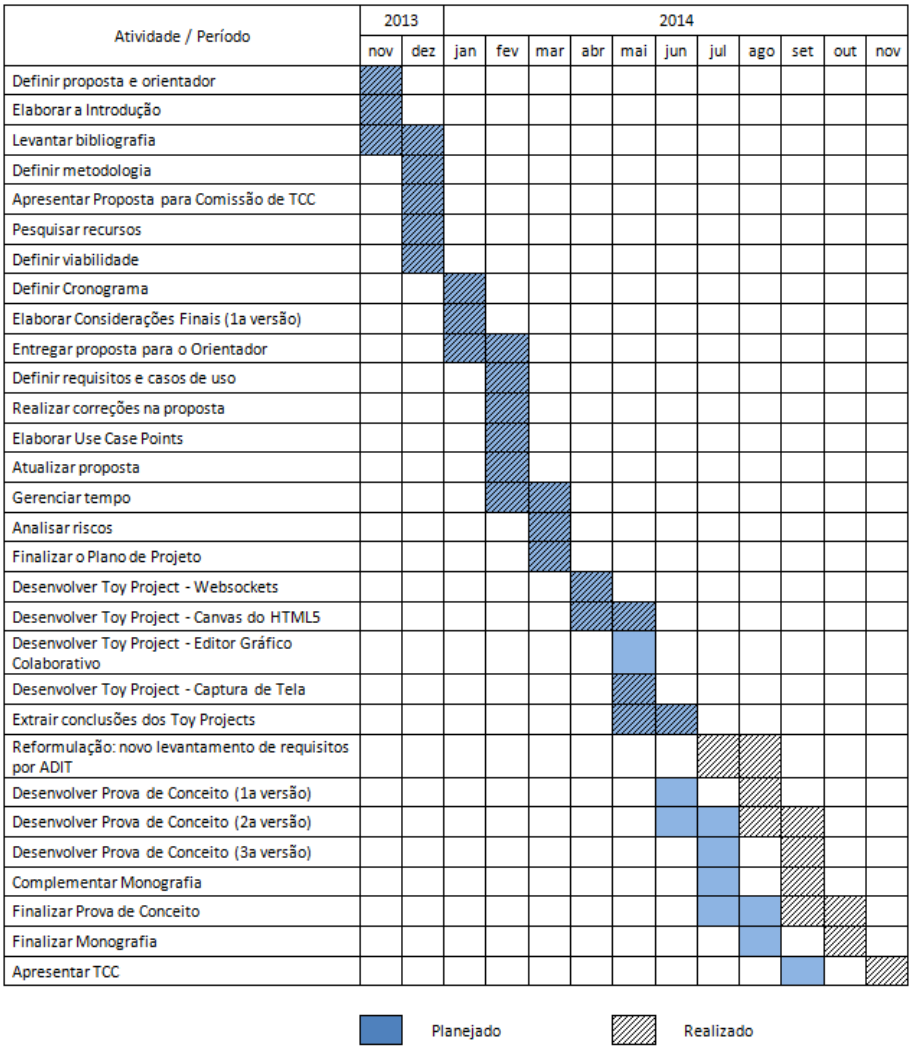

Figura 12: Comparação gráfica entre os cronogramas planejado e executado.

vistas no cronograma. Algumas das dificuldades encontradas no desenvolvimento foram: o emprego de tecnologias novas, ainda não suportada por todas as ferramentas utilizadas ou com uso limitado, como por exemplo, o uso de Web Workers para paralelizar a execução de código JavaScript. Essa tecnologia só é suportada pelo navegador Chrome, por motivos de segurança, se a thread que contem o Web Worker for hospedada em um servidor; sua execução local não é permitida. Outra limitação foi encontrada no uso de HTML5: como tecnologia em desenvolvimento, é comum que os navegadores não tenham todas as funcionalidades da documentação implementadas.

Quanto às horas de trabalho empregadas foi feita uma projeção de aproximadamente 700 horas de trabalhos necessárias para desenvolvimento do trabalho, conforme a estimativa por pontos de casos de uso, apresentado em detalhes no Apêndice A (Equação A.10). A realidade, entretanto, se mostrou muito diferente da projeção, superando as 1300 horas utilizadas pelas equipe para o desenvolvimento, quase alcançando o dobro da projeção, sem contar as horas

utilizadas para reprojetar o software por ADIT, o qual gerou um gasto adicional de 400 horas de trabalho, totalizado ao final por 1700 horas de trabalho. As horas excedentes indicam que havia uma grande subestimação do problema durante o planejamento, no qual diversos outros Casos de Uso necessários para implementação do protótipo não foram previstos, como indicado durante o reprojeto por ADIT.

6.2 QUANTO AOS ASPECTOS METODOLÓGICOS

Um conceito importante da área de desenvolvimento de software é a de reutilização de código. Apesar de ser incentivada, foram recorrentes, durante as pesquisas executadas ao longo do TCC na busca de soluções que atendessem as necessidades de projeto, códigos fonte interessantes, mas com pouquíssima capacidade de reutilização, devido ao forte acoplamento das soluções de diversos problemas em blocos que poderiam ser facilmente divididos em sub-componentes.

Sob a ideia de evitar os mesmos erros de reutilização de código o projeto se aproveitou de duas ideias fundamentais: o *Design by Contract* e os *Design Patterns*.

Apesar deste trabalho ter sido em parte focado em um processo de engenharia de software bem estruturado, ele não seguiu as recomendações de um programa de qualidade de software. Um programa de qualidade de software visa melhorar a capacidade de desenvolvimento de software considerando normas e modelos internacionalmente reconhecidos, boas práticas da engenharia de software e as necessidades de negócio da indústria (SOFTEX, 2014), como, por exemplo, o MPS.BR (Melhoria de Processo do Software Brasileiro). Ele é um programa em conformidade com as normas ISO/IEC 12207 e 15504, compatível com CMMI - DEV (um modelo de maturidade para melhoria de processos, destinado ao desenvolvimento e manutenção de produtos e serviços, e composto por práticas que cobrem o ciclo de vida do produto) e adequado à realidade brasileira (SOFTEX, 2012). Em forma resumida, o MPS.BR é uma certificação de processo de produção de software com base numa padronização de recomendações que visam melhorar a qualidade do processo.

O MPS.BR tem um largo escopo, abrangendo desde os passos de projeto de software em si até a gerência em alto nível dos projetos da empresa, o que tornaria interessante para uma eventual tomada de rumo para o desenvolvimento de um produto comercial. Para este TCC as recomendações a nível G (gerência de projeto e requisitos) seria mais do que o suficiente, mas a equipe decidiu por não seguir este nível de recomendações devido a necessidade de recursos extras, já demasiadamente escassos, para seguir um programa de qualidade, o que excederia ainda mais o custo de horas do projeto.

6.3 DO PROTÓTIPO AO PRODUTO

O resultado apresentado ao fim do projeto é uma aplicação protótipo, ideal para utilização gratuita, de teste ou avaliação, longe de ser um produto viável comercial devido à falta de polimento em diversos aspectos, como interface gráfica ou facilidade de uso. Direcionada para esse tema, esta seção discute acerca de algumas ideias que podem fazer o protótipo deste TCC se tornar um possível produto comercial, desde aspectos de segurança até de arquitetura.

O primeiro aspecto diz respeito aos papeis assumidos pelas partes devido à arquitetura de cliente-servidor da aplicação. A atual estrutura da aplicação não encapsula muitas das suas funções e por consequência, obriga o usuário a ter conhecimento do funcionamento de segurança em sistemas de informação e redes de computadores, como, por exemplo: saber o que é e possuir um certificado digital, ter conhecimento da configuração de uma rede e do firewall que eventualmente protege os componentes da rede, saber precisamente qual o endereço IP e porta utilizados pela aplicação, entre outros... A retirada destes ônus do usuário pode ser facilmente alcançada com uma mudança de arquitetura, em que o atual esquema cliente-servidor gerenciados pelos usuários da aplicação dão lugar a um servidor privado central que registra os clientes que permitem o controle de suas respectivas áreas de trabalho e dá fácil acesso a clientes que os desejam controlá-los. O servidor central abre possibilidade para diversas melhorias para o usuário: ao invés do usuário ter de digitar todo o “IP:porta”, que em geral é dinâmico para a maioria, ou seja, varia ao longo do tempo, seria necessário apenas entrar na página do serviço, por exemplo, srdweb.com.br, e ter a disposição um identificador único, de fácil memorização para o usuário, como um nome ou e-mail, que identificam a máquina a ser acessada na lista do servidor. Além disso, o servidor central também centraliza a necessidade de um certificado digital para ambas as pontas da aplicação, podendo utilizar inclusive um certificado válido ao invés de auto assinado, garantindo desta forma uma comunicação segura sem a necessidade de o usuário saber o que é um certificado digital ou como gerá-lo. Os gastos com recursos nesse caso se resumem aos seguintes elementos: custo de ter um domínio, custo de uma máquina servidora , custo de um certificado assinado por uma CA reconhecida e por último, o custo de desenvolver uma aplicação servidora.

O segundo aspecto diz respeito a possíveis estratégias de retorno financeiro, assumindo que a aplicação tenha se tornado uma viável aplicação comercial. Isso não é facilmente alcançado com a arquitetura atual, mas a utilização de um servidor central abre possibilidade de fornecer serviços mediante contratação. Como o serviço de segurança obriga a passagem dos dados pelo servidor, o que implica em gasto com banda, que é geralmente limitada na maioria dos contratos de servidores, o trafego sob SSL/TLS poderia ser limitado apenas a usuários que

pagam pelo serviço de área de trabalho remota, enquanto os demais usuários podem se aproveitar do serviço de indexação e fácil acesso a máquina remota, mas se conectam diretamente e não passam pela criptografia no canal de comunicação.

Um último aspecto serve apenas como nota para este trabalho, pois nada foi aplicado sob o ponto de vista do trade off eficiência versus legibilidade de código. Isso é importante ser considerado, pois os códigos fonte gerados para o sistema cliente em forma de HTML e JavaScript não são compilados e são enviados "*as it is*" para os clientes que enviam as requisições HTTP. Os códigos gerados neste TCC estão numa forma que pode ser denominada "para ambiente de desenvolvimento", pois eles estão numa forma legível para os desenvolvedores. Entretanto, isso implica num excesso de redundâncias apenas para fins de legibilidade e que não fazem diferença semântica do ponto de vista da máquina, como "*line feeds*", identificadores de funções com nomes legíveis, entre outros. A eliminação de caracteres desnecessários e/ou redundantes diminui o tamanho do arquivo com o código fonte, implicando num alívio da carga no canal de comunicação, sem interferir no desempenho da aplicação. Numa eventual aplicação comercial, com uso em "ambiente de produção", a compactação do código fonte é um ponto essencial e que de forma alguma deveria ser esquecida.

6.4 SUGESTÕES DE TRABALHOS FUTUROS

O projeto está longe de atingir um estado final, sendo passível de diversas melhorias e algumas delas estão listadas nesta seção, divididas sob três faces: melhorias de caráter de interface com o usuário, a nível de aspectos não funcionais e a nível de otimização.

Na interface com o usuário, apesar de ter sido previsto durante a fase de projeto, é notável o excesso de simplificação ou a ausência de uma interface gráfica. Isso abre a possibilidade de futuros trabalhos no design e implementação de uma interface para o servidor, principalmente na fase de configuração, e na melhoria da interface do cliente, que está excessivamente simplificada. Outro aspecto não abordado pelo escopo deste projeto foi a passagem de outros tipos de dados, além dos quadros da área de trabalho remota, notáveis pelo audio e pela troca de arquivos entre cliente e servidor.

Nos requisitos não funcionais, diversos aspectos já implementados no protótipo poderiam receber algum tipo de melhoria, entre eles estão: melhora do esquema de encoding do *framebuffer* e rever a possibilidade de repassar vídeos ao invés de apenas imagens, melhorar o envio de eventos de mouse/teclado, que ainda apresenta algumas imperfeições, pertinentes a especificidades de plataforma, utilização de *canvas layering* no cliente.

Na otimização, pouco foi implementado neste trabalho, que poderiam ser representados pelos aspectos de: criar melhorias a nível de sistema operacional, como implementação de drivers que interceptem diretamente os *framebuffers* do vídeo para algum tipo de processamento a baixo nível do sinal e diminuir assim o tempo de processamento e o tamanho do *framebuffer*; uso de recursos específicos de cada navegador web para acelerar o processamento, como o NACL do Google Chrome (CHROME, 2014).

Apesar de haverem diversas frentes de melhoria em aberto neste TCC, é interessante notar que muitas das mudanças propostas correm contra o princípio da portabilidade e em direção à otimização personalizada para plataformas específicas.

6.5 CONSIDERAÇÕES FINAIS DO CAPÍTULO

O gasto do recurso de tempo no desenvolvimento do projeto refletiu os problemas de projeto do software, no qual foram subestimados a quantidade de casos de uso que o software realmente deveria abranger. Entretanto, de um modo geral, o atraso no cronograma executado ficou visível apenas no reprojeto do software, o qual foram consumidos aproximadamente 2 meses de trabalho, sendo que os demais itens do cronograma mantiveram a projeção do período de tempo durante a fase de projeto.

A melhoria do desenvolvimento de software pode ser abordado de diversas formas. Este TCC focou-se apenas no processo de desenvolvimento (ADIT), mas muitas outras faces do problemas tiveram seus riscos aceitos apenas, sem nenhuma medida de garantia de qualidade do processo, garantido por padronizações como a MPS.BR, aqui citado como importância principalmente no caso de interesse de investimento do protótipo para torná-lo uma solução comercial.

Com relação à melhoria e transformação do resultado num produto comercial diversas modificações podem ser feitas. Entretanto, a alteração de maior impacto é a de modificar a arquitetura do software em benefício da facilidade de utilização da aplicação para o usuário. De um ponto de vista mais funcional, o resultado deste TCC ainda é acometido por uma ligeira falha de desempenho, o qual pode ser melhorado em diversos aspectos, como processo de Canvas Layer no cliente e novos processos de codificação da imagem da tela da Área de Trabalho do servidor.

7 CONSIDERAÇÕES FINAIS

Considera-se que o trabalho foi bem sucedido pois atendeu ao planejamento, cumpriu os requisitos e durante o desenvolvimento proporcionou aprendizado e resultou em uma ferramenta funcional para acesso a áreas de trabalho remotas.

A revisão bibliográfica se mostrou importante, pois algumas decisões de projeto e alterações ocorreram com base nas sugestões presentes nas referências levantadas durante a fundamentação teórica. Foi o caso do protocolo desenvolvido - baseado no RFB, o uso de Web Workers para paralelizar a execução de código no cliente, o uso de TSL/SSL para segurança, a escolha de uma sequência de imagens com compressão em detrimento de um algoritmo de compressão de vídeo, as linguagens de programação HTML5 e JavaScript, a tecnologia WebSocket para a comunicação entre cliente e servidor, entre outros. Também aproveitando os dados do levantamento bibliográfico, garantiu-se a reutilização do código pelo emprego de *Design Patterns*.

No que diz respeito ao desenvolvimento, foi dado o início utilizando-se as ferramentas desenhadas pelo padrão UML. Embora elas possuam seu mérito e sejam bastante úteis para o desenvolvimento, elas não definem como a modelagem do sistema deve ser concebida. Dessa forma, o modelo foi completado fazendo uso do método ADIT que, trabalhado de maneira incremental e iterativa, permite a aquisição de mais conhecimento relacionado ao domínio do trabalho a cada iteração alcançada durante o desenvolvimento de software. A natural ligação das fases do ADIT faz com que um processo de desenvolvimento em V valide o resultado do projeto com os objetivos propostos.

Durante a fase de testes, mudanças foram efetuadas em aspectos não funcionais durante os testes de aceitação. As principais alterações foram na quantidade de eventos de mouse e teclado enviados e em condições relacionadas à velocidade de atualização do Canvas.

Quanto ao planejamento de tempo, houve um acréscimo de horas trabalhadas em relação ao que foi planejado. Essa mudança se deu por decisão dos desenvolvedores em usar uma fatia de tempo maior com o projeto e modelagem do software para facilitar a posterior

implementação.

Entre os aprendizados obtidos durante o desenvolvimento, cita-se: o uso de novas linguagens como HTML5 e JavaScript, o trabalho relacionado a codificação e compressão de imagens para envio na rede, o uso de WebSockets e tópicos em segurança de sistemas de informação.

O projeto ainda está longe de atingir um estado final, podendo ser realizadas diversas melhorias. A interface com o usuário foi bastante simplificada abrindo espaço para futuros trabalhos relacionados ao design. Outro aspecto não abordado no escopo deste projeto foi a passagem de outros tipos de dados entre cliente e servidor como áudio e a possibilidade de realizar troca de arquivos.

Nos requisitos não funcionais, diversos aspectos poderiam ser melhorados: o esquema de encoding do *framebuffer*, a realização de um estudo mais profundo relacionado a possibilidade de repassar vídeos ao invés de apenas imagens, aperfeiçoar o envio de eventos de mouse/teclado e utilização de *canvas layering* no cliente. Quanto à otimização, poderiam ser realizadas melhorias a nível de sistema operacional, como implementação de drivers que interceptem diretamente os *framebuffers* do vídeo para algum tipo de processamento a baixo nível do sinal e diminuir assim o tempo de processamento e o tamanho do *framebuffer*, uso de recursos específicos de cada navegador web para acelerar o processamento, como o NACL do Google Chrome.

Embora hajam vários ramos de melhoria em aberto, nota-se que algumas mudanças são contrárias ao princípio de portabilidade e de uma ferramenta genérica, levando a uma otimização sob medida para plataformas específicas.

REFERÊNCIAS

BIDELMAN, E. **The Basics of Web Workers**. out. 2014. Disponível em: <http://www.html5rocks.com/en/tutorials/workers/basics/>.

CHAFFER, J.; SWEDBERG, K. **Learning JQuery 1.3: Better Interaction and Web Development with Simple JavaScript Techniques**. [S.l.]: Packt Publishing, 2009. 444 p.

CHEN, B.; XU, Z. A framework for browser-based multiplayer online games using webgl and websocket. **2011 International Conference on Multimedia Technology (ICMT)**, p. 471-474, 2011.

CHROME, G. **NaCl and PNaCl**. dez. 2014. Disponível em: <https://developer.chrome.com/native-client/nacl-and-pnacl>.

CROCKFORD, D. **The application/json Media Type for JavaScript Object Notation (JSON)**. [S.l.], jul. 2006.

CROCKFORD, D. **JavaScript: The Good Parts: The Good Parts**. [S.l.]: O'Reilly Media, Inc., 2008. 172 p.

DAMM, S. **HTML 5, CSS 3 + JavaScript IDE shootout - A comparison of tools for the development of HTML 5 Applications**. dez. 2013. Disponível em: <http://www.oio.de/public/opensource/comparison-IDE-for-HTML5-CSS3-JavaScript-shootout.htm>.

DARMSTADT, T. U. **Zertifikate (Server, Nutzer, PKI)**. out. 2014. Disponível em: <http://www.hrz.tu-darmstadt.de/itsicherheit/pki/pki_tudcag01/keytool.de.jsp>.

DEVLOOP. **About - Xpra**. dez. 2014. Disponível em: <https://www.xpra.org/trac/wiki/About>.

DIERKS, T.; RESCORLA, E. **The Transport Layer Security (TLS) Protocol**. [S.l.], ago. 2008.

ECLIPSE, T. F. **Jetty**. out. 2014. Disponível em: <http://www.eclipse.org/jetty/>.

FETTE, I.; MELNIKOV, A. **The WebSocket Protocol**. [S.l.], dez. 2011.

FLANAGAN, D. **JavaScript: the definitive guide. 3rd ed.** [S.l.]: O'Reilly Media, Inc., 1998. 776 p.

FUKAI, Y. et al. Web browser based gui for tv. **1st IEEE Global Conference on Consumer Electronics**, p. 579-580, 2012.

GAMMA, E. **Design patterns: elements of reusable object-oriented software.** [S.l.]: Addison-Wesley, 1995. 395 p.

GOODMAN, D. **JavaScript, a Bíblia**. [S.l.]: Campus Ed., 2001. 909 p.

GOOGLE. **Der Chrome-Browser - Sicherheit**. dez. 2013. Disponível em: <https://www.google.com/intl/de/chrome/browser/features.html# security>.

JSON, G. **Introducing JSON**. out. 2014. Disponível em: <http://json.org>.

JUMPER, M. **Guacamole Manual, Introduction**. nov. 2013. Disponível em: <http://guac-dev.org/doc/gug/preface.html>.

KAAZING. **Transport Layer Security (TLS/SSL) Concepts**. nov. 2014. Disponível em: <http://developer.kaazing.com/documentation/jms/4.0/security/c_tls.html>.

KARNER, G. Resource estimation for objectory projects. **Objective Systems SF AB**, 1993.

KO, H.-Y.; LEE, J.-H.; KIM, J.-O. Implementation and evaluation of fast mobile vnc systems. **IEEE Transactions on Consumer Electronics**, v. 58, n. 4, p. 1211-1218, 2012.

LAMPING, U.; SHARPE, R.; WARNICKE, E. **Wireshark User's Guide**. [S.l.], 2013. V1.11.3-rc1-1925-g0f73f79.

LARMAN, C. **Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo**. [S.l.]: Bookman, 2007.

LAWSON, B.; SHARP, R. **Introducing HTML5**. [S.l.]: New Riders, 2011. 312 p.

LEVITTE, R. **HOWTO certificates**. out. 2014. Disponível em: <https://www.openssl.org/docs/HOWTO/certificates.txt>.

LIAO, X. et al. Towards virtualized desktop environment. **Concurrency and Computation: Practice and Experience**, v. 22, p. 419-440, 2010.

MARTIN, J.; AL. et. **Bug 594502 - Method for accepting certificates for secure WebSockets connections (wss://)**. set. 2014. Disponível em: <https://bugzilla.mozilla.org/show_bug.cgi?id=594502>.

MARTIN, J.; LEVY, O. **noVNC: HTML5 VNC Client**. nov. 2013. Disponível em: <https://github.com/kanaka/noVNC/blob/master/README.md>.

MAZIERO, C. A. **Sistemas Operacionais: Conceitos e Mecanismos**. [S.l.: s.n.], 2013.

MOZILLA, D. N. **WebSockets**. out. 2014. Disponível em: <https://developer.mozilla.org/de/docs/WebSockets>.

MSDN. **Remote Desktop Protocol**. dez. 2014. Disponível em: <http://msdn.microsoft.com/en-us/library/aa383015.aspx>.

NEGRINO, T.; SMITH, D. **JavaScript for the World Wide Web**. [S.l.]: Pearson Education, 2009. 430 p.

OLIVETTI; ORACLE, R. L. **rfbProto**. 1998. Disponível em: <http://stuff.mit.edu/afs/athena/project/ssh/mindterm/mindbright.old/vnc/rfbProto.java>.

ORACLE. **18 Java API for WebSocket**. out. 2014. Disponível em: <http://docs.oracle.com/javaee/7/tutorial/doc/websocket.htm>.

ORACLE. **Debugging and Testing JavaScript in an HTML5 Application**. mar. 2014. Disponível em: <https://netbeans.org/kb/docs/webclient/html5-js-support.html>.

ORACLE. **Hilfreiche Konzepte und Glossar mit Definitionen**. out. 2014. Disponível em: <https://java.com/de/download/faq/helpful_concepts.xml>.

ORACLE. **Java Platform, Standard Edition 7 API Specification**. out. 2014. Disponível em: <http://docs.oracle.com/javase/7/docs/api/>.

ORACLE. **VirtualBox**. out. 2014. Disponível em: <https://www.virtualbox.org>.

PÉREZ, C. T. **Web-based collaboration through screen sharing**. Master's Thesis — Luleå University of Technology, Luleå, Sweden, 2011.

PILGRIM, M. **HTML5: Up and Running**. [S.l.]: O'Reilly Media, Inc., 2010. 222 p.

PIMENTEL, V.; NICKERSON, B. G. Communicating and displaying real-time data with websocket. **IEEE Internet Computing**, v. 16, n. 4, p. 45-53, 2012.

RAJLICH, N.; ROHMER, D. **Java-WebSocket**. out. 2014. Disponível em: <http://java-websocket.org>.

REALVNC, L. **RealVNC**. out. 2014. Disponível em: <https://www.realvnc.com>.

RICHARDSON, T. **The RFB Protocol**. [S.l.], nov. 2010.

ROUSSET, D. **Introduction to HTML5 Web Workers: The JavaScript Multi-threading Approach**. out. 2014. Disponível em: <http://msdn.microsoft.com/en-us/hh549259.aspx>.

SOFTEX. **MPS.BR - Melhoria de Processo do Software Brasileiro - Guia Geral MPS de Software**. [S.l.], dez. 2012.

SOFTEX. **MPS.BR | Softex**. out. 2014. Disponível em: <http://www.softex.br/mpsbr/mps/mps-br-em-numeros/>.

SYMANTEC, C. **Symantec Secure Site Pro SSL-Zertifikate**. out. 2014. Disponível em: <http://www.symantec.com/de/de/ssl-certificates/secure-site-pro>.

TIGHTVNC, S. **TightVNC**. out. 2014. Disponível em: <http://www.tightvnc.com>.

VMWARE. **VMWare Fusion**. out. 2014. Disponível em: <http://www.vmware.com/products/fusion>.

WESSELS, A. et al. Remote data visualization through websockets. **Eighth International Conference on Information Technology: New Generations**, p. 1050-1051, 2011.

WILKINS, G. et al. **Jetty/Tutorial/Embedding Jetty**. out. 2014. Disponível em: <https://wiki.eclipse.org/Jetty/Tutorial/Embedding_Jetty>.

ZHU, G. et al. Html5 based media player for real-time video surveillance. **5th International Congress on Image and Signal Processing**, p. 245-248, 2012.

APÊNDICE A – O LEVANTAMENTO DE REQUISITOS POR CASOS DE USO

Este apêndice contém o documento que descreve a abordagem do levantamento de requisitos por casos de uso executada durante a fase de proposta de projeto deste TCC. Ela está aqui colocada para fins de arquivamento e comparação com os resultados do ADIT, abordagem utilizada posteriormente no desenvolvimento do projeto. Este documento não reflete o atual estado do protótipo ou dos resultados do TCC.

A.1 LEVANTAMENTO DE REQUISITOS

A primeira parte do projeto consistiu no levantamento de requisitos do sistema. Foram enumerados os dois tipos de requisitos, os funcionais, que estão diretamente ligados ao comportamento, e os não-funcionais, que ditam a forma como o sistema funciona. Os requisitos obtidos para este projeto são:

Requisitos funcionais:

FR 1– O sistema deve mostrar a área de trabalho do desktop remoto no navegador.

FR 2– O sistema deve permitir que o cliente lance aplicações no desktop remoto.

FR 3– O sistema deve permitir que o cliente encerre aplicações no desktop remoto.

FR 4– O sistema deve interagir com as aplicações remotas através do mouse local.

FR 5– O sistema deve interagir com as aplicações remotas através do teclado local.

Requisitos não-funcionais:

NFR 1– O tempo de atraso entre envio e disposição da imagem da área de trabalho remota no navegador não deve ser superior a 3 segundos.

NFR 2– Cliente e servidor devem estar conectados fisicamente pela rede

NFR 3– O servidor deve estar ligado e com o aplicativo funcionando no momento em que o cliente for estabelecer uma conexão

NFR 4– O sistema deve estabelecer uma conexão segura entre cliente e servidor

A.2 CASOS DE USO

A partir dos requisitos, as interações e o comportamento do sistema são modeladas de forma abstrata a partir dos casos de uso, que estão detalhados sob pré e pós-condições, fluxo de funcionamento e as relações com atores do sistema. Os casos de uso foram enumerados nas Tabelas 2, 3, 4, 5 e 6. Por fim a Figura 13 mostra visualmente os casos de uso e suas relações.

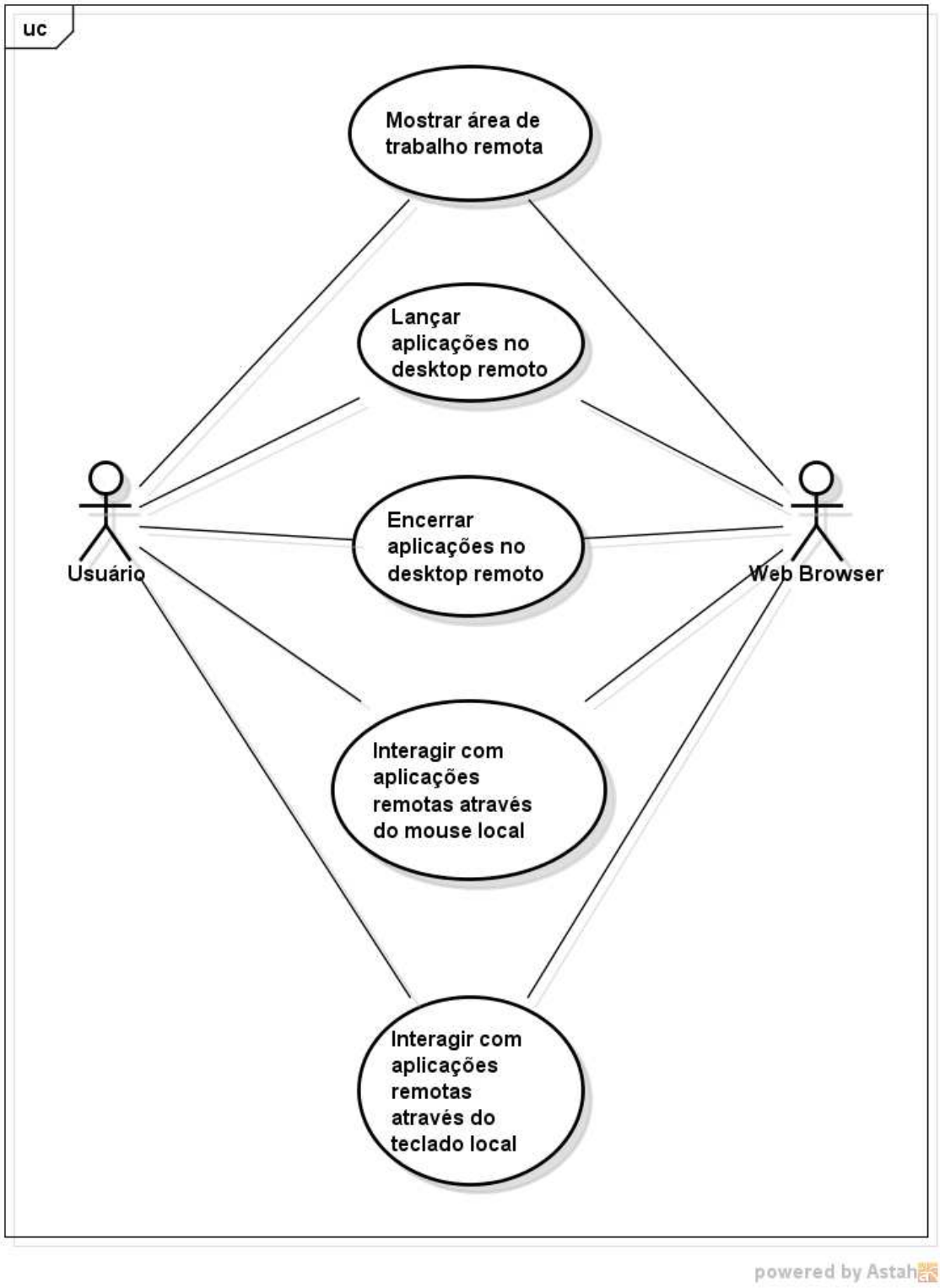

Figura 13: Diagrama de casos de uso do projeto.

Tabela 2: Caso de uso 1 – Mostrar área de trabalho remota.

Caso de Uso	UC 1 – Mostrar área de trabalho remota
Descrição	Caso de uso executado quando o cliente está conectado ao servidor
Ator Principal	Usuário
Ator de Suporte	Web Browser
Pré-condições	Cliente conectado ao servidor
Pós-condições	Área de trabalho remota exibida no navegador do cliente
Fluxo Básico	•Cliente solicita uma atualização de tela ao servidor •Servidor envia as informações da tela ao cliente •As informações são exibidas no web browser
Fluxo Alternativo	•A qualquer momento o cliente encerra a conexão •O sistema encerra o caso de uso

Tabela 3: Caso de uso 2 – Lançar aplicações no desktop remoto.

Caso de Uso	UC 2 – Lançar aplicações no desktop remoto
Descrição	Caso de uso executado quando o usuário abre uma aplicação no desktop remoto
Ator Principal	Usuário
Ator de Suporte	Web Browser
Pré-condições	Cliente conectado ao servidor
Pós-condições	Aplicação lançada no desktop remoto
Fluxo Básico	•Usuário lança uma aplicação remota como se fosse local •O evento é enviado ao servidor •A aplicação é lançada no desktop remoto
Fluxo Alternativo	•A qualquer momento o cliente encerra a conexão O sistema encerra o caso de uso.

Tabela 4: Caso de uso 3 – Encerrar aplicações no desktop remoto.

Caso de Uso	UC 3 – Encerrar aplicações no desktop remoto
Descrição	Caso de uso executado quando o usuário encerra uma aplicação no desktop remoto
Ator Principal	Usuário
Pré-condições	Cliente conectado ao servidor
Pós-condições	Evento de mouse enviado ao servidor
Fluxo Básico	•Usuário encerra uma aplicação remota como se fosse local •O evento é enviado ao servidor •A aplicação é encerrada no desktop remoto
Fluxo Alternativo	•A qualquer momento o cliente encerra a conexão O sistema encerra o caso de uso.

Tabela 5: Caso de uso 4 – Interagir com aplicações remotas através do mouse local.

Caso de Uso	UC 4 – Interagir com aplicações remotas através do mouse local
Descrição	Caso de uso executado quando o usuário clica com o mouse local
Ator Principal	Usuário
Ator de Suporte	Web Browser
Pré-condições	Cliente conectado ao servidor
Pós-condições	Evento do mouse local executado no desktop remoto
Fluxo Básico	•Usuário dispara algum evento do mouse •Cliente detecta o evento do mouse •Cliente envia ao servidor o evento do mouse •Servidor executa o evento realizado com o mouse no cliente
Fluxo Alternativo	•A qualquer momento o cliente encerra a conexão O sistema encerra o caso de uso.

Tabela 6: Caso de uso 5 – Interagir com aplicações remotas através do teclado local.

Caso de Uso	UC 5 – Interagir com aplicações remotas através do teclado local
Descrição	Caso de uso executado quando o usuário digita no teclado local
Ator Principal	Usuário
Ator de Suporte	Web Browser
Pré-condições	Cliente conectado ao servidor
Pós-condições	Evento do teclado local executado no desktop remoto
Fluxo Básico	•Usuário dispara algum evento do teclado •Cliente detecta o evento do mouse •Cliente envia ao servidor o evento do teclado •Servidor executa o evento realizado com o teclado no cliente
Fluxo Alternativo	•A qualquer momento o cliente encerra a conexão O sistema encerra o caso de uso.

A.3 ESTIMATIVA ATRAVÉS DE PONTOS DE CASO DE USO

A partir dos casos de uso, uma ferramenta adicional para projeto de sistemas é a estimativa de tempo necessário para execução do projeto através dos pontos de casos de uso, ou *Use Case Points*, uma técnica proposta por Gustav Kamer para medir o tamanho funcional de sistemas (KARNER, 1993).

O primeiro passo é classificar os atores de cada caso de uso em simples, médio e complexo. Um ator simples consiste em um outro sistema acessado através de uma *API* de programação. Ator médio é aquele representado por um outro sistema que interage através de um protocolo de comunicação. Por fim, um ator complexo é um usuário interagindo através de uma interface gráfica. Através da soma dos produtos dos número de atores de cada tipo pelo seu peso obtem-se o UAW (*Unadjusted Actor Weight*), conforme mostrado na Tabela 7.

Tabela 7: *Unadjusted Actor Weight*.

Tipo de Ator	**Peso**	**Atores**	**Resultado**
Ator Simples	1	0	0
Ator Médio	2	1	2
Ator Complexo	3	1	3
Total de *UAW*			**5**

O segundo passo é o cálculo do peso dos casos de uso UUCW (*Unadjusted Use Case Weight*). Os casos de uso também são classificados de acordo com sua complexidade. Um caso de uso simples é aquele em que há até 3 transações (incluindo o fluxo alternativo), um caso de uso médio possui de 4 a 7 transações e o complexo, mais de 7 transações. Os resultados são exibidos na Tabela 8.

Tabela 8: *Unadjusted Use Case Weight*.

Tipo de Caso de Uso	**Peso**	**Atores**	**Resultado**
Casos de Uso Simples	5	0	0
Casos de Uso Médio	10	5	50
Casos de Uso Complexo	15	0	0
Total de *UUCW*			**50**

O terceiro passo é o cálculo dos pontos de caso de uso não ajustados UUCP (*Unadjusted Use Case Point*), que é dado pela Equação A.1.

$$UUCP = UnadjustedActorWeight(UAW) + UnadjustedUseCaseWeight(UUCW) \quad \text{(A.1)}$$

$$UUCP = 50 + 5 = 55 \quad \text{(A.2)}$$

Após o cálculo do *UUCP*, procede-se com cálculos dos fatores de ajuste, que compreendem fatores técnicos (relacionados a requisitos funcionais) e de ambiente (relacionados a requisitos não funcionais).

A tabela 9 expõem os 13 fatores técnicos, seus pesos e o valor de cada um. O valor 0 significa que o fator é irrelevante e 5 representa algo essencial.

Tabela 9: Fatores Técnicos.

Fator	Fatores que contribuem para a complexidade	Peso	Valor	Total
F1	Sistemas distribuídos	2	4	8
F2	Tempo de resposta	1	1	1
F3	Eficiência para o usuário final (on-line)	1	1	1
F4	Processamento interno complexo	1	1	1
F5	Código reusável	1	0	0
F6	Facilidade de instalação	0,5	0	0
F7	Facilidade de uso (facilidade operacional)	0,5	1	0,5
F8	Portabilidade	2	2	4
F9	Facilidade de mudança	1	0	0
F10	Concorrência (acesso simultâneo à aplicação)	1	0	0
F11	Recursos de segurança	1	4	4
F12	Fornece acesso direto para terceiros	1	0	0
F13	Requer treinamento especial para o usuário	1	0	0
TFactor				**19,5**

O valor do fator de complexidade técnica TCF (*Technical Complexity Factor*) é calculado através da equação A.3.

$$TCF = 0,6 + (0,01 x TFactor) \tag{A.3}$$

$$TCF = 0,6 + (0,01 x 19,5) = 0,795 \tag{A.4}$$

A seguir são considerados os fatores ambientais. Da mesma forma que nos fatores técnicos, 0 significa que o fator é irrelevante e 5, essencial. A tabela 10 mostra esses valores.

Tabela 10: Fatores Ambientais.

Fator	Fatores que contribuem para a complexidade	Peso	Valor	Total
F1	Familiaridade da equipe com o processo formal de desenvolvimento adotado	1,5	2	3
F2	Colaboradores de meio período	-1	0	0
F3	Capacidade do líder do projeto em análise de requisitos e modelagem	0,5	2	1
F4	Experiência da equipe em desenvolvimento de aplicações do gênero em questão	0,5	4	2
F5	Experiência em Orientação a Objetos	1	2	2
F6	Motivação da equipe	1	4	4
F7	Dificuldades com a linguagem de programação	-1	0	0
F8	Requisitos estáveis	2	4	8
Efactor				**20**

O Fator de Complexidade de Ambiente ECF (*Environmental Complexity Factor*) é calculado através da Equação A.5.

$$ECF = 1.4 + (-0.03 x Efactor) \tag{A.5}$$

$$ECF = 1.4 + (-0.03 x 20) = 0.8 \tag{A.6}$$

Por fim, obtem-se o valor total do sistema em UCP (*Use Case Points*) ajustados através da equação A.7. O tempo necessário para o desenvolvimento de um projeto pode ser estimado com uma média de 20 horas de trabalho por *UCP*, conforme equação A.9. A equação A.10 representa o valor total em horas utilizado para o desenvolvimento.

$$UCP = UUCP x TCF x EF \tag{A.7}$$

$$UCP = 55 x 0,795 x 0,8 = 34,98 \tag{A.8}$$

$$Tempoestimado = UCP x 20 \tag{A.9}$$

$$Tempoestimado = 34,98 x 20 = 699,6 \tag{A.10}$$

A.4 PROCEDIMENTOS DE TESTE E VALIDAÇÃO

Como forma de assegurar o funcionamento do produto e minimizar erros de programação a fase de testes se torna importante dentro do projeto, e também é, em geral, a mais longa entre as demais.

Com relação à codificação, o JavaScript domina a implementação do projeto, por essa razão os primeiros testes são sobre os módulos desenvolvidos nessa linguagem. Para isso ferramentas de teste estão disponíveis, variando conforme o nível e a necessidade do desenvolvimento do sistema.

Num primeiro momento as funções da implementação são testadas através de ferramentas de teste unitário, que fornecem entradas para cada uma das funções ou módulos do sistema e avaliam-se suas respectivas respostas conforme o resultado esperado. Alguns exem-

plos são os próprios plugins para IDEs que implementam a integração com navegadores web e as bibliotecas de teste unitário, como no Netbeans, que apresenta um depurador de um test driver para a tarefa (ORACLE, 2014b).

O passo intermediário está nos testes de integração dos módulos do sistema. Eles são definidos por base nas relações, portanto são definidos num estágio mais avançado do projeto.

Os testes e validações num sistema como um todo são feitos em caixa preta, ou seja, sem dar atenção a estrutura do programa, mas as respostas do sistema diante das interações. Elas serão feitas com base nos requisitos e casos de uso levantados no capítulo 5 e verificar visualmente ou através de ferramentas de auditoria se eles são atendidos ou não, da maneira como está estruturada na tabela 11.

No quesito segurança, ferramentas de rede são utilizados para verificação da comunicação entre as máquinas, o que dá possibilidade de demonstrar se existe um caminho de comunicação entre elas ou se os dados estavam devidamente criptografados. Uma dessas ferramentas é o Wireshark, que é um analisador de pacotes de rede, os quais são coletados e detalhados, com a finalidade de avaliar e depurar protocolos de comunicação e sua segurança (LAMPING et al., 2013).

Finalmente, a validação em alto nível se dá com a utilização do sistema completo. Ela pode ser realizada através da utilização do produto por terceiros, ou, em caso de restrição temporal/financeiro, será realizada pelos próprios integrantes, junto ao orientador do projeto.

Tabela 11: Procedimentos de teste conforme as exigências do sistema.

Exigência do Sistema	**Forma de Teste**
FR 1 – O sistema deve mostrar a área de trabalho do desktop remoto no navegador	Executar o sistema e verificar a imagem obtida.
FR 2 – O sistema deve permitir que o cliente lance aplicações no desktop remoto	Tentar abrir um programa com o sistema em execução, através do cliente e verificar se é aberto no servidor.
FR 3 – O sistema deve permitir que o cliente encerre aplicações no desktop remoto	Tentar fechar a aplicação com o sistema em execução, através do cliente e verificar se a aplicação é encerrada.
FR 4 – O sistema deve interagir com as aplicações remotas através do mouse local	Tentar utilizar uma aplicação com o sistema em execução, através do cliente e verificar se a aplicação responde.
FR 5 – O sistema deve interagir com as aplicações remotas através do teclado local	Tentar digitar um texto num campo de texto selecionado no servidor, através do cliente e com o sistema em execução.
NFR 1 – O tempo de atraso entre envio e disposição da imagem da área de trabalho remota no navegador não deve ser superior a 3 segundos.	Verificar se uma alteração gráfica (ex: mexer ponteiro do mouse, clicar num botão de menu, selecionar um icone) demora demais para ocorrer no lado do cliente enquanto o sistema está em execução.
NFR 2 – Cliente e servidor devem estar conectados fisicamente pela rede	Verificar através de ferramentas de rede, como ping, se o computador remoto responde.
NFR 3 – O servidor deve estar ligado e com o aplicativo funcionando no momento em que o cliente for estabelecer uma conexão	Executar o servidor e verificar se ele aceita conexões.
NFR 4 – O sistema deve estabelecer uma conexão segura entre cliente e servidor	Utilizar uma ferramenta de auditoria de rede, como o Wireshark, para capturar pacotes e verificar se estão criptografados ou não.

APÊNDICE B – ADIT – A1

Este documento tem o intuito de apresentar os resultados da fase A1. O documento foi reestruturado para ser inserido ao apêndice do relatório de TCC “ACESSO DE ÁREA DE TRABALHO REMOTA VIA APLICAÇÃO WEB PROTEGIDO POR CRIPTOGRAFIA”, da Engenharia de Computação da UTFPR campus Curitiba.

B.1 DESCRIÇÃO INFORMAL

O SRDWeB deve permitir o usuário visualizar e comandar a área de trabalho de seu computador remotamente, de forma segura e através de uma aplicação web. O protocolo do sistema será baseado no RFB (VNC) estendido. Uma parte do sistema consiste num programa servidor instalado na área de trabalho alvo, o qual disponibilizará uma conexão pela rede. O servidor em execução fica a espera de uma conexão, a partir do qual inicializará a execução do protocolo RFB com o cliente. Ao abrir a conexão: o servidor disponibiliza um simples servidor HTTP, com o qual o cliente poderá obter a aplicação web para controle da máquina remota; a partir da conexão HTTP o cliente não deve mais precisar entrar com o endereço de destino. O cliente entra com o endereço do servidor e recebe a aplicação web no seu navegador. A partir deste momento este estará conectado, enviando eventos de mouse e teclado e recebendo as imagens da máquina remota. A comunicação deve ser segura.

B.2 O DIAGRAMA DE CONTEXTO

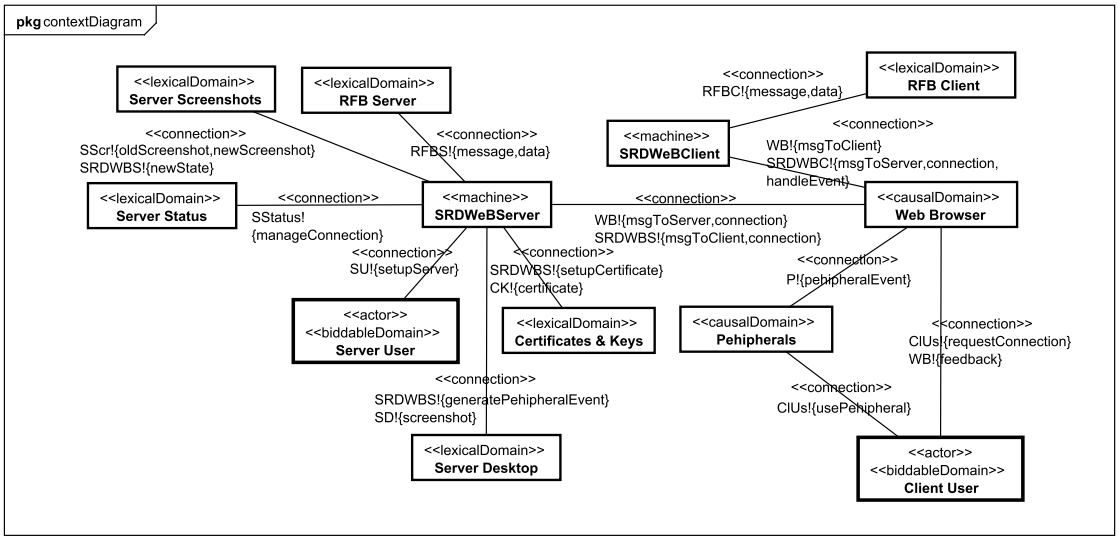

Figura 14: Diagrama de Contexto do projeto, levantado na fase A1.

B.3 A IDENTIFICAÇÃO DOS *STATEMENTS*

A primeira letra representa ao tipo do *statement*: A para suposições, F para fatos, R para requisitos, S para especificações. Os demais caracteres são o código de identificação do *statement*. Caso o código de identificação comece com uma letra, ela identifica a qual Outro *statement* ela faz referência, e todos os *statement* que fazem referência a outro *statement* são diferenciados por uma letra minúscula.

B.4 A LISTA DE *ASSUMPTIONS*

Suposições: lista de condições necessárias a respeito do ambiente para que os requisitos sejam alcançáveis.

(A01). Servidor possui conexão com a rede ou Internet.

(A02). Cliente possui conexão com a rede ou Internet.

(A03).O Firewall do Servidor permite requisições de conexão externas.

(A04).O Firewall do Cliente permite abrir conexões com a Internet.

(A05). Servidor possui JVM e executa programas em Java.

(A06).Máquina Cliente possui navegador web compatível com as tecnologias HTML5, JavaScript e WebSockets.

(A07). Cliente possui algum tipo de interface de vídeo ativa.

(A08). Cliente possui mouse e teclado.

(A09). Certificado Digital não é necessário para garantir Autenticidade.

(A10). Cliente e Servidor possuem configurações mínimas necessárias para executar os softwares necessários.

(A11). Comunicação é feita via mensagens e não caracteres.

(A12). Usuário possui privilégios elevados no Servidor.

B.4.1 REGISTRO DE ALTERAÇÕES DA LISTA

(AR01a). Client User conhece o endereço que identifica o Servidor na rede.

(AR01b). Servidor e cliente estão conectados.

(AR24a). Ao término de "generatePehipheralEvent" os eventos já afetaram o Sistema Operacional.

B.5 A LISTA DE *FACTS*

Fatos: descrição de propriedades fixas do ambiente, independente da forma como a "máquina" é construída.

(F01). Cada computador possui um endereço único na rede.

(F02). Apenas portas livres podem ser utilizadas.

(F03). As portas definem quais aplicações são acessados.

(F04).O endereço para acessar uma aplicação num computador remoto é definido por "IP:porta".

(F05).A conexão utilizada pelos WebSockets é do tipo confiável.

(F06).TLS/SSL garante confidencialidade e autenticidade com um certificado válido.

(F07).TLS/SSL garante apenas confidencialidade com um certificado auto assinado.

(F08).O certificado deste projeto é auto assinado.

(F09).O servidor é reativo.

B.5.1 REGISTRO DE ALTERAÇÕES DA LISTA

(FR01a). Web Browser e Web Server executam um protocolo que permite se encontrarem, abrirem conexão e trocar dados, desde que haja conectividade e endereço válido que identifiquem o Servidor na rede.

(FR01b). Web Browser recebe os dados de "newBC" e executa "newConnection" na tentativa de abrir uma conexão com Bidirectional Comm.

(FR01c). Web Browser e Bidirectional Comm executam um protocolo que permite se encontrarem, abrirem conexão e trocar dados, desde que haja conectividade e endereço válido que identifiquem o Servidor na rede.

(FR02a). Bidirectional Comm e Web Browser medeiam comunicação entre SRDWBS_Protocol e SRDWBC_Protocol.

(FR03a). Todo "newConnection" bem sucedido ao Bidirectional Comm é seguido pela execução de "onOpenToClient" e "onOpenToServer".

(FR03b). Server Status guarda as chamadas feitas por "increaseConnected" e "decreaseConnected".

(FR03c). Caso a diferença de ocorrências de "increaseConnected" por "decreaseConnected" seja igual ou superior a 2 após a chamada de um "increaseConnected", um "denyConnection" ocorre, caso contrário um "grantConnection" ocorre.

(FR03d). Bidirectional Comm e Web Browser medeiam a comunicação entre Cliente e Servidor.

(FR04a). Web Browser emite as mensagens de atualização do cliente via "sendUpdateToServer".

(FR05,06,07a).O Web Browser é responsável por inspecionar o comportamento de Timer, Pehipherals e os traduz para "handleEvent".

(FR09a). Server Setup medeia a comunicação entre Server User e "machine".

(FR16a). Um "requestPage" ocorre apenas quando há um "start" bem sucedido.

(FR22,23a).O Web Browser envia mensagem de atualização em nome do Cliente.

(FR31a). Web Browser e Remote Desktop Canvas funcionam como meio de saída do sistema para Client User.

B.6 A LISTA DE *REQUIREMENTS*

Requisitos: características desejáveis do sistema.

(R01). Cliente abre conexão com Servidor depois de receber do Usuário Cliente o IP e porta que identificam Servidor na rede.

(R02). Cliente e Servidor decidem configurações básicas do protocolo depois de conectados.

(R03). Servidor não aceita outras conexões enquanto estiver conectado a um Cliente.

(R04). Servidor envia bloco de atualização após uma mensagem de atualização do Cliente.

(R05). Cliente envia uma mensagem de atualização a cada evento de Mouse.

(R06). Cliente envia uma mensagem de atualização a cada evento de Teclado.

(R07). Cliente envia uma mensagem de atualização a cada evento de Timer.

(R09).O Servidor, depois de configurado, disponibiliza conexões de Servidor Web e Comunicação Bidirecional.

(R10).A configuração do Servidor é o primeiro passo e consiste na escolha de uma porta para acesso da aplicação, uso de modo depuração para realização da conexão sem criptografia e a escolha de uso de certificado digital próprio ou padrão da aplicação para a configuração do Servidor Web e do canal de Comunicação Bidirecional.

(R22). Servidor gera eventos de Mouse para o Sistema Operacional do Servidor depois de receber a mensagem de atualização de Mouse do Cliente.

(R23). Servidor gera eventos de Teclado para o Sistema Operacional do Servidor depois de receber a mensagem de atualização de Teclado do Cliente.

(R24).O Servidor captura imagens da Área de Trabalho do Servidor depois de gerados os eventos de periféricos no Sistema Operacional.

(R25).O Servidor guarda o estado anterior da Área de Trabalho do Servidor depois de adquirir uma nova imagem da Área de Trabalho.

(R26).A nova Área de Trabalho do Servidor é comparada com a antiga e apenas as regiões modificadas são codificadas na mensagem de bloco de atualização enviada para o Cliente.

(R31).O Cliente, ao receber a mensagem de atualização do Servidor, interpreta a mensagem e monstra o resultado da atualização na tela para o Usuário Cliente.

(R32).O Cliente, depois de inicializado, quando acometido por algum tipo de exceção ou sinal de encerramento, mostra uma mensagem correspondente para o Usuário Cliente e encerra a conexão com o servidor.

(R33).O servidor, depois de inicializado, quando acometido por algum tipo de exceção ou sinal de encerramento, encerra a conexão com o Cliente.

B.6.1 REGISTRO DE ALTERAÇÕES DA LISTA

(R08).(REMOVIDO, motivo: requisito não funcional) Comunicação no módulo de comunicação bidirecional ocorre a base de mensagens.

(R12).(INCORPORADO ao requisito 10) Modo Debug do Servidor permite conexão sem criptografia.

(R13).(INCORPORADO ao requisito 9) Servidor disponibiliza Servidor Web.

(R14).(INCORPORADO ao requisito 9) Servidor disponibiliza módulo de comunicação bidirecional.

(R15).(REMOVIDO, motivo: não é requisito, faz parte das escolhas tecnológicas) WebSocket troca mensagens entre Cliente e Servidor.

(R16).(REMOVIDO, motivo: R01 e R09 cumprem requisito) Depois de configurado o Servidor, o Servidor Web é inicializado e disponibiliza páginas web e scripts para os clientes que requisitarem conexão.

(R17).(INCORPORADO ao requisito 10) Depois de configurado o modo de funcionamento do Servidor, o Servidor Web e o canal de Comunicação Bidirecional passam a utilizar o certificado configurado.

(R18).(INCORPORADO ao requisito 17) Usuário pode utilizar um certificado digital próprio.

(R19).(INCORPORADO ao requisito 5) Cliente detecta eventos de mouse.

(R20).(INCORPORADO ao requisito 6) Cliente detecta eventos de teclado.

(R21).(REMOVIDO, motivo: não é requisito, faz parte das escolhas tecnológicas) Protocolo é baseado em extensão do *RFB*.

(R27).(REMOVIDO, motivo: requisito não funcional) Servidor divide a Área de Trabalho em 9 retângulos de mesmo tamanho no formato 3 x 3.

(R28).(REMOVIDO) Servidor envia num bloco de atualização apenas os retângulos com alteração superior a XYZ

(R29).(REMOVIDO) Servidor enfileira blocos de atualização.

(R30).(DESNECESSÁRIO) Fila de blocos de atualização guarda apenas 1 bloco.

APÊNDICE C – ADIT – A2

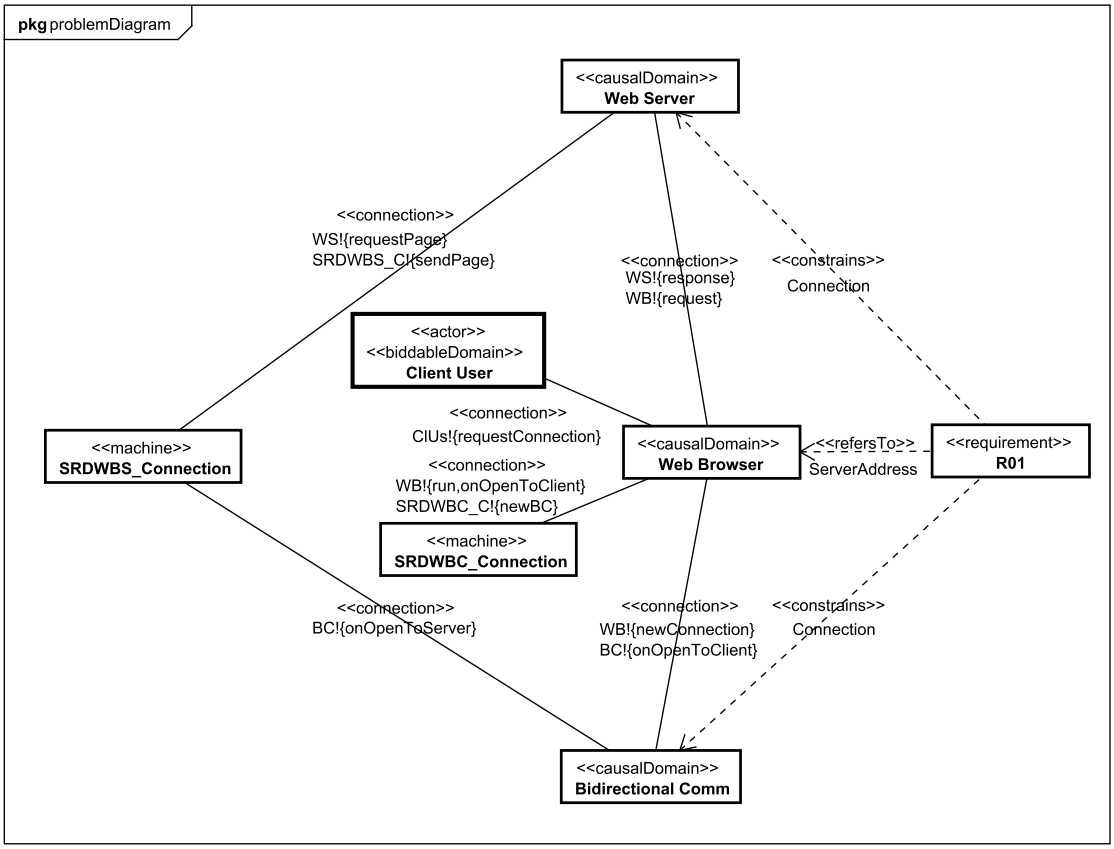

Figura 15: Diagrama de Subproblema para o requisito R01, levantado na fase A2.

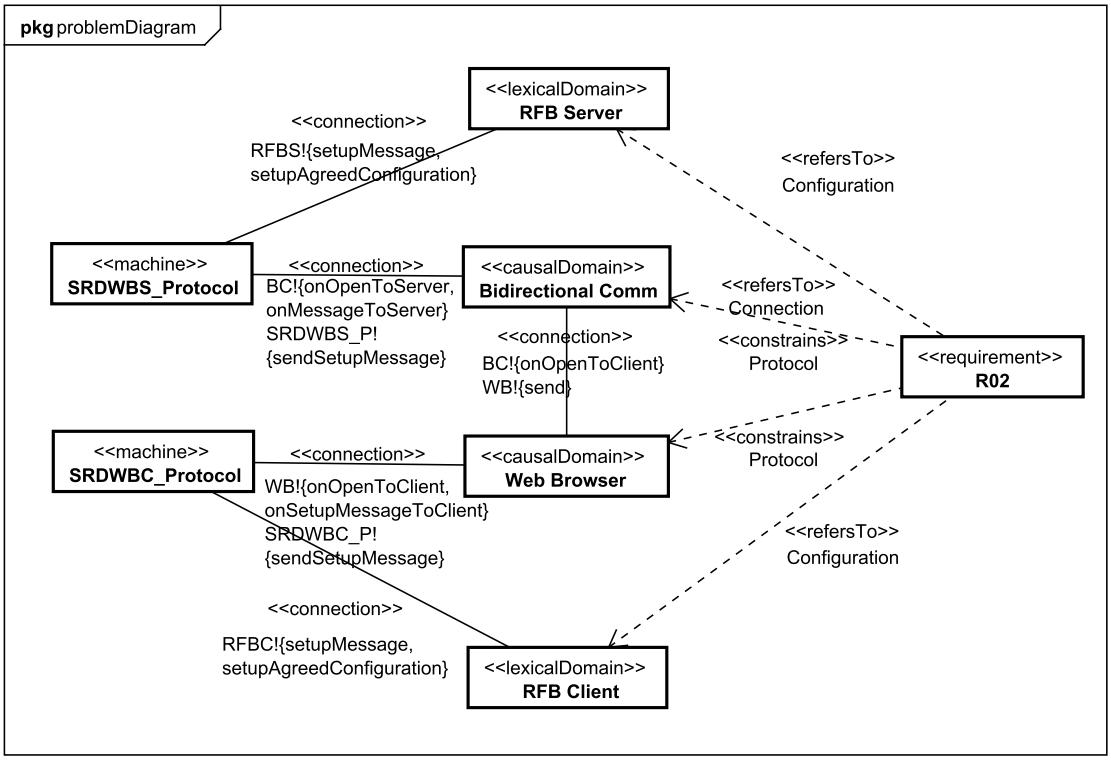

Figura 16: Diagrama de Subproblema para o requisito R02, levantado na fase A2.

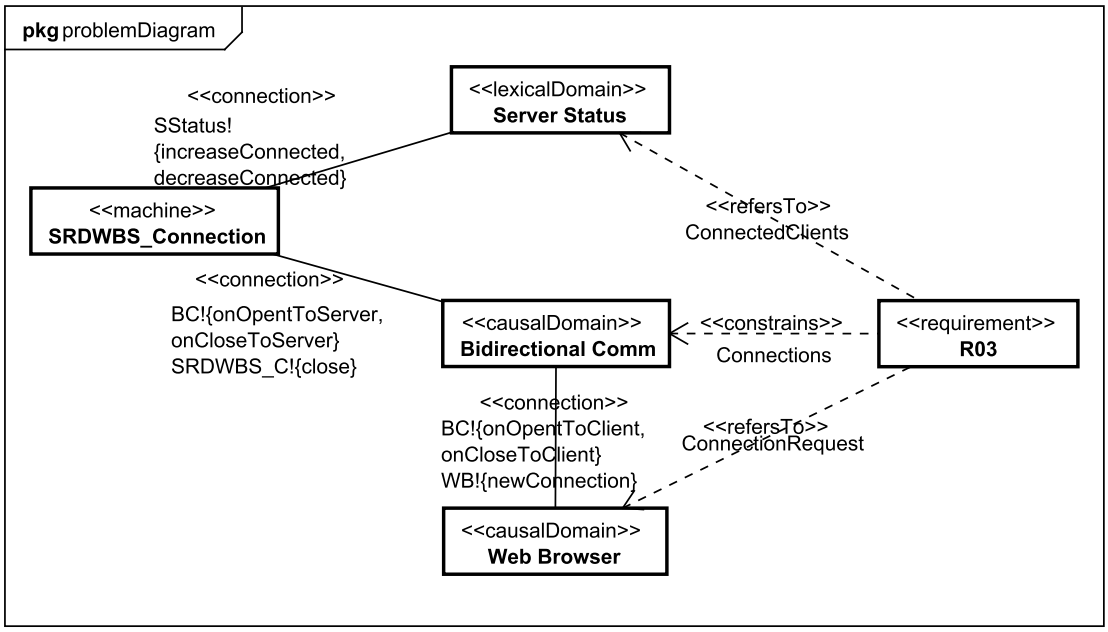

Figura 17: Diagrama de Subproblema para o requisito R03, levantado na fase A2.

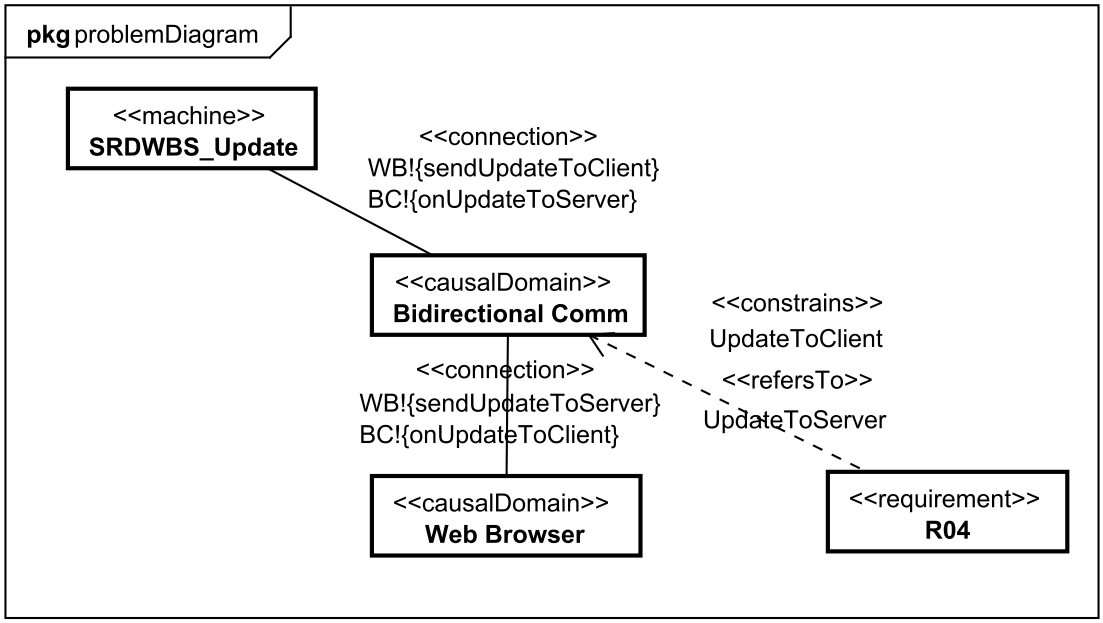

Figura 18: Diagrama de Subproblema para o requisito R04, levantado na fase A2.

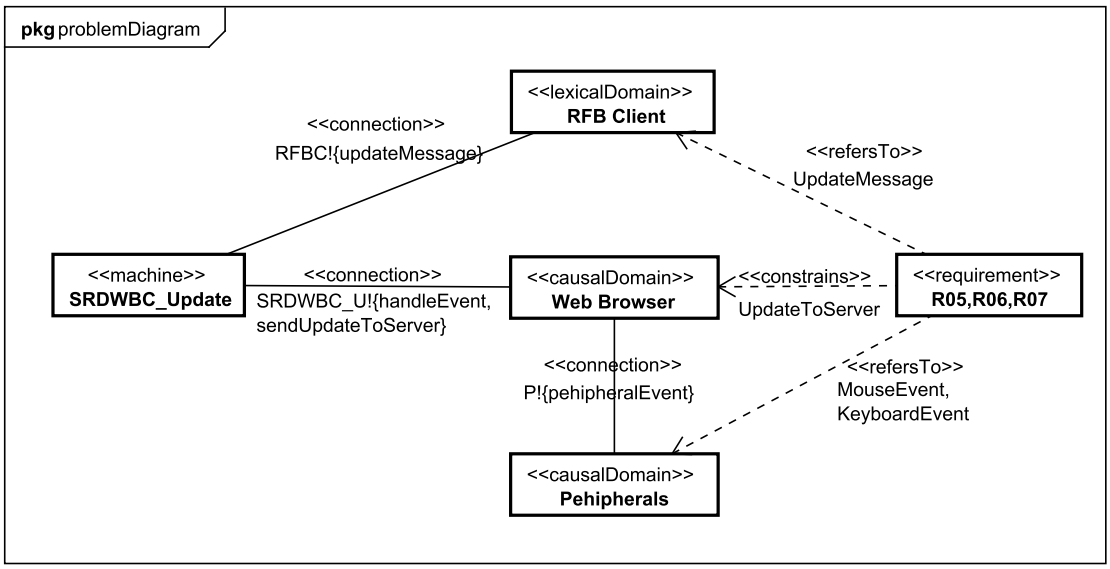

Figura 19: Diagrama de Subproblema para os requisitos R05, R06, R07, levantado na fase A2.

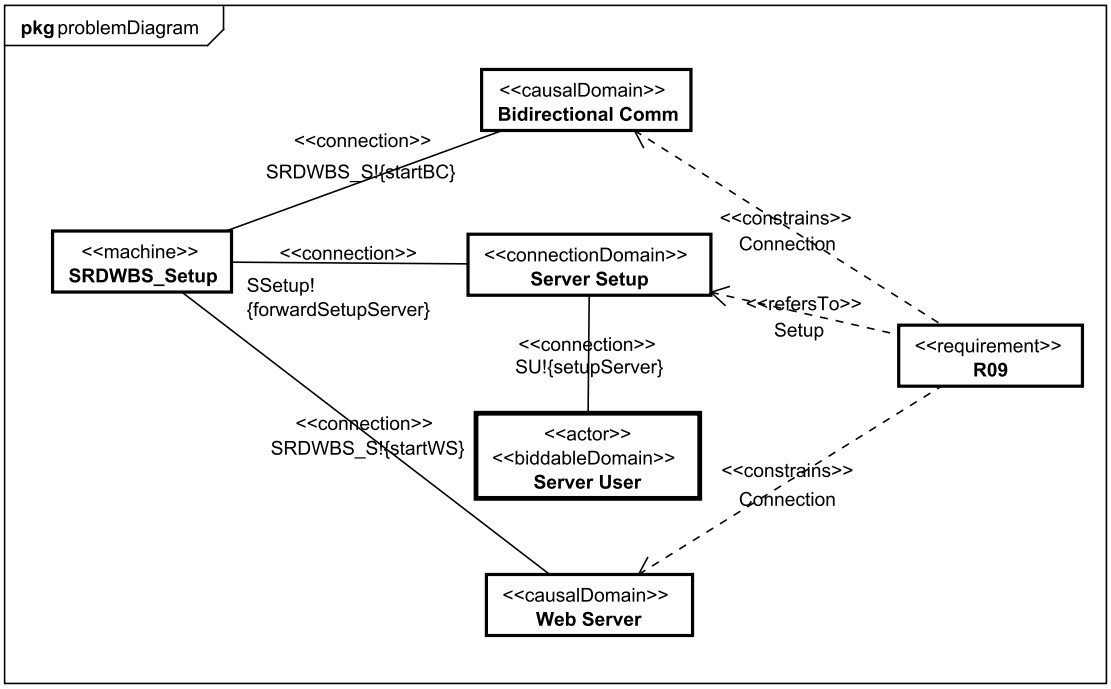

Figura 20: Diagrama de Subproblema para o requisito R09, levantado na fase A2.

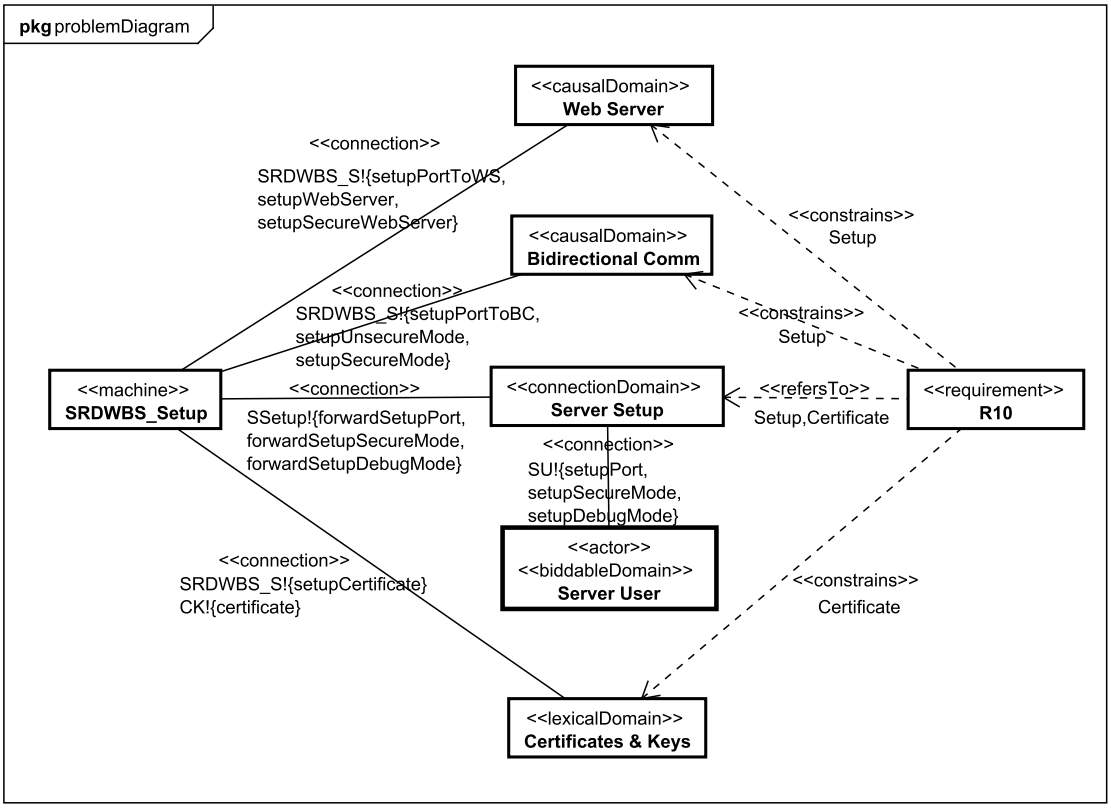

Figura 21: Diagrama de Subproblema para o requisito R10, levantado na fase A2.

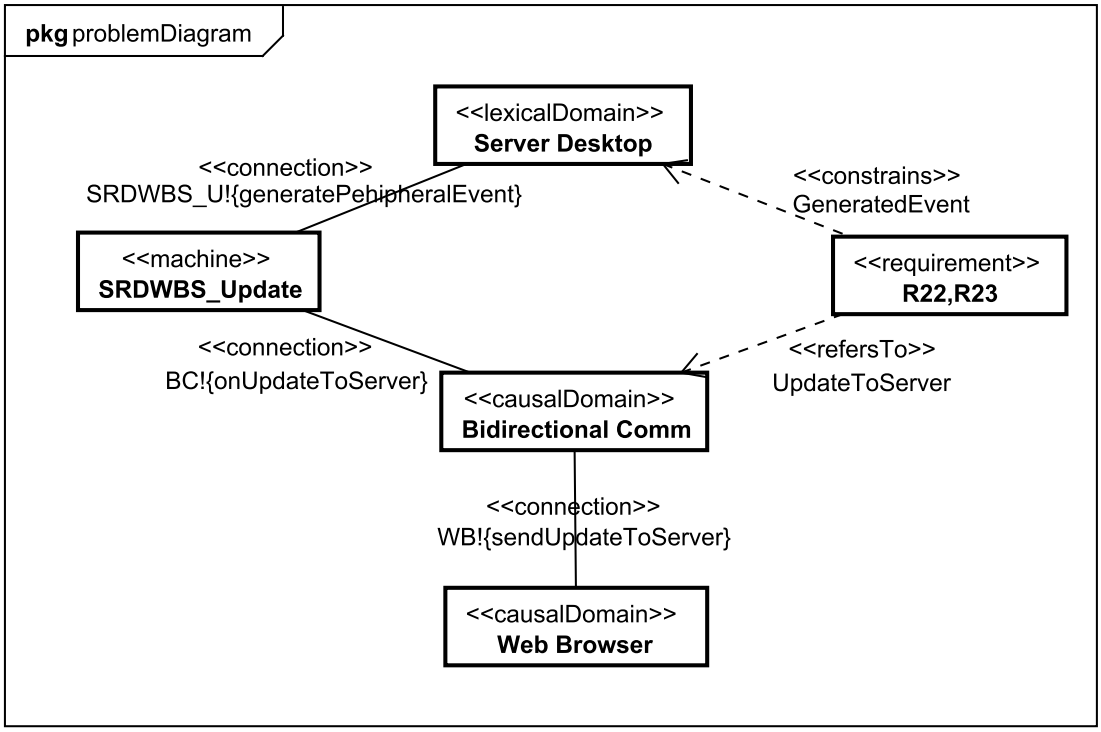

Figura 22: Diagrama de Subproblema para os requisitos R22, R23, levantado na fase A2.

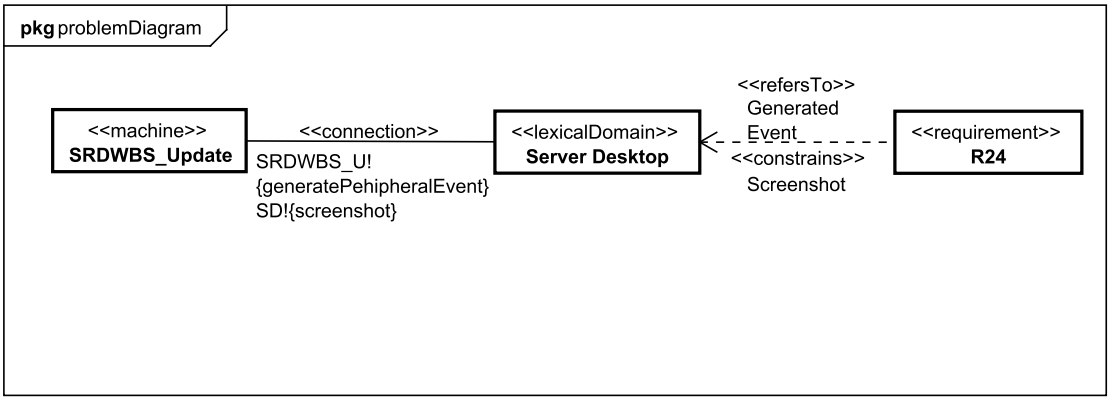

Figura 23: Diagrama de Subproblema para o requisito R24, levantado na fase A2.

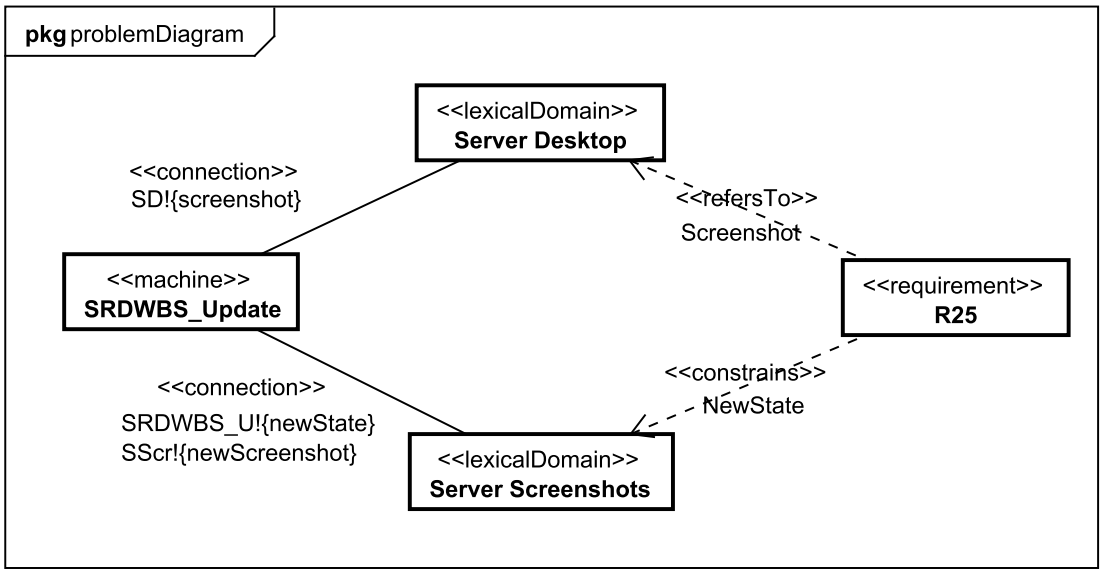

Figura 24: Diagrama de Subproblema para o requisito R25, levantado na fase A2.

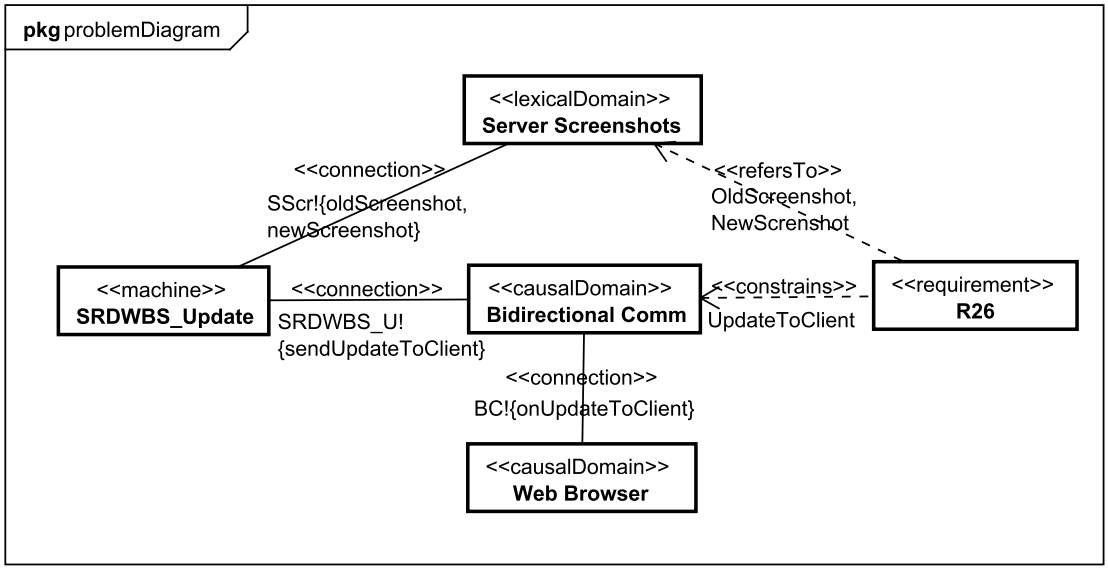

Figura 25: Diagrama de Subproblema para o requisito R26, levantado na fase A2.

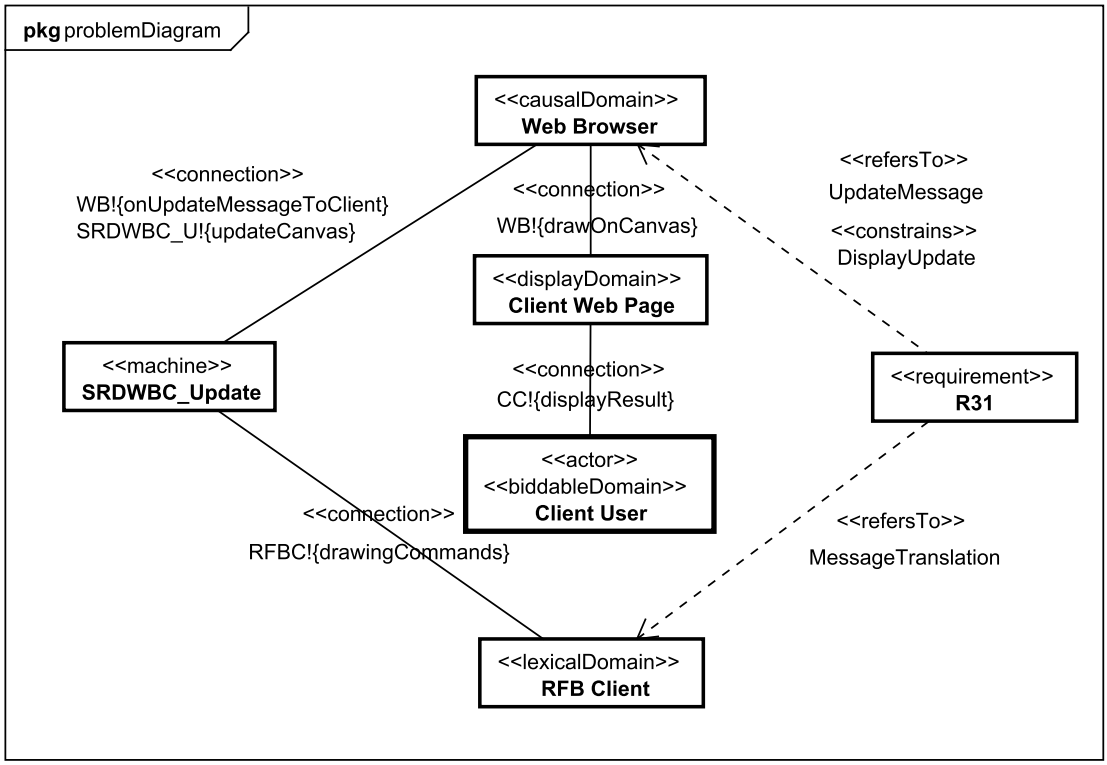

Figura 26: Diagrama de Subproblema para o requisito R31, levantado na fase A2.

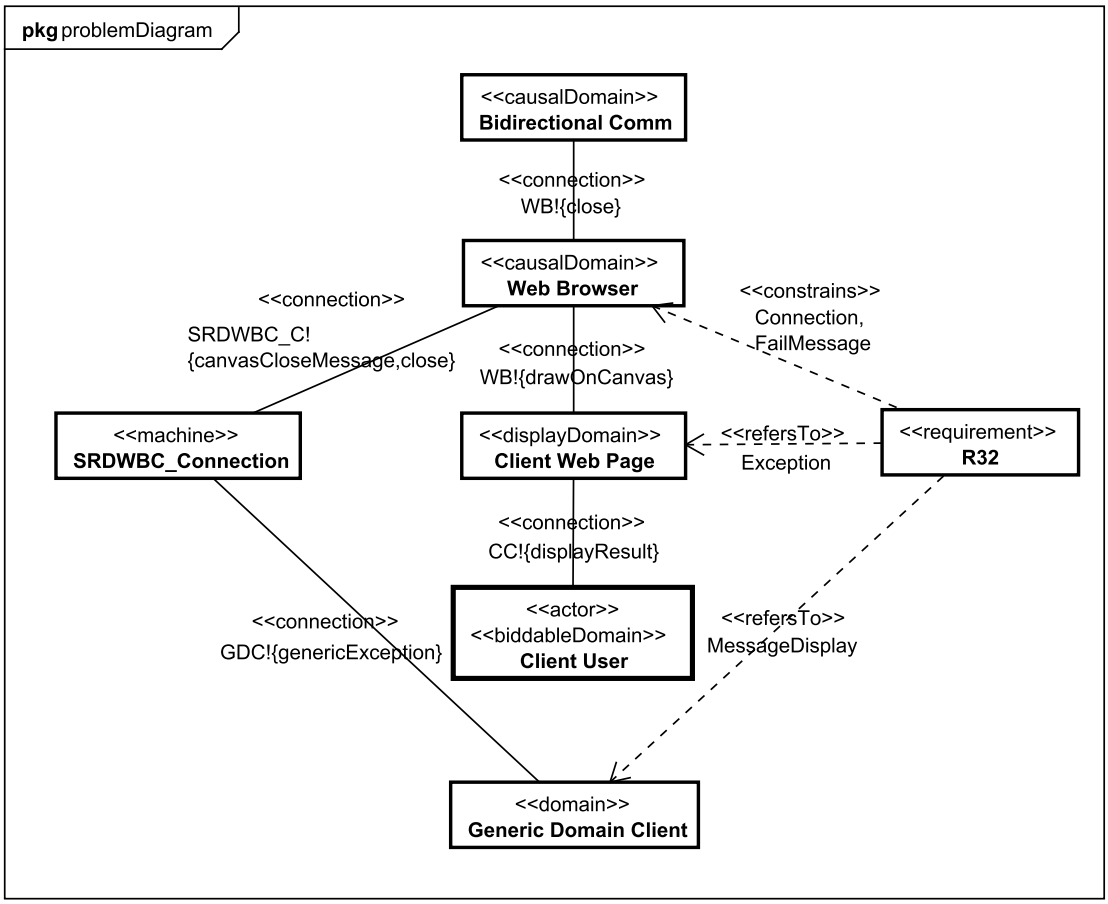

Figura 27: Diagrama de Subproblema para o requisito R32, levantado na fase A2.

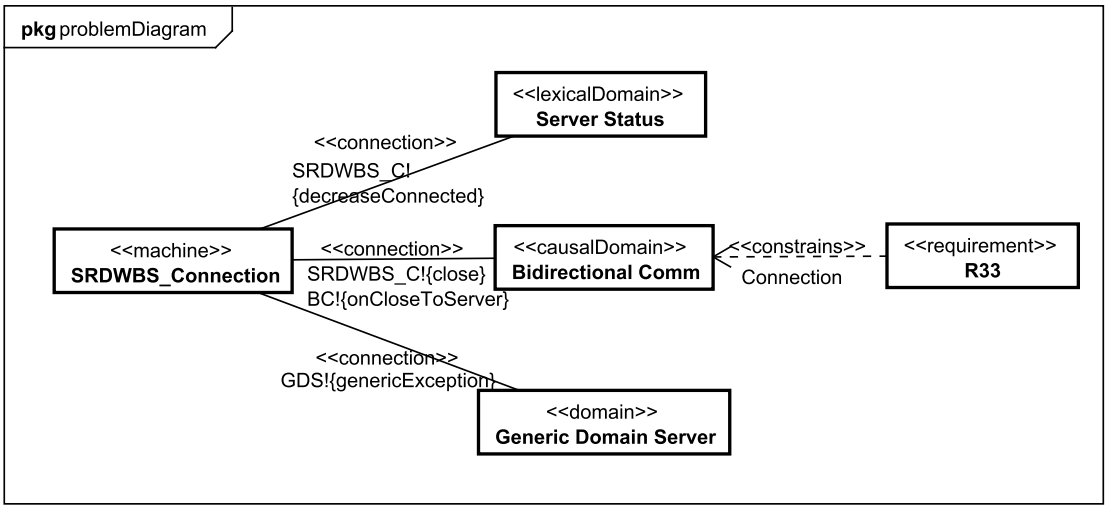

Figura 28: Diagrama de Subproblema para o requisito R33, levantado na fase A2.

APÊNDICE D – ADIT – A3

D.1 ESPECIFICAÇÕES DERIVADAS

Tabela 12: Definição das especificações para o requisito R01.

Requisito	**(R01).**Cliente abre conexão com Servidor depois de receber do Usuário Cliente o IP e porta que identificam Servidor na rede.
Suposições e Fatos	**(AR01a).**Client User conhece o endereço que identifica o Servidor na rede. **(FR01a).**Web Browser e Web Server executam um protocolo que permite se encontrarem, abrirem conexão e trocar dados, desde que haja conectividade e endereço válido que identifiquem o Servidor na rede. **(AR01b).**Servidor e cliente estão conectados. **(FR01b).**Web Browser recebe os dados de "newBC" e executa "newConnection" na tentativa de abrir uma conexão com Bidirectional Comm. **(FR01c).**Web Browser e Bidirectional Comm executam um protocolo que permite se encontrarem, abrirem conexão e trocar dados, desde que haja conectividade e endereço válido que identifiquem o Servidor na rede.
Especificações	**(S01a).**Ao receber "requestWebPage", "machine" envia por "sendWebPage" a página obtida por "webPage". **(S01b).**Ao receber "run", "machine" abre a conexão bidirecional executando "newBC".

Tabela 13: Definição das especificações para o requisito R02.

Requisito	**(R02).**Cliente e Servidor decidem configurações básicas do protocolo depois de conectados.
Suposições e Fatos	**(FR02a).**Bidirectional Comm e Web Browser medeiam comunicação entre SRDWBS_Protocol e SRDWBC_Protocol.
Especificações	**(S02a).**Ao receber "onOpenToServer", "machine" obtém a mensagem de configuração do protocolo por "setupMessage" e a envia por "sendSetupMessage". **(S02b).**Ao receber "onSetupMessageToClient", "machine" deixa a interpretação da mensagem para RFB Client através de "setupMessage", que retorna a mensagem . Em seguida ocorre um "sendSetupMessage". **(S02c).**Ao receber "onSetupMessageToServer", "machine" configura e recebe a mensagem de confirmação por "setupAgreedConfiguration". Ocorre então "sendAgreedConfigurationMessage". **(S02d).**Ao receber "onAgreedConfigurationMessage", "machine" repassa a mensagem via "setupAgreedConfiguration".

Tabela 14: Definição das especificações para o requisito R03.

Requisito	**(R03).**Servidor não aceita outras conexões enquanto estiver conectado a um Cliente.
Suposições e Fatos	**(FR03a).**Todo "newConnection" bem sucedido ao Bidirectional Comm é seguido pela execução de "onOpenToClient" e "onOpenToServer". **(FR03b).**Server Status guarda as chamadas feitas por "increaseConnected" e "decreaseConnected". **(FR03c).**Caso a diferença de ocorrências de "increaseConnected" por "decreaseConnected" seja igual ou superior a 2 após a chamada de um "increaseConnected", um "denyConnection" ocorre, caso contrário um "grantConnection" ocorre. **(FR03d).**Bidirectional Comm e Web Browser medeiam a comunicação entre Cliente e Servidor.
Especificações	**(S03a).**Toda vez que "machine" receber um "onOpenToServer", um "increaseConnected" ocorre em seguida. Ao receber um "denyConnection" como resposta, "machine" causa um "close". Caso um "grantConnection" seja recebido a comunicação continua. **(S03b).**Toda vez que "machine" perceber um "onCloseToServer" um "decreaseConnected" deve ocorrer.

Tabela 15: Definição das especificações para o requisito R04.

Requisito	**(R04).**Servidor envia bloco de atualização após uma mensagem de atualização do Cliente.
Suposições e Fatos	**(FR03d).**Bidirectional Comm e Web Browser medeiam a comunicação entre Cliente e Servidor. **(FR04a).**Web Browser emite as mensagens de atualização do cliente via "sendUpdateToServer".
Especificações	**(S04).**Ao receber uma mensagem de atualização por "onUpdateToServer" o bloco de atualização é enviado por "sendUpdateToClient".

Tabela 16: Definição das especificações para o requisito R05, 06, 07.

Requisito	**(R05).**Cliente envia uma mensagem de atualização a cada evento de Mouse. **(R06).**Cliente envia uma mensagem de atualização a cada evento de Teclado. **(R07).**Cliente envia uma mensagem de atualização a cada evento de Timer.
Suposições e Fatos	**(FR05,06,07a).**O Web Browser é responsável por inspecionar o comportamento de Timer, Pehipherals e os traduz para "handleEvent".
Especificações	**(S05,06,07).**Ao recebimento de um "handleEvent", "machine" executa "sentUpdateToServer".

Tabela 17: Definição das especificações para o requisito R09.

Requisito	**(R09).**O Servidor, depois de configurado, disponibiliza conexões de Servidor Web e Comunicação Bidirecional.
Suposições e Fatos	**(FR09a).**Server Setup medeia a comunicação entre Server User e "machine".
Especificações	**(S09).**Depois de concluído "forwardSetupServer", "machine" emite "startBC" e "startWS".

Tabela 18: Definição das especificações para o requisito R10.

Requisito	**(R10).**A configuração do Servidor é o primeiro passo e consiste na escolha de uma porta para acesso da aplicação, uso de modo depuração para realização da conexão sem criptografia e a escolha de uso de certificado digital próprio ou padrão da aplicação para a configuração do Servidor Web e do canal de Comunicação Bidirecional.
Suposições e Fatos	**(FR09a).**Server Setup medeia a comunicação entre Server User e "machine".
Especificações	**(S10a).**Ao receber "forwardSetupPort", "machine" identifica as portas e informa a configuração via "setupBCPort" e "setupWSPort". **(S10b).**Ao receber "forwardSetupDebugMode" para configurar a comunicação sem criptografia, "machine" emite um "setupUnsecureMode". **(S10c).**(INCORPORADO a especificação 10d) Ao receber "forwardSetupSecureMode", "machine" configura o meio de segurança com a emissão de "setupCertificate". **(S10d).**Depois de um "forwardSetupSecureMode" bem sucedido, "machine" executa "setupSecureConnection" e "setupCertificate" com os dados obtidos em "certificate".

Tabela 19: Definição das especificações para o requisito R22, 23.

Requisito	**(R22).**Servidor gera eventos de Mouse para o Sistema Operacional do Servidor depois de receber a mensagem de atualização de Mouse do Cliente. **(R23).**Servidor gera eventos de Teclado para o Sistema Operacional do Servidor depois de receber a mensagem de atualização de Teclado do Cliente.
Suposições e Fatos	**(FR03d).**Bidirectional Comm e Web Browser medeiam a comunicação entre Cliente e Servidor. **(FR22,23a).**O Web Browser envia mensagem de atualização em nome do Cliente.
Especificações	**(S22,23).**Depois de receber as mensagens de atualização de periféricos via "onUpdateToServer", "machine" os interpreta e gera os eventos simulados para o Sistema Operacional por "generatePehipheralEvent".

Tabela 20: Definição das especificações para o requisito R24.

Requisito	**(R24).**O Servidor captura imagens da Área de Trabalho do Servidor depois de gerados os eventos de periféricos no Sistema Operacional.
Suposições e Fatos	**(AR24a).**Ao término de "generatePehipheralEvent" os eventos já afetaram o Sistema Operacional.
Especificações	**(S24).**Ao termino de "generatePehipheralEvent","machine" obtém a captura de imagem da Área de Trabalho via "screenshot".

Tabela 21: Definição das especificações para o requisito R25.

Requisito	**(R25).**O Servidor guarda o estado anterior da Área de Trabalho do Servidor depois de adquirir uma nova imagem da Área de Trabalho.
Suposições e Fatos	
Especificações	**(S25).**As imagens armazenadas por Server Screenshot são atualizadas por "machine" ao passar a nova imagem, recebida por "screenshot", e a antiga imagem, recebida por "newScreenshot", pelo comando "newState".

Tabela 22: Definição das especificações para o requisito R26.

Requisito	**(R26).**A nova Área de Trabalho do Servidor é comparada com a antiga e apenas as regiões modificadas são codificadas na mensagem de bloco de atualização enviada para o Cliente.
Suposições e Fatos	
Especificações	**(S26).**As regiões modificadas são obtidas por "machine" ao comparar "oldScreenshot" com "newScreenshot". O resultado é codificado o resultado na mensagem de atualização para o Cliente enviada por "sendUpdateToClient".

Tabela 23: Definição das especificações para o requisito R31.

Requisito	**(R31).**O Cliente, ao receber a mensagem de atualização do Servidor, interpreta a mensagem e monstra o resultado da atualização na tela para o Usuário Cliente.
Suposições e Fatos	**(FR03d).**Bidirectional Comm e Web Browser medeiam a comunicação entre Cliente e Servidor. **(FR31a).**Web Browser e Remote Desktop Canvas funcionam como meio de saída do sistema para Client User.
Especificações	**(S31).**Ao receber um “onUpdateMessageToClient”, “machine” lista as alterações de tela através de “drawingCommands”. Com a listagem “updateCanvas” é executado.

Tabela 24: Definição das especificações para o requisito R32.

Requisito	**(R32).**O Cliente, depois de inicializado, quando acometido por algum tipo de exceção ou sinal de encerramento, mostra uma mensagem correspondente para o Usuário Cliente e encerra a conexão com o servidor.
Suposições e Fatos	**(FR03d).**Bidirectional Comm e Web Browser medeiam a comunicação entre Cliente e Servidor. **(FR31a).**Web Browser e Remote Desktop Canvas funcionam como meio de saída do sistema para Client User.
Especificações	**(S32).**Ao receber um “genericClientException”, “machine” envia os comandos “canvasCloseMessage” e “close”.

Tabela 25: Definição das especificações para o requisito R33.

Requisito	**(R33).**O servidor, depois de inicializado, quando acometido por algum tipo de exceção ou sinal de encerramento, encerra a conexão com o Cliente.
Suposições e Fatos	**(FR03d).**Bidirectional Comm e Web Browser medeiam a comunicação entre Cliente e Servidor.
Especificações	**(S33).**Ao receber um “genericServerException”, “machine” envia o comando “close”. **(S03b).**Toda vez que “machine” perceber um “onCloseToServer” um “decreaseConnected” deve ocorrer.

D.2 REGISTRO DE ALTERAÇÕES NA LISTA

Tabela 26: Registro de alteração das especificações para o requisito R16.

Tipo	REMOÇÃO
Descrição	**(R01) e (R09)**cumprem o papel do requisito (R16).
Requisito	**(R16).**Depois de configurado o Servidor, o Servidor Web é inicializado e disponibiliza páginas web e scripts para os clientes que requisitarem conexão.
Suposições e Fatos	**(FR09a).**Server Setup medeia a comunicação entre Server User e "machine". **(FR16a).**Um "requestPage" ocorre apenas quando há um "start" bem sucedido.
Especificações	**(S16a).**Depois de finalizado "forwardSetupServer", "machine" executa "start". **(S16b).**Um "requestPage" é atendido pela execução de "webPage", que utiliza o conteúdo recebido pelo comando "sendPage".

Tabela 27: Registro de alteração das especificações para o requisito R17.

Tipo	INCORPORAÇÃO
Descrição	**(R17)**INCORPORADO ao (R10). **(S17a)**INCORPORADO a (S10d).
Requisito	**(R17).**Depois de configurado o modo de funcionamento do Servidor, o Servidor Web e o canal de Comunicação Bidirecional passam a utilizar o certificado configurado.
Suposições e Fatos	**(FR09a).**Server Setup medeia a comunicação entre Server User e "machine".
Especificações	**(S17a).**Depois de um "forwardServerSetup" bem sucedido, "machine" executa "setupSecureConnection" e "setupCertificate" com os dados obtidos em "certificate".

D.3 DIAGRAMAS DE ESPECIFICAÇÃO

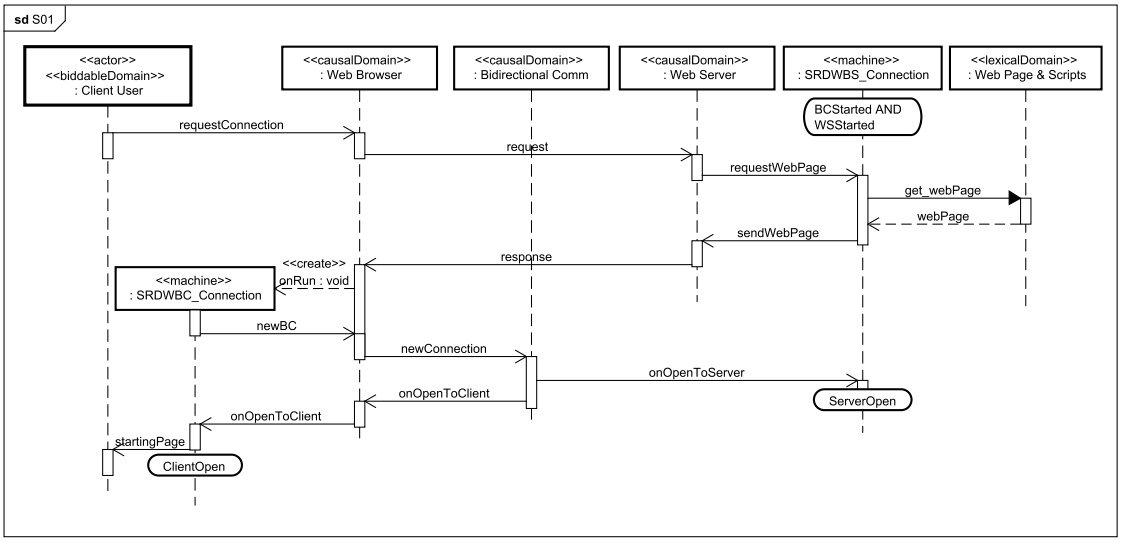

Figura 29: Diagrama de Especificação para a especificação S01, levantado na fase A3.

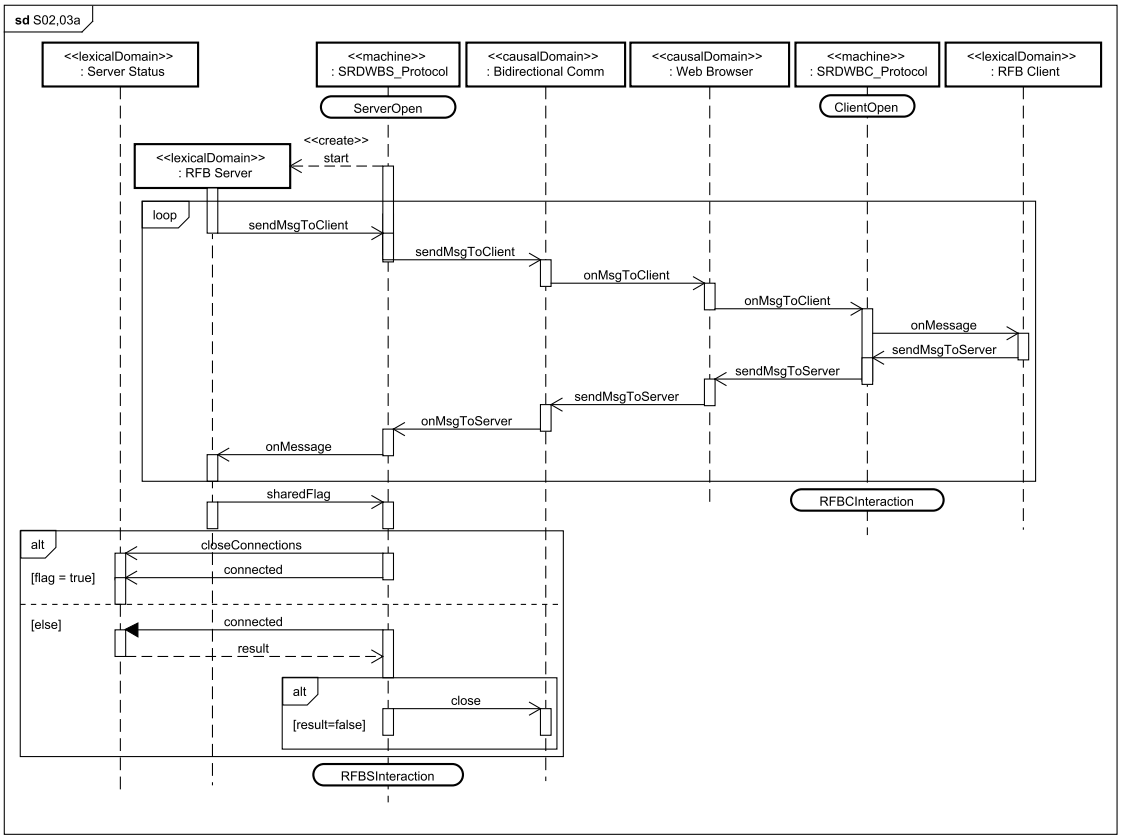

Figura 30: Diagrama de Especificação para as especificações S02, 03a, levantado na fase A3.

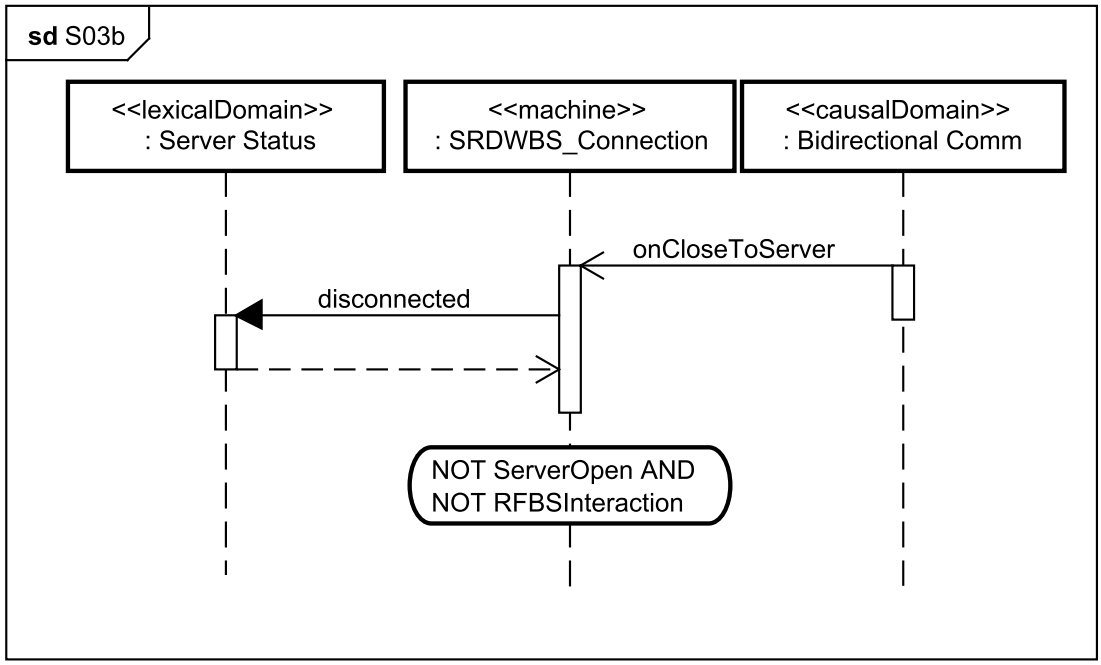

Figura 31: Diagrama de Especificação para a especificação S03b, levantado na fase A3.

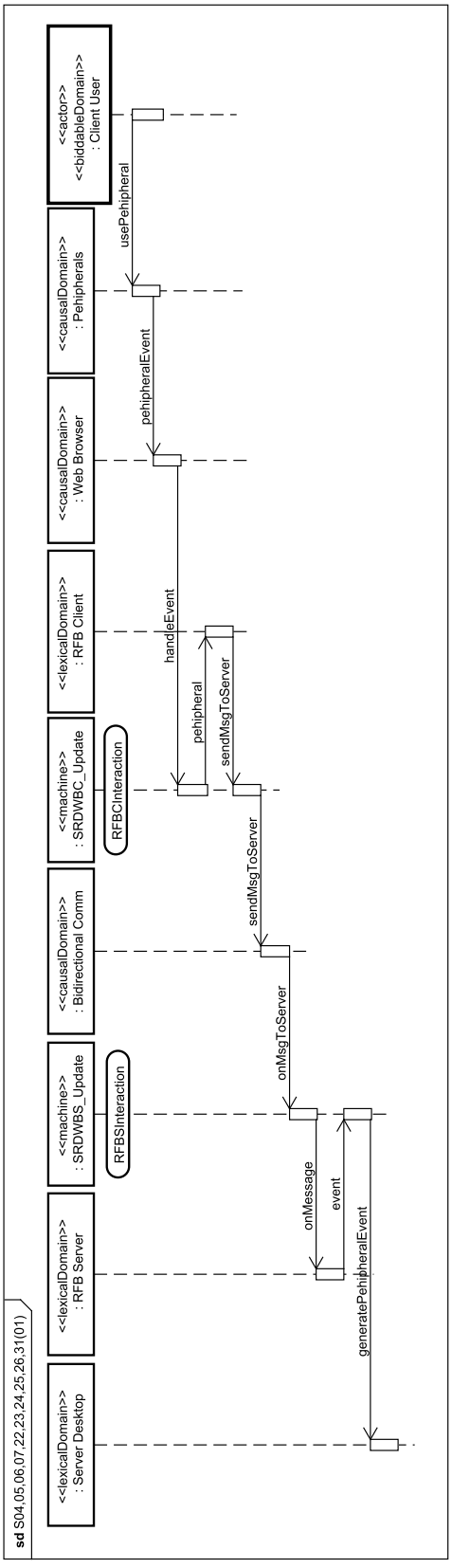

Figura 32: Diagrama de Especificação 01 para as especificações S04, 05, 06, 07, 22, 23, 24, 25, 26, 31, levantado na fase A3.

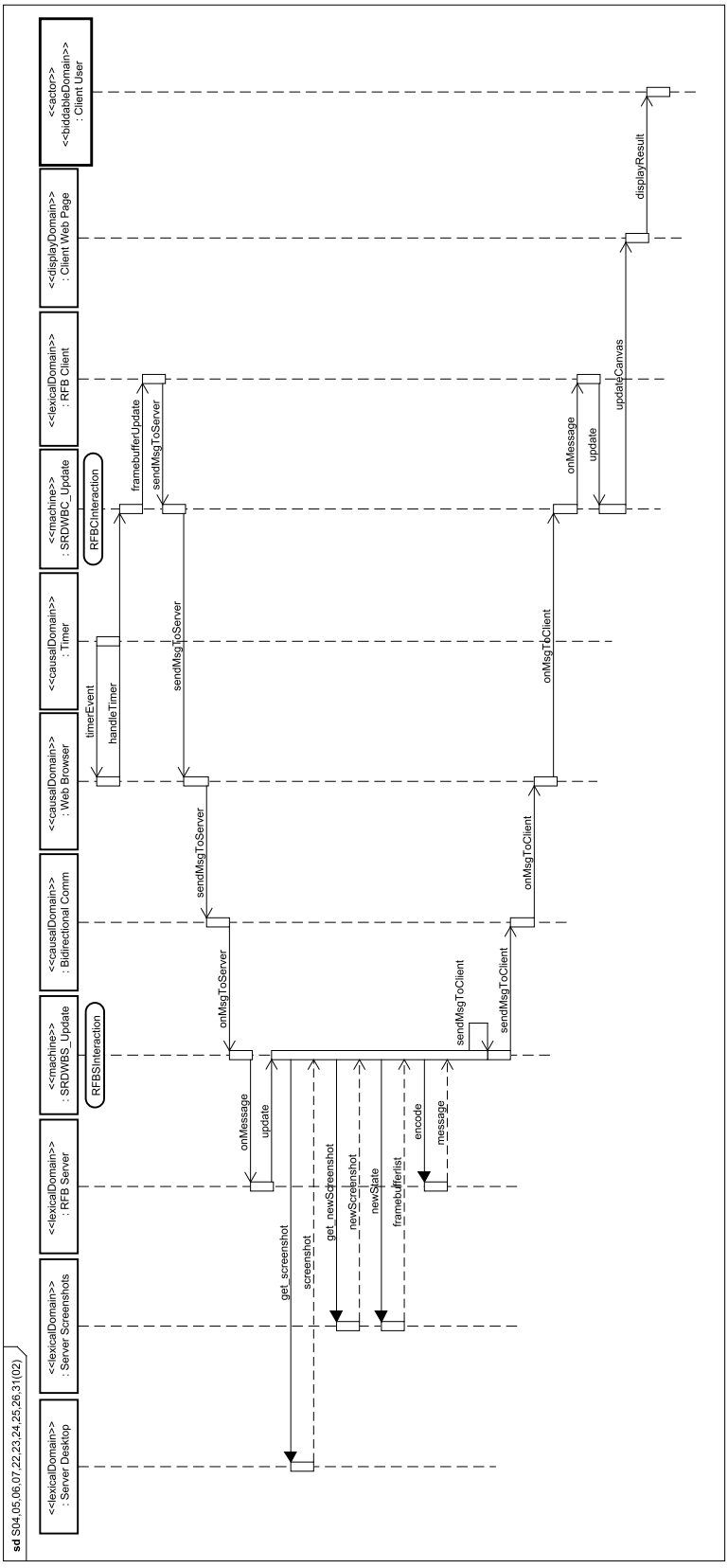

Figura 33: Diagrama de Especificação 02 para as especificações S04, 05, 06, 07, 22, 23, 24, 25, 26, 31, levantado na fase A3.

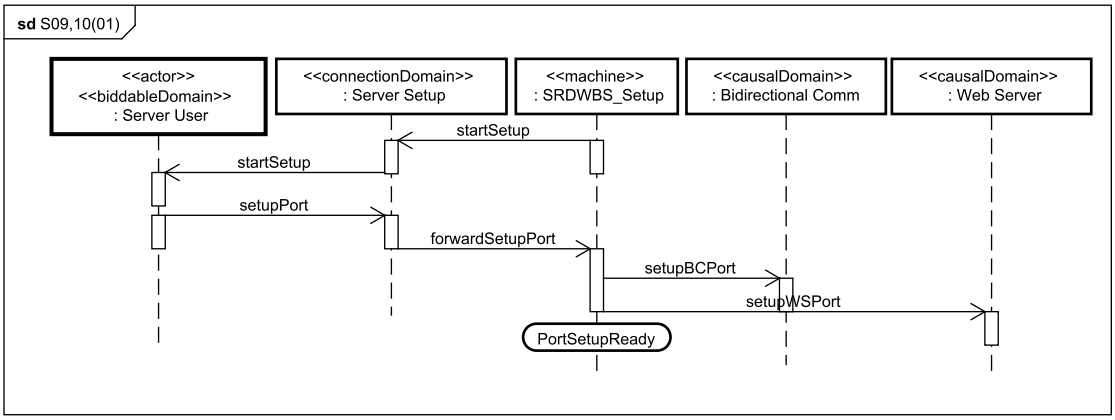

Figura 34: Diagrama de Especificação 01 para as especificações S09, 10, levantado na fase A3.

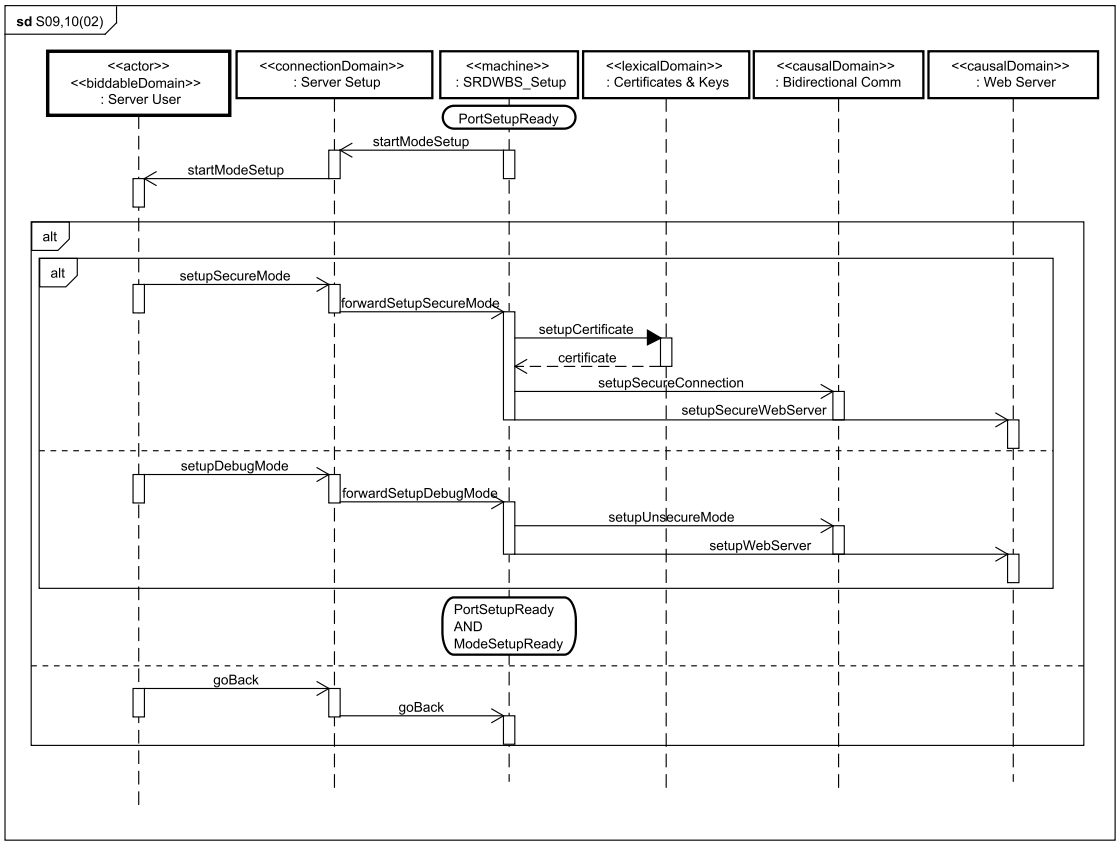

Figura 35: Diagrama de Especificação 02 para a especificações S09, 10, levantado na fase A3.

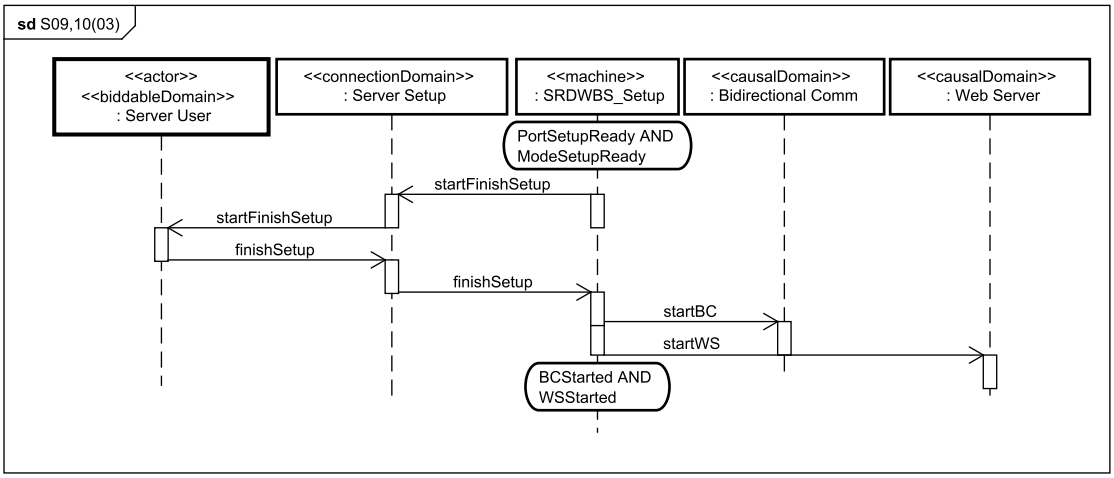

Figura 36: Diagrama de Especificação 03 para a especificações S09, 10, levantado na fase A3.

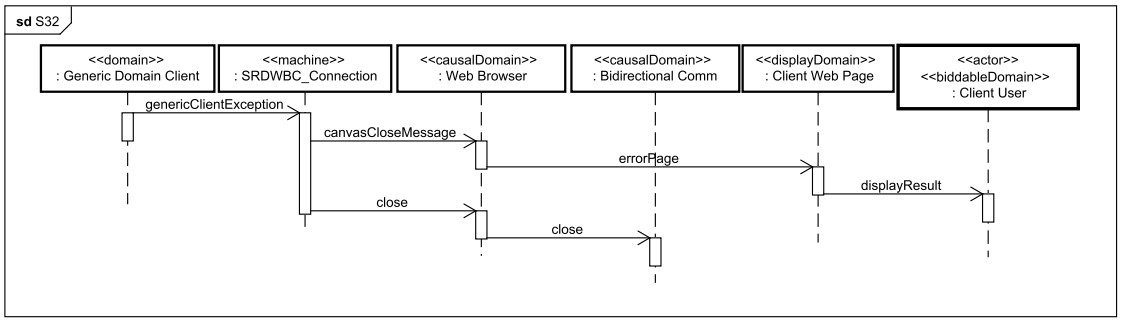

Figura 37: Diagrama de Especificação para a especificação S32, levantado na fase A3.

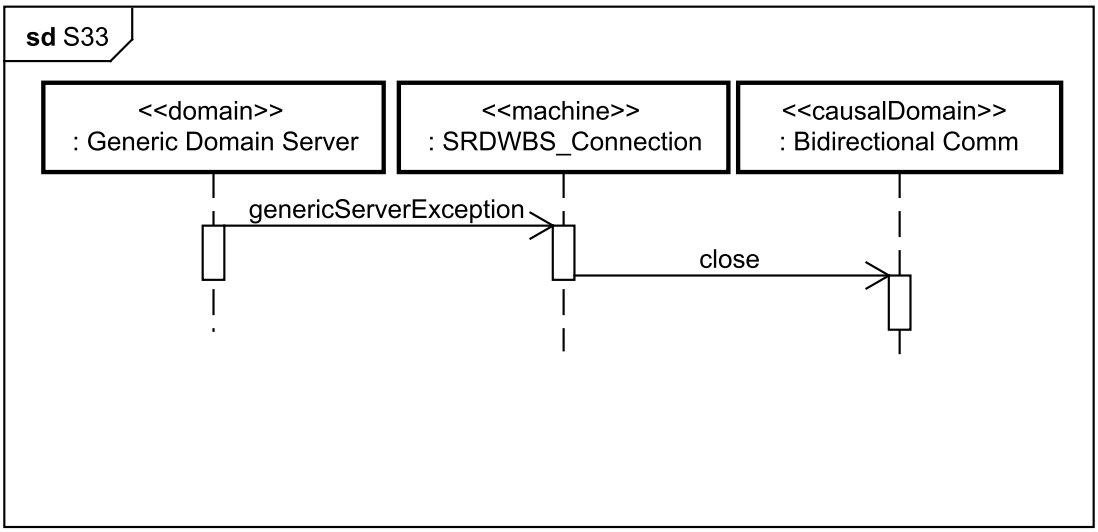

Figura 38: Diagrama de Especificação para a especificação S33, levantado na fase A3.

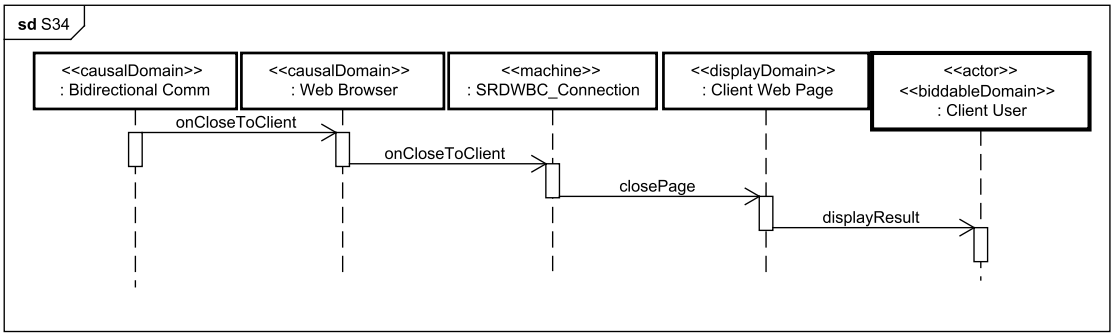

Figura 39: Diagrama de Especificação para a especificação S34, levantado na fase A3.

APÊNDICE E – ADIT – A4

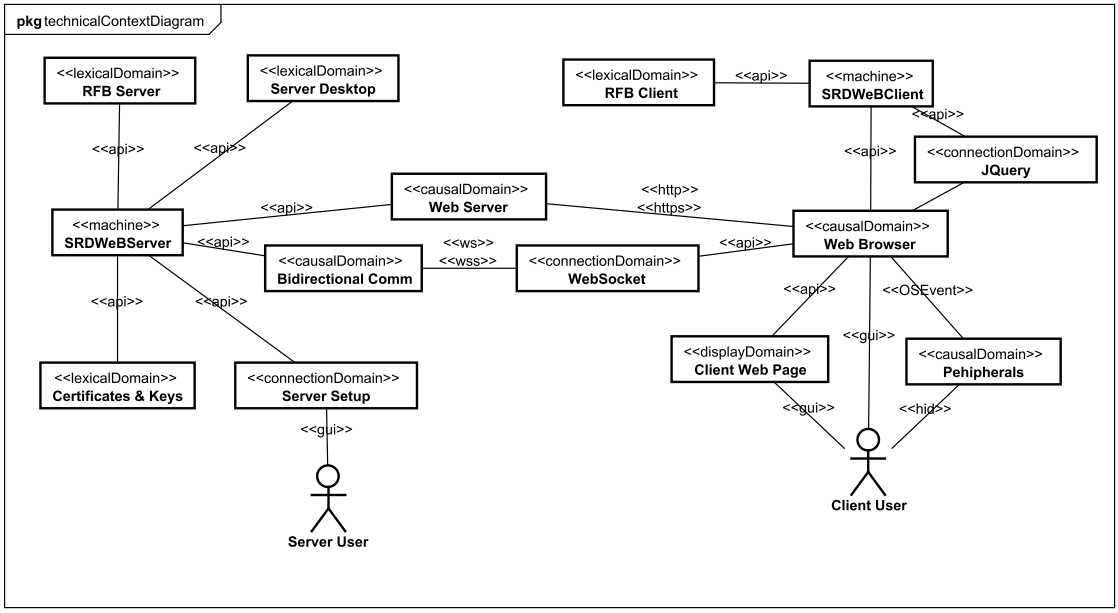

Figura 40: Diagrama de Contexto Técnico, levantado na fase A4.

APÊNDICE F – ADIT – A5

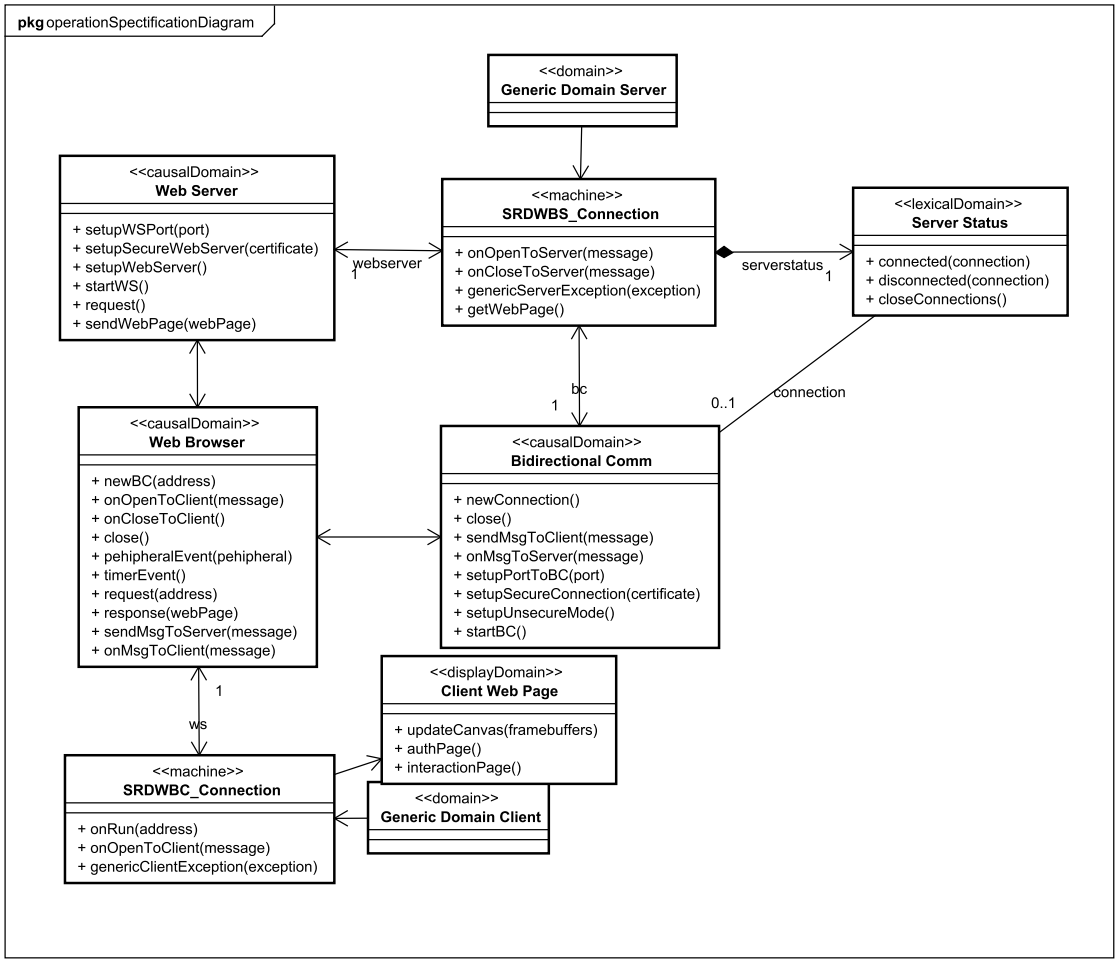

Figura 41: Diagrama de Especificação de Operações para a sequência “Connection”, levantado na fase A5.

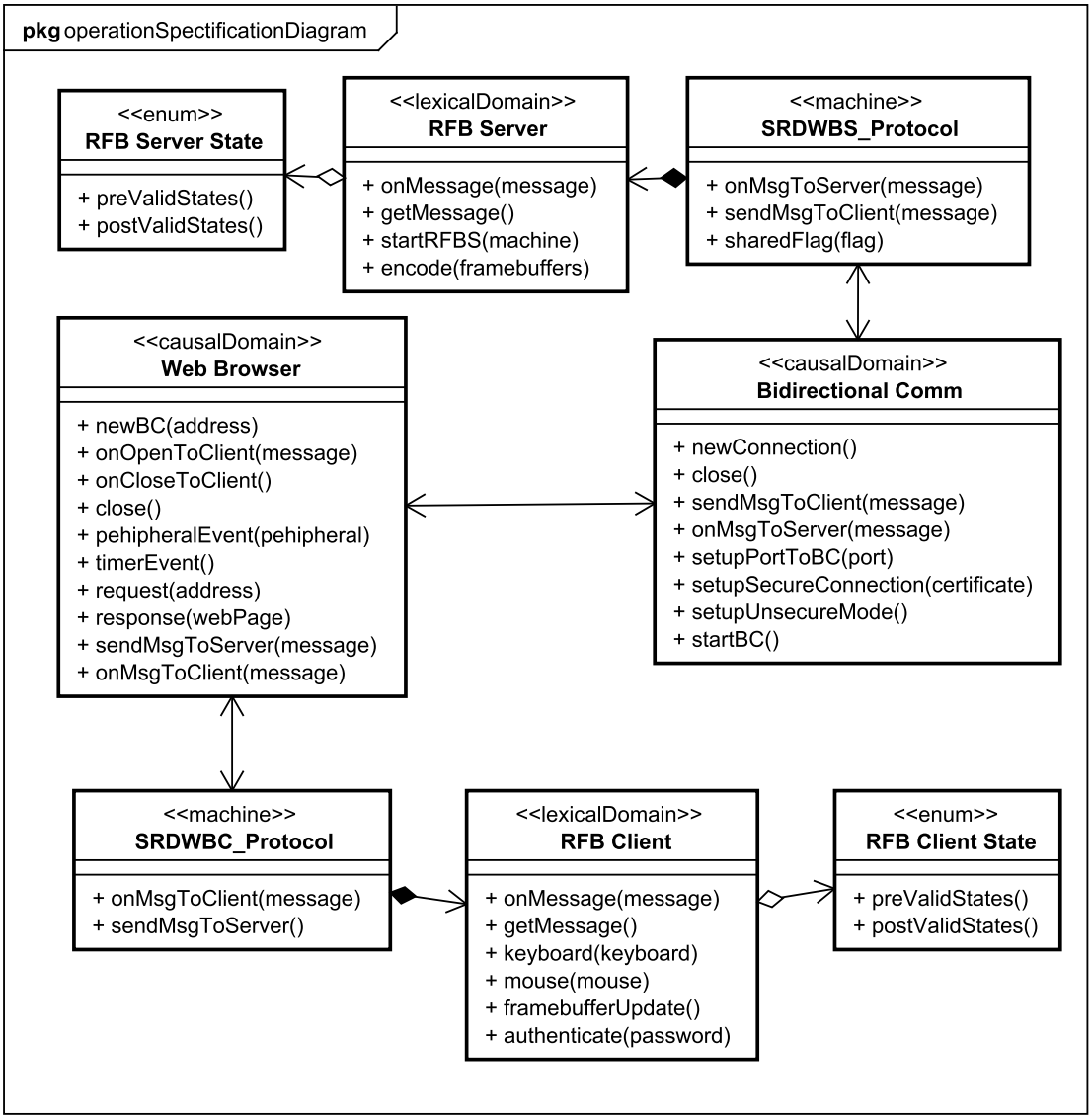

Figura 42: Diagrama de Especificação de Operações para a sequência “Protocol”, levantado na fase A5.

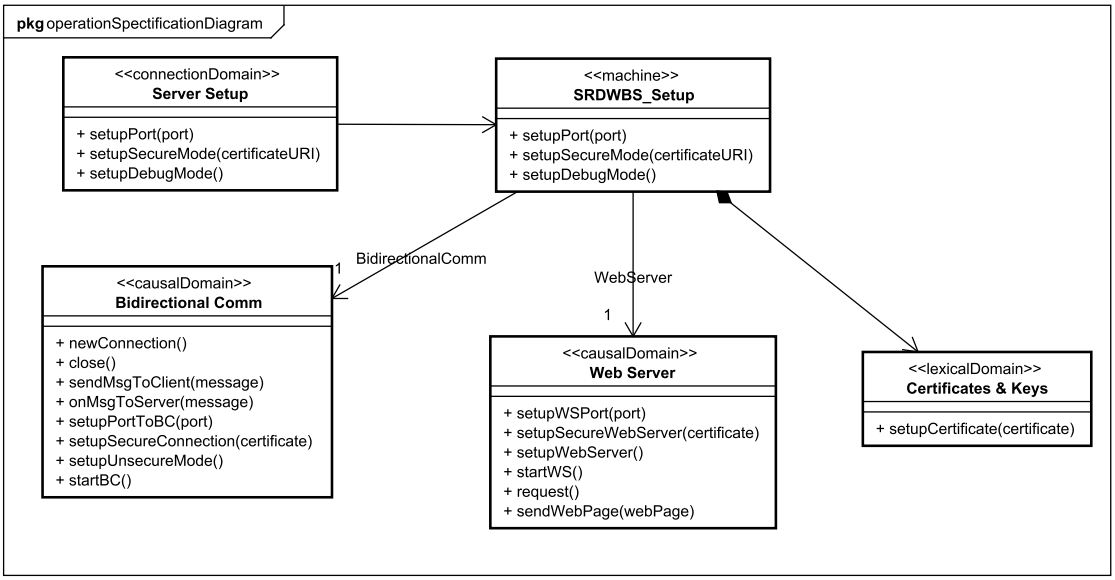

Figura 43: Diagrama de Especificação de Operações para a sequência “Setup”, levantado na fase A4.

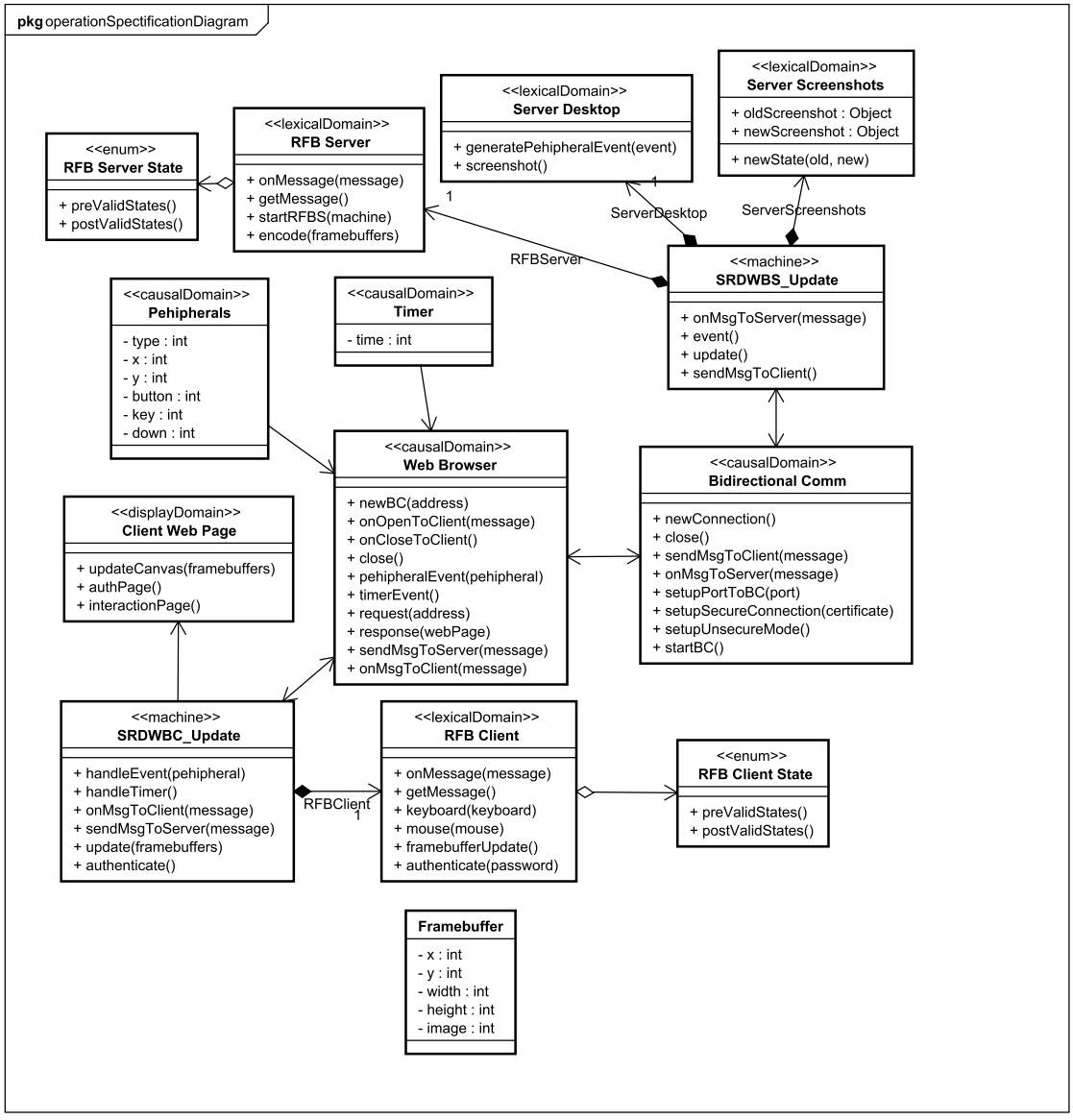

Figura 44: Diagrama de Especificação de Operações para a sequência “Update”, levantado na fase A4.

APÊNDICE G – ADIT – D1

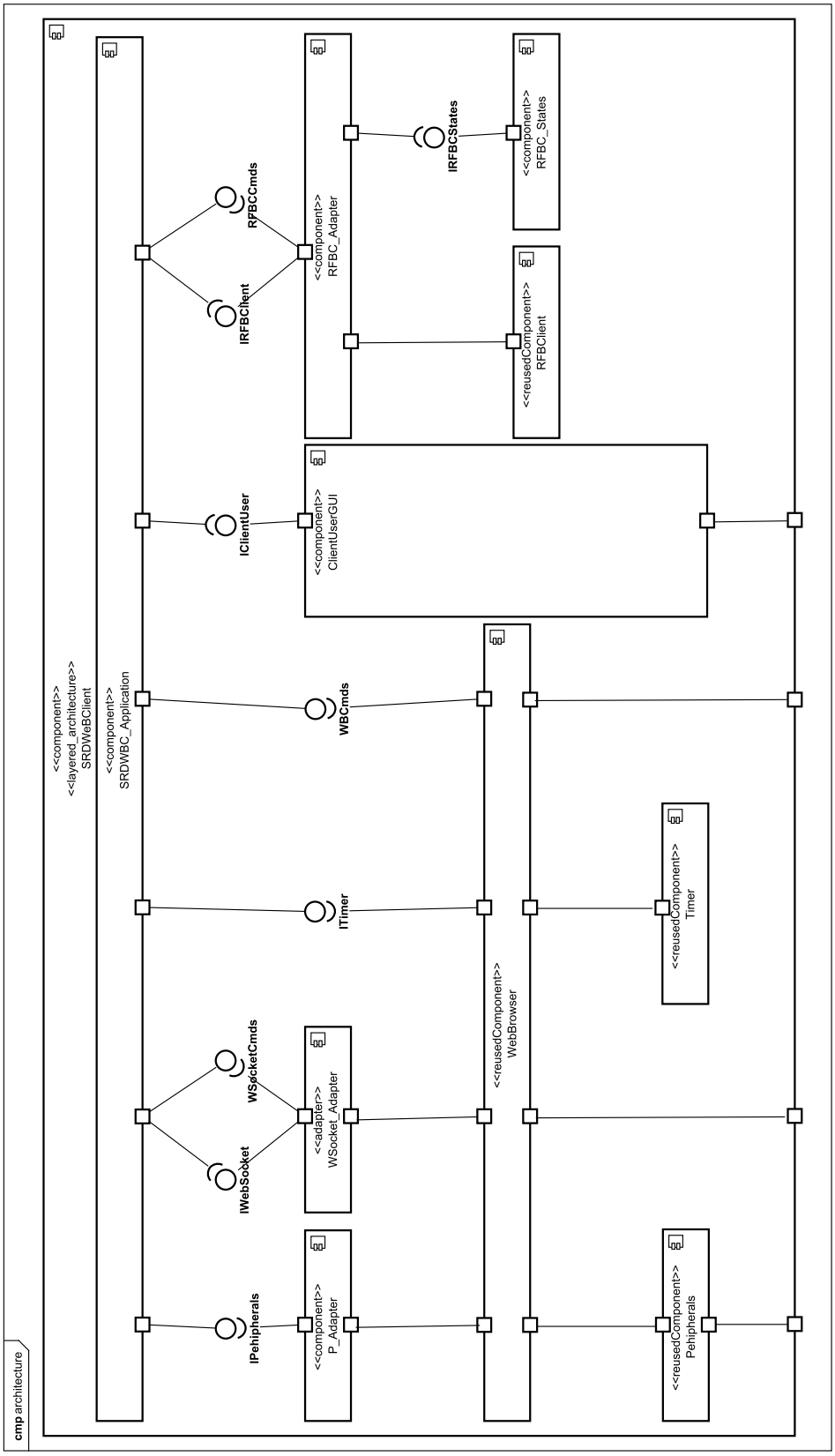

Figura 45: Diagrama de Arquitetura para o SRDWeBClient, levantado na fase D1.

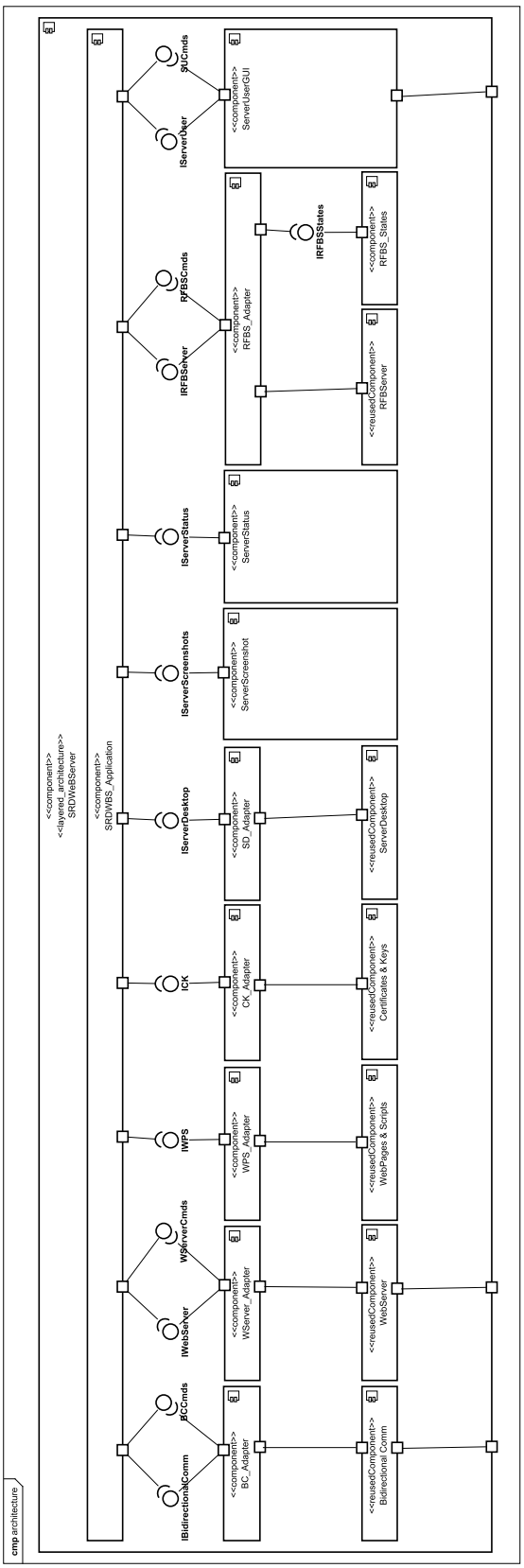

Figura 46: Diagrama de Arquitetura para o SRDWeBServer, levantado na fase D1.

APÊNDICE H – ADIT – D2

H.1 DIAGRAMAS DE ESPECIFICAÇÃO INTER-COMPONENTE SRDWEBCLIENT

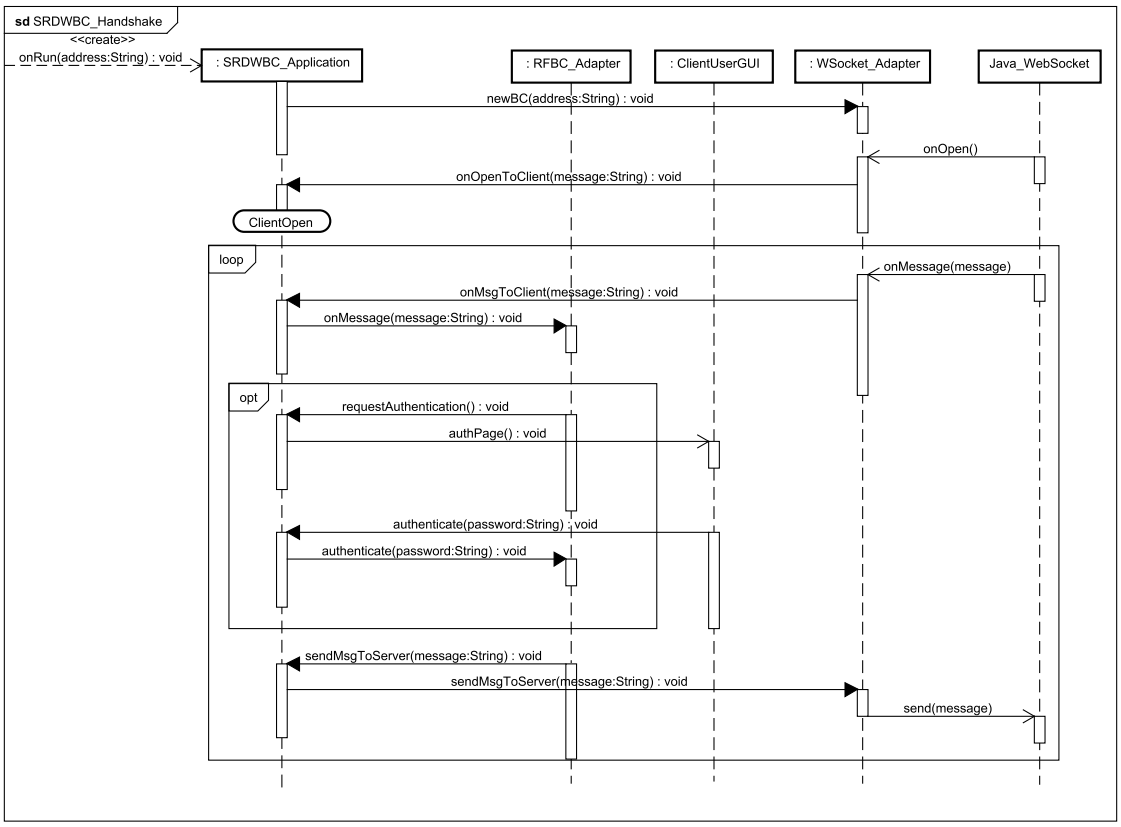

Figura 47: Diagrama de Especificação Inter-Componente para a sequência "SRDWBC_Handshake", levantado na fase D2.

H.2 DIAGRAMAS DE ESPECIFICAÇÃO INTER-COMPONENTE SRDWEBSERVER

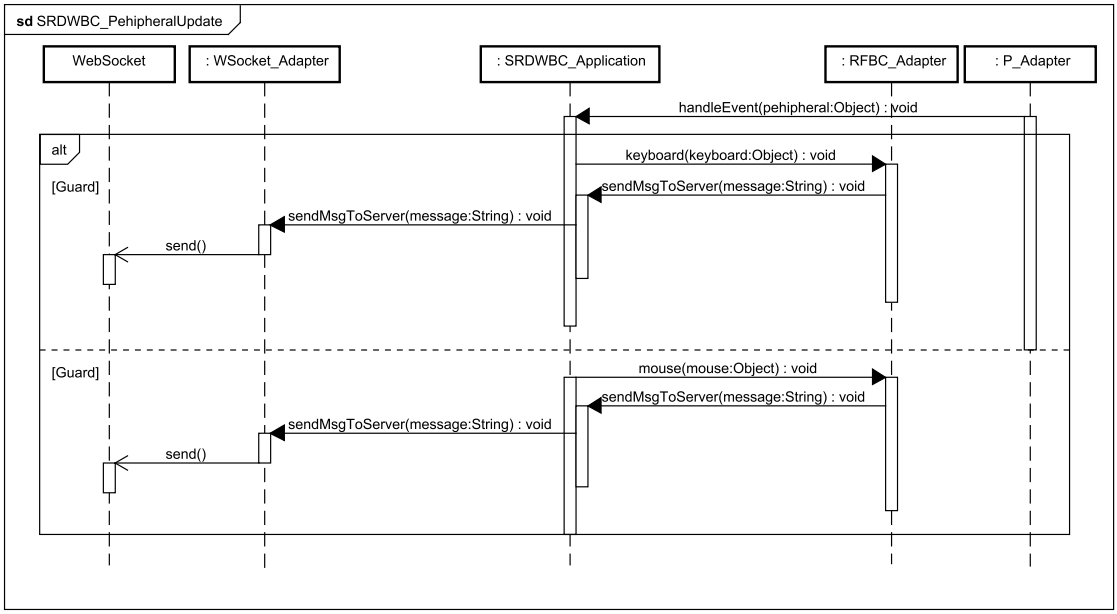

Figura 48: Diagrama de Especificação Inter-Componente para a sequência "SRDWBC_PehipheralUpdate", levantado na fase D2.

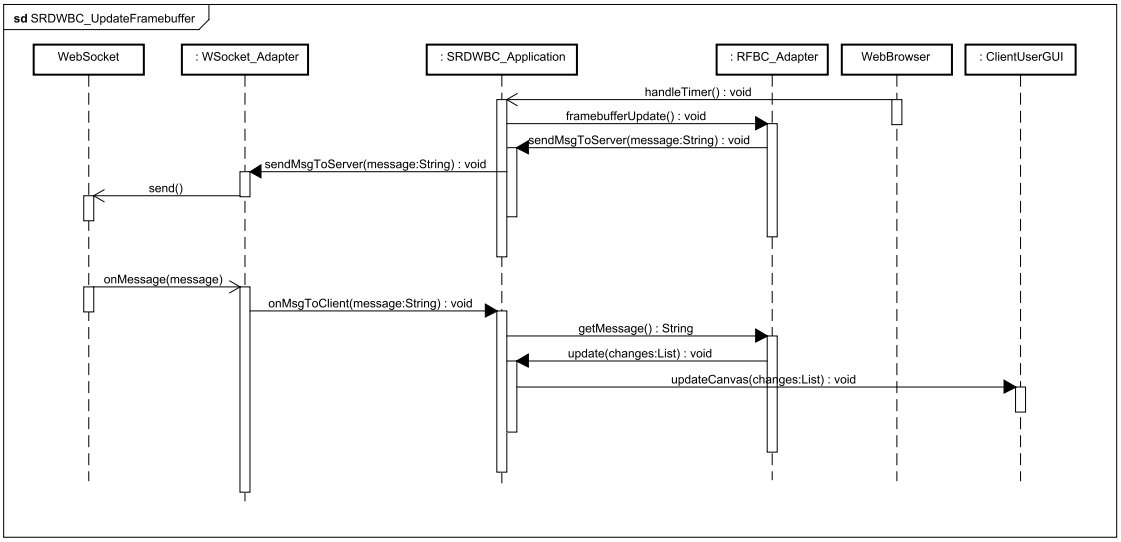

Figura 49: Diagrama de Especificação Inter-Componente para a sequência "SRDWBC_UpdateFramebuffer", levantado na fase D2.

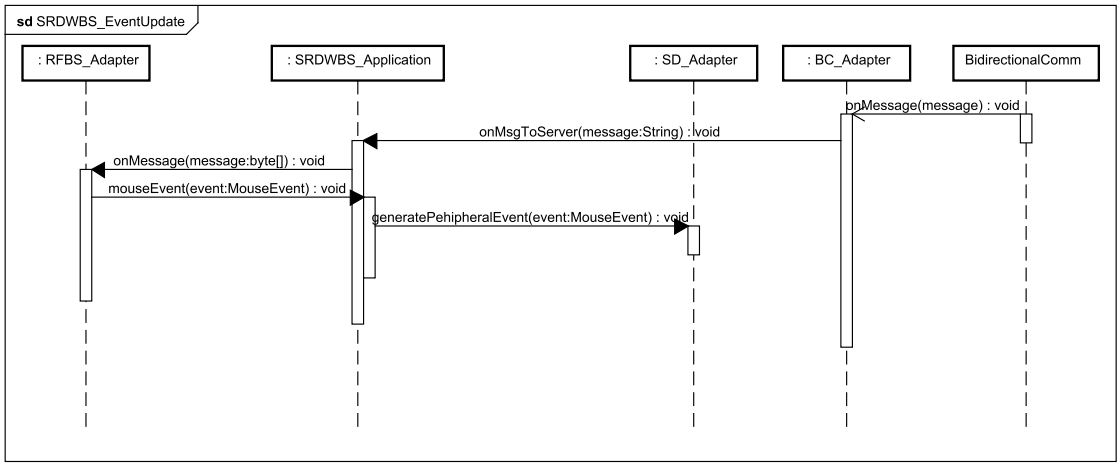

Figura 50: Diagrama de Especificação Inter-Componente para a sequência "SRDWBS_EventUpdate", levantado na fase D2.

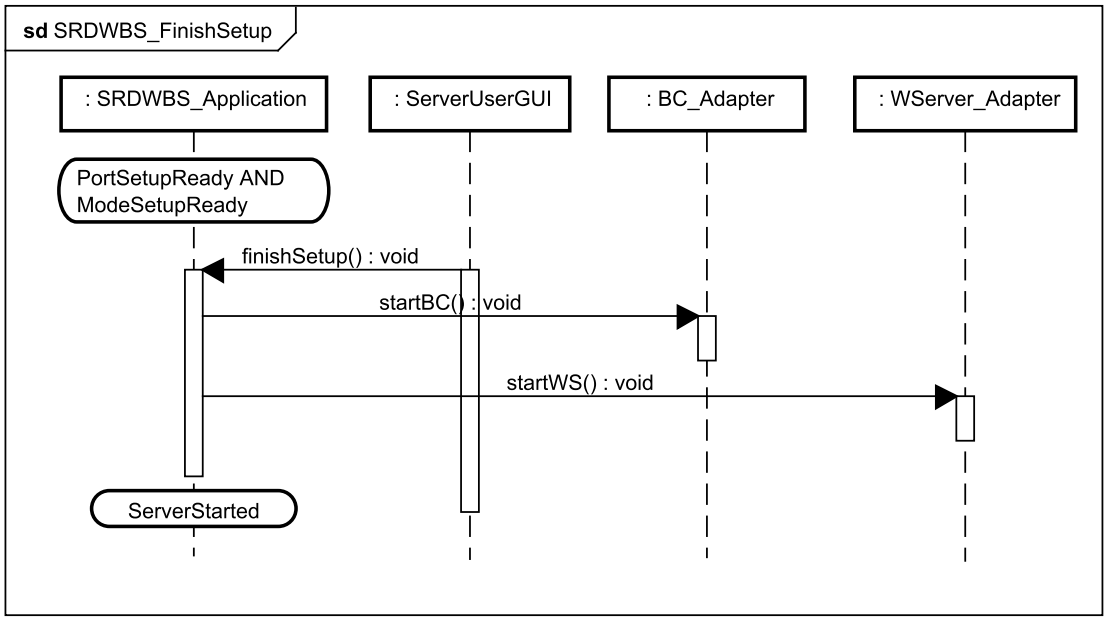

Figura 51: Diagrama de Especificação Inter-Componente para a sequência "SRDWBS_FinishSetup", levantado na fase D2.

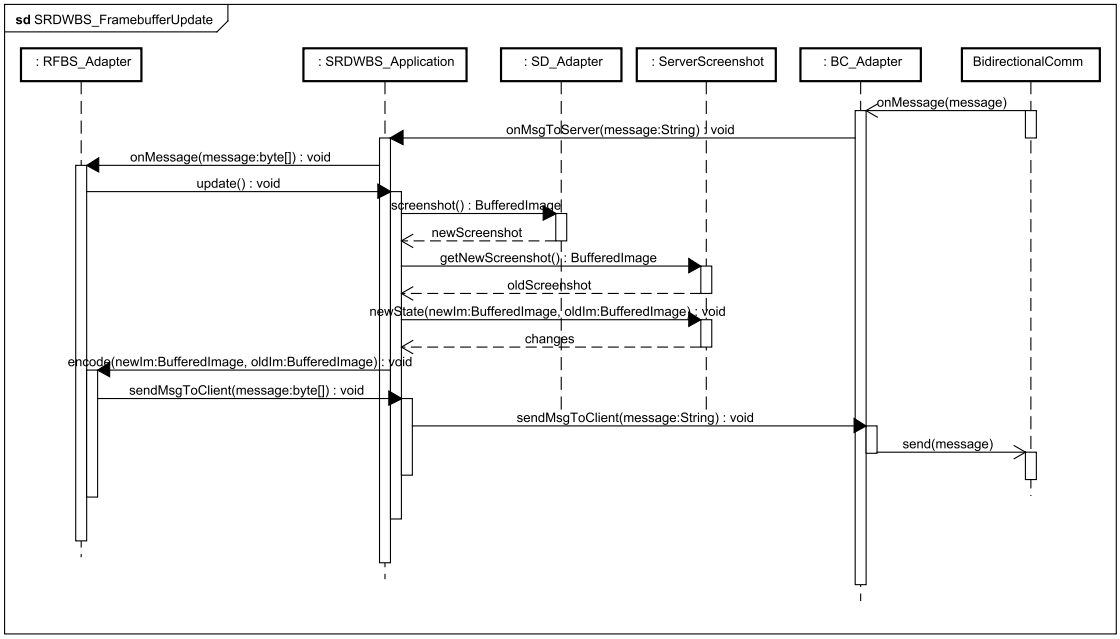

Figura 52: Diagrama de Especificação Inter-Componente para a sequência "SRDWBS_FramebufferUpdate", levantado na fase D2.

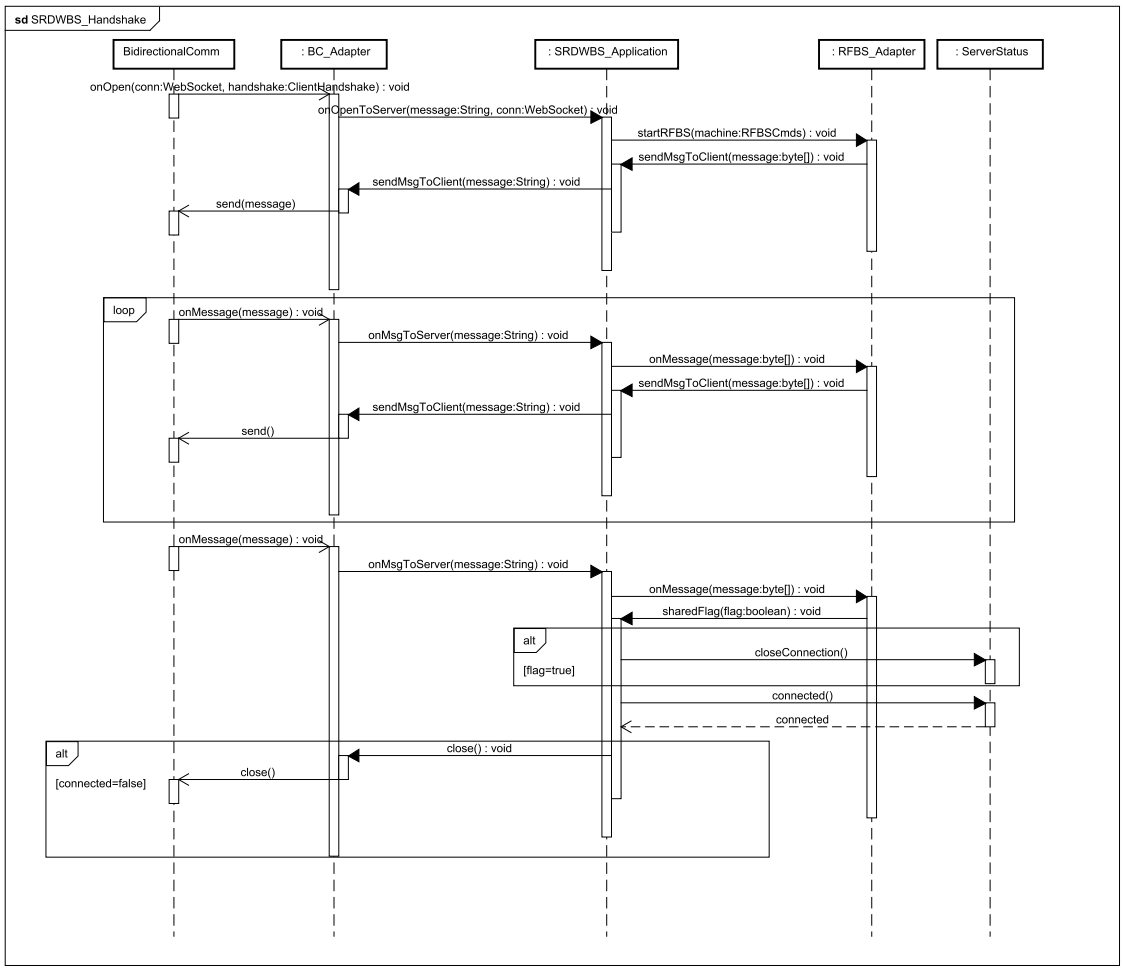

Figura 53: Diagrama de Especificação Inter-Componente para a sequência "SRDWBS_Handshake", levantado na fase D2.

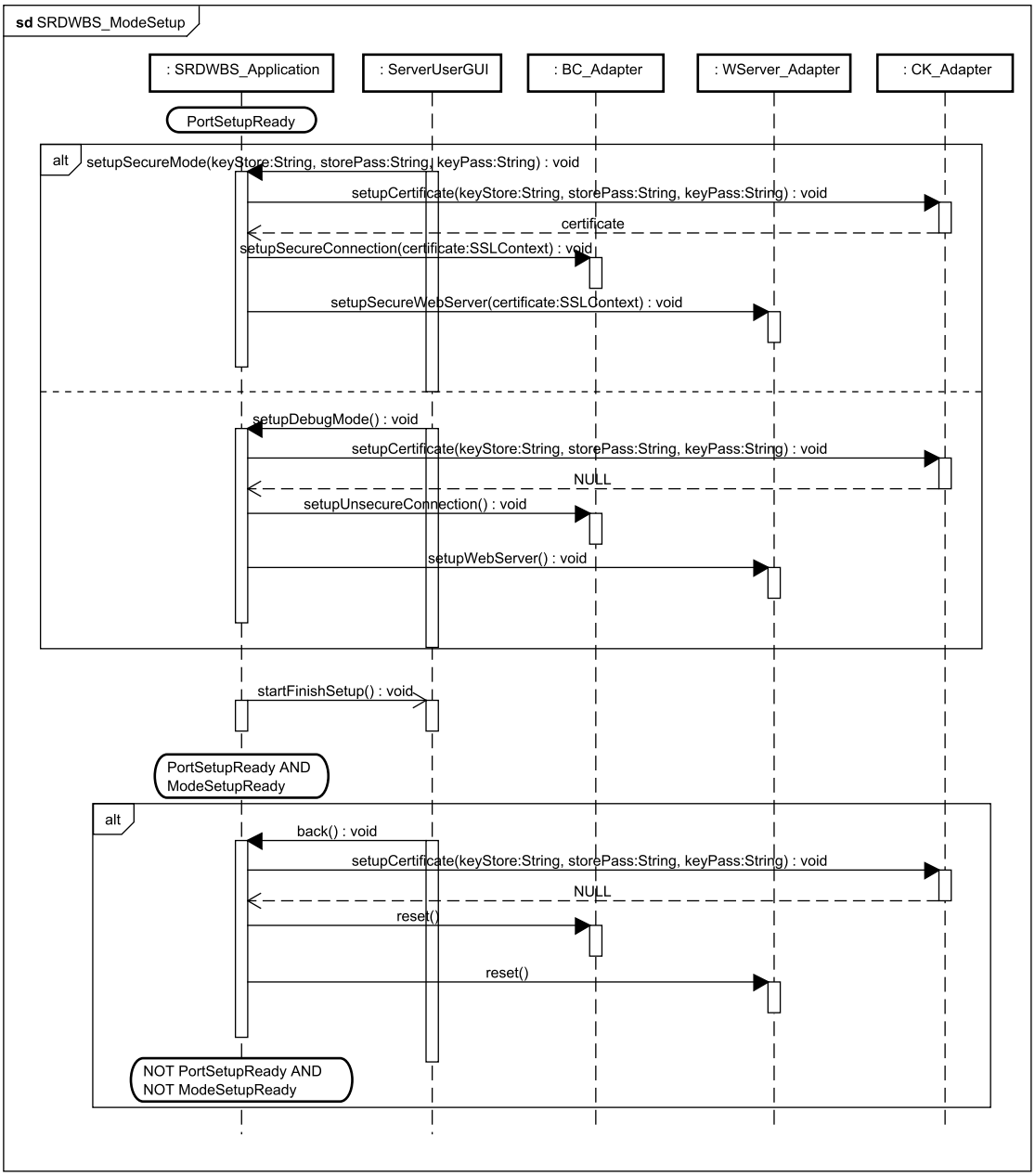

Figura 54: Diagrama de Especificação Inter-Componente para a sequência “SRDWBS_ModeSetup”, levantado na fase D2.

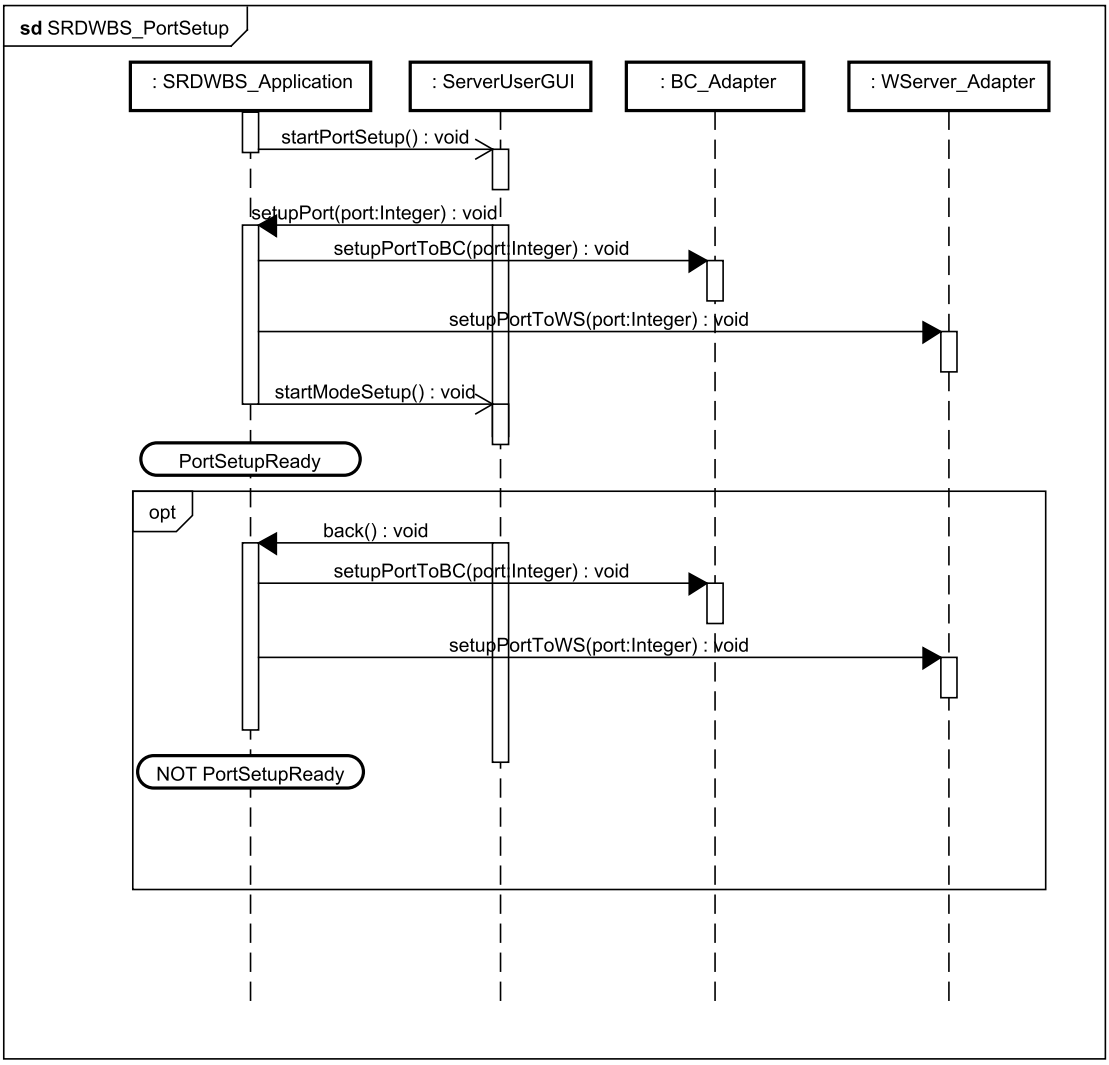

Figura 55: Diagrama de Especificação Inter-Componente para a sequência "SRDWBS_PortSetup", levantado na fase D2.

APÊNDICE I – ADIT – *IMPLEMENTATION AND TESTING*

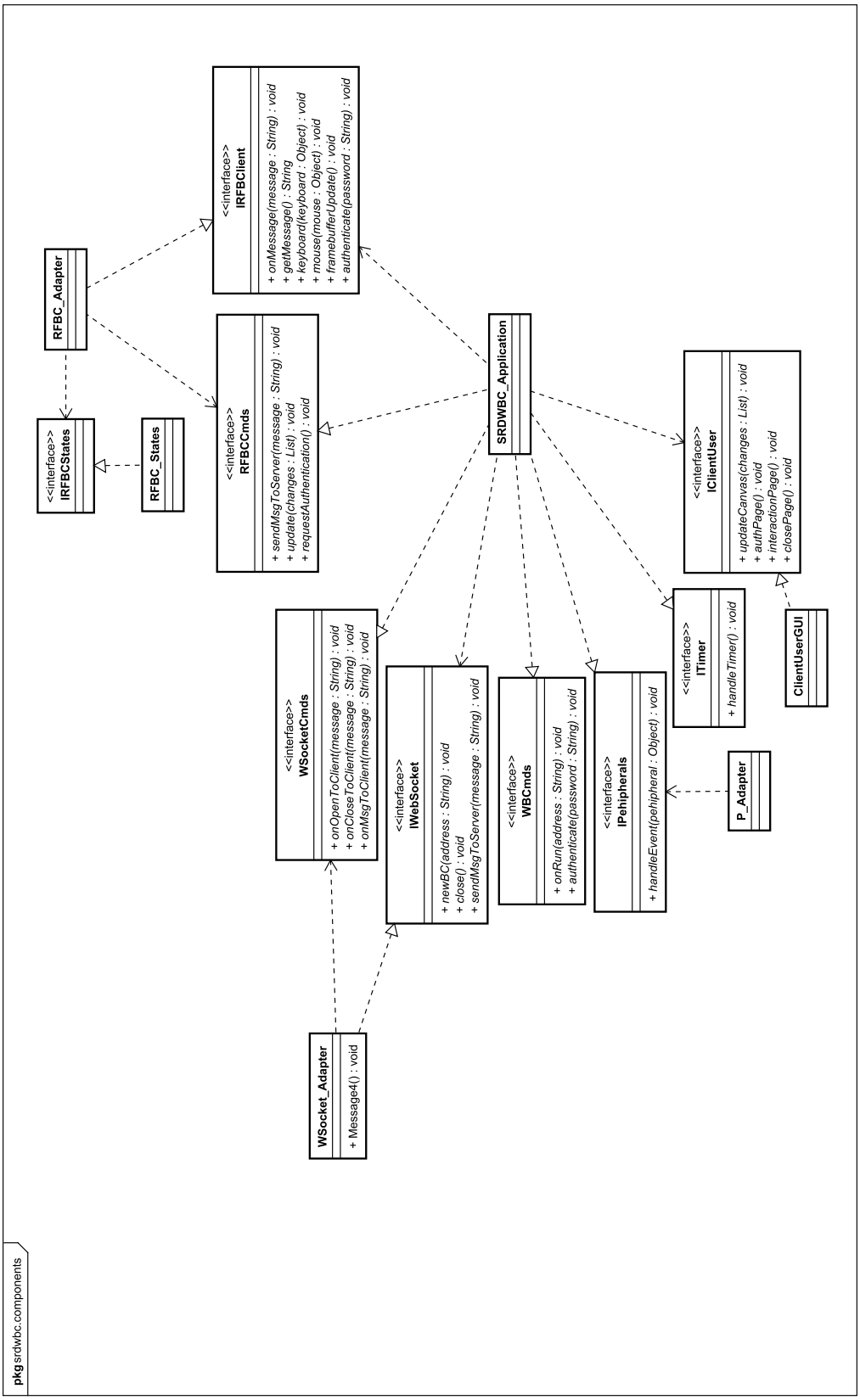

Figura 56: Diagrama de Classes para SRDWeBClient.

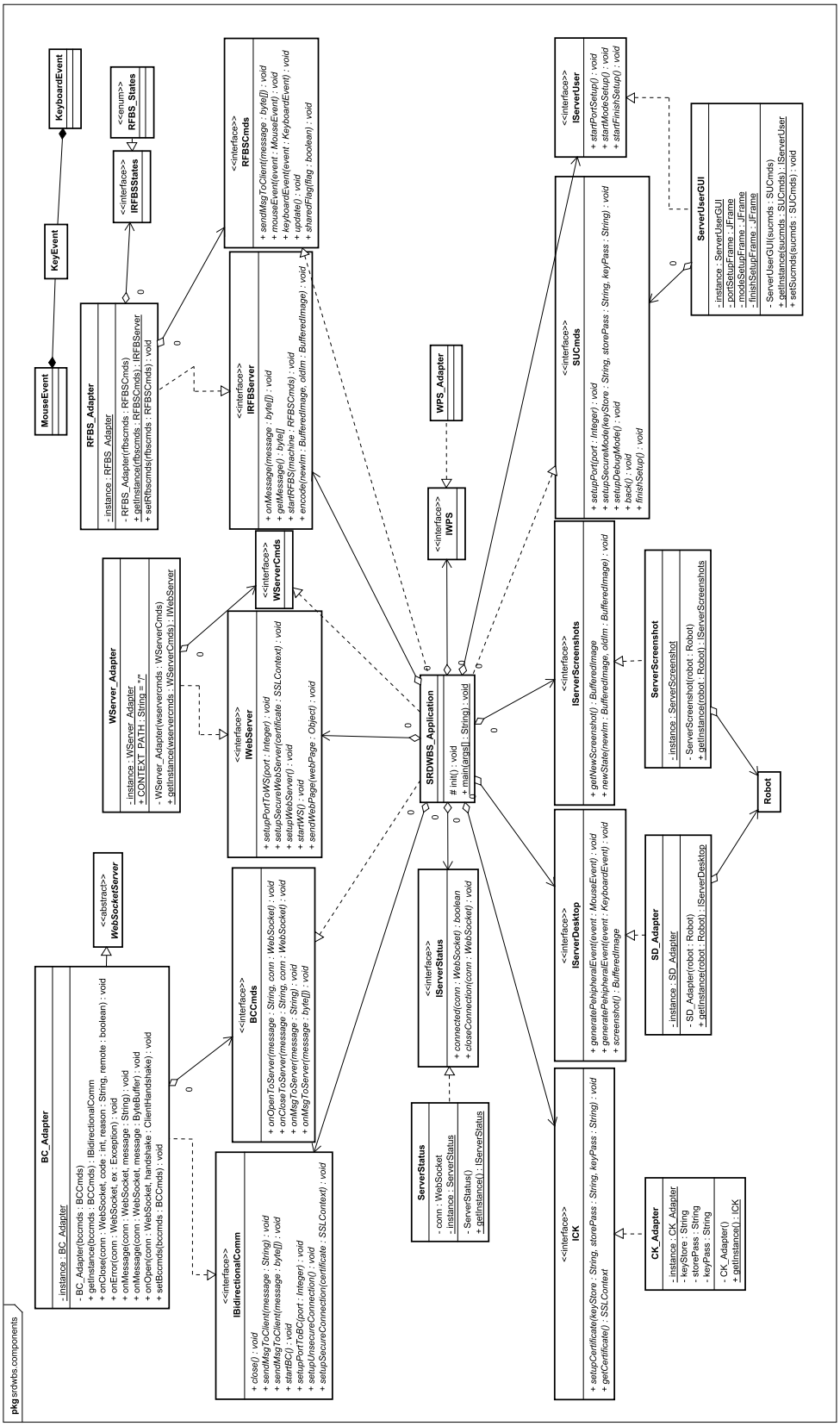

Figura 57: Diagrama de Classes para SRDWeBServer.