

# **Trabalho Final**

## **Sistemas Operacionais I**

Florianópolis, 30 de Junho de 2015.

**Tema do projeto:** aplicação em PC para configuração de rede, captura de tela, instalação de pacotes do SO e outras ações de um sistema de gerenciamento de aula em laboratório de informática.

**Sobre o projeto:** o projeto final da disciplina “*Sistemas Operacionais I*” é denominado *LabSpy* e será disponibilizado de forma gratuita e de código-aberto. O objetivo principal do programa é melhorar o processo de ensino-aprendizagem em laboratórios de computadores fornecendo ao docente ferramentas para monitoramento e controle dos computadores dos discentes.

**Grupo:**

Caique Rodrigues Marques,  
Emmanuel Podestá Junior,  
Fernando Jorge Mota,  
Fernando Paladini.

## Conteúdo abordado:

### 1-3 Primeira Entrega

#### 1. Requisitos

1.1. Requisitos funcionais

1.2. Requisitos não-funcionais

#### 2. Diagramas

#### 3. Funcionamento planejado (**importante**)

#### 4. Segunda Entrega

4.1. Novidades

4.2. Pré-requisitos

4.3. Comunicação

4.4. Visualizar Tela do Aluno

4.5. Controle Remoto

4.6. Processo para iniciar e usar as funcionalidades

4.1. Diagramas

#### 5. Terceira Entrega

5.1. Novidades

5.2. Pré-requisitos

5.3. Instalação Remota

5.4. Instalação Local

5.5. Configurações

5.6. Atualização da quantidade de frames

5.7. Atualização da quantidade de colunas

5.8. Enviar mensagem para o aluno

5.9. Editar comprimento e altura dos elementos na Grid

5.10. Bloqueio

5.11. Abrir URL

5.12. Função de Restart, Shutdown e comandos arbitrários

5.13. Adicionar IPs confiáveis para o aluno

#### 6. Funcionalidades inacabadas

6.1. Chat

6.2. Bloquear sites e endereços IPs

# 1-3 Primeira Entrega

## 1 Requisitos

### 1.1 Requisitos Funcionais

Os requisitos funcionais do projeto *LabSpy* são os que seguem abaixo:

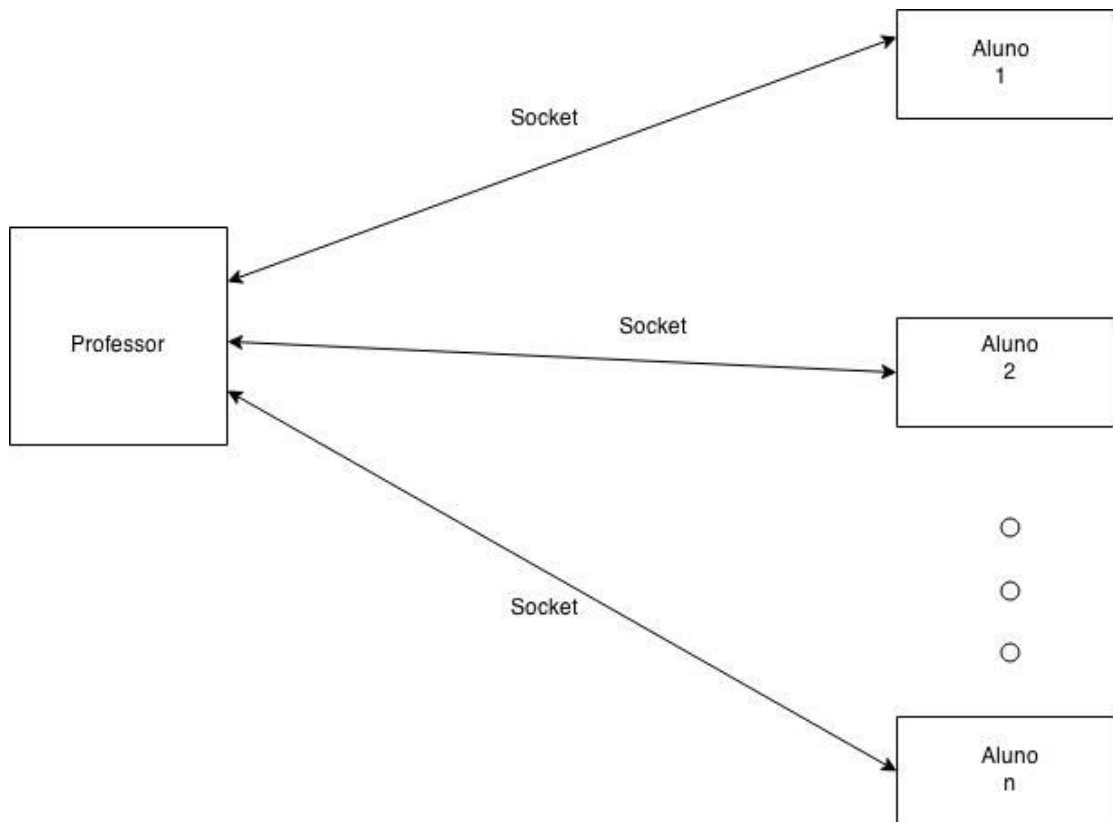
- **Overview das áreas de trabalho:** o programa deve permitir que o docente tenha um *overview* da área de trabalho de todos os computadores conectados na rede, mostrando as imagens dos *desktops* ao vivo ou em uma frequência relativamente alta;
- **Visualização de área de trabalho:** O programa deve permitir que o docente possa ter uma visualização completa (toda a área de tela do programa) da área de trabalho de um computador em específico;
- **Controle remoto do mouse:** O programa deve permitir que o docente controle o *mouse* do computador de um discente a qualquer momento;
- **Controle remoto do teclado:** O programa deve permitir que o docente controle o *teclado* do computador de um discente a qualquer momento;
- **Bloqueio de computadores:** O programa deve permitir que o docente bloqueie e desbloqueie um computador da rede em qualquer dado momento;
- **Proxy definido pelo professor:** O programa deve permitir que o docente possa bloquear sites de serem acessados pelos alunos em qualquer dado momento.
- **Chat:** O programa deve permitir que o docente possa abrir um chat e enviar mensagens instantâneas para todas os estudantes (computadores) da turma.

### 1.2 Requisitos Não-Funcionais

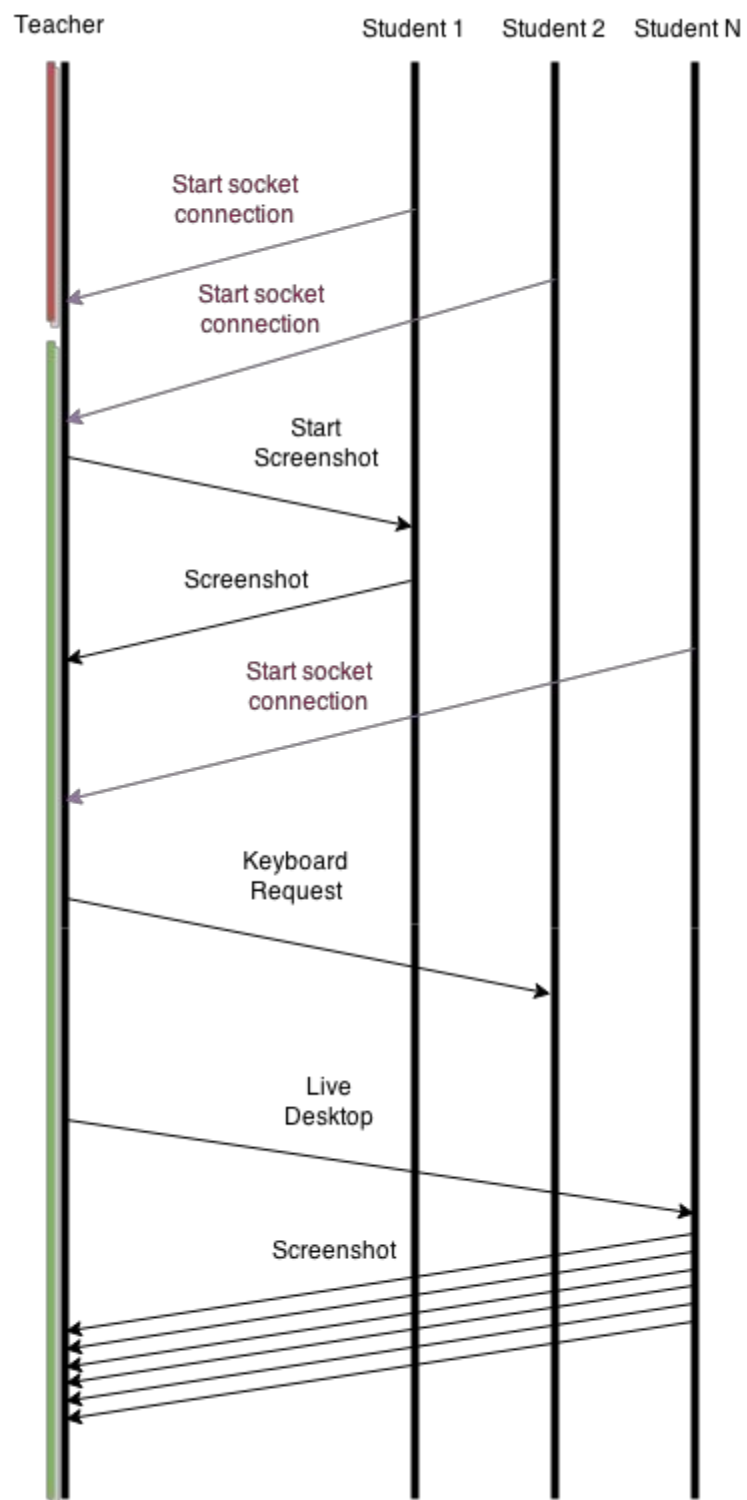
Os requisitos não-funcionais do projeto são os que seguem abaixo:

- O projeto deve ser escrito principalmente na linguagem de programação Java;
- O projeto deve ser disponibilizado de forma gratuita;
- O projeto deve ser *open-source* (código-aberto);
- O projeto deve ser multiplataforma e executar sem maiores problemas em ambientes Linux (principalmente a distribuição Ubuntu) e em ambientes Windows (Windows XP e superiores).

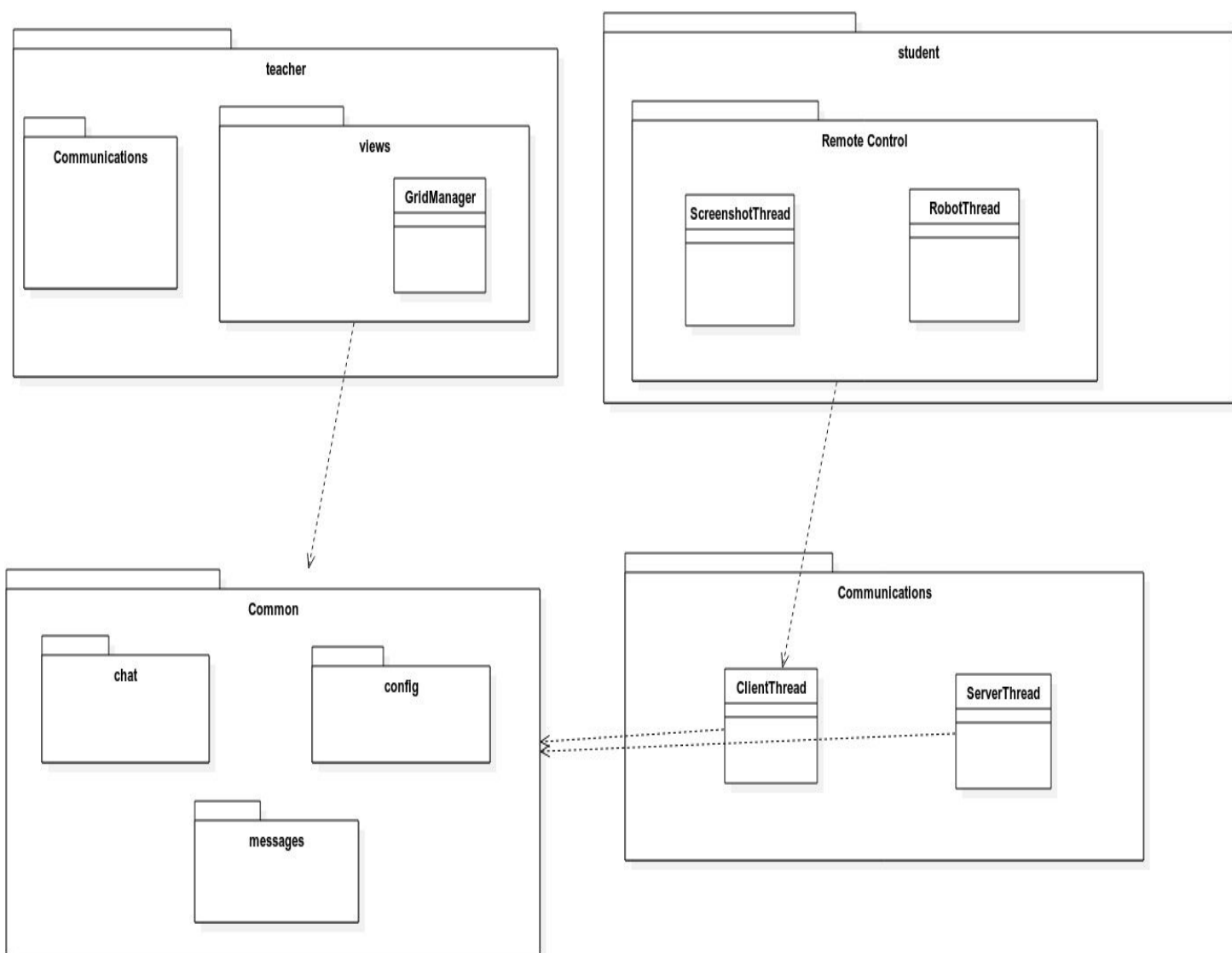
## 2 Diagramas



(Figura 1.1) Abstração da conexão entre computador do professor e dos alunos.



(Figura 1.2) Relação entre requisições e respostas de computadores em Contexto Estudante e em Contexto Professor



(Figura 1.3) Relação entre pacotes e classes do sistema.

### 3 Funcionamento planejado

Para realizar a instalação, configuração e uso do LabSpy em máquinas de um dado laboratório, os seguintes passos se fazem necessários:

1. **Instalação do LabSpy no computador do professor:** haverá um programa de *setup* para configurar o LabSpy no computador do professor. Durante o processo de instalação o administrador será orientado a criar um *hostname* ou um *IP fixo* para o computador em questão (o computador que rodará em Contexto Professor). Após finalizado, o processo fará com que o computador inicie junto com um “processo-servidor” do LabSpy, este processo permitirá que os computadores em Contexto Estudante se conectem, enquanto iniciam, ao computador do professor, estabelecendo assim uma comunicação entre os mesmos.
2. **Instalação do LabSpy no computador do aluno:** haverá um programa de *setup* para configurar o LabSpy nos computadores dos alunos. O *setup* deve ser feito em cada máquina onde é desejado o uso do LabSpy no Contexto Estudante. Esse processo de instalação fará com que o computador em questão sempre inicie com um “processo-cliente” do LabSpy que fará a conexão com o computador em Contexto Professor. Dessa forma, ao iniciar, o sistema em Contexto Estudante começará a comunicação com o computador em Contexto Professor.
3. **Aguardando conexões de computadores em Contexto Estudante:** após ser inicializado, o computador em Contexto Professor terá que aguardar pelo início dos computadores em Contexto Estudante, que ficarão responsáveis por iniciar a comunicação entre computadores *estudante-professor*. Contudo, na ocorrência de uma situação contrária (ou seja, o computador do estudante iniciar antes do computador do professor), o computador em Contexto Estudante terá que esperar até que o computador em Contexto Professor seja iniciado. A partir desse momento, o “servidor” (computador do professor) existirá e será possível que o “cliente” (computador do estudante) faça o início da comunicação. Um computador em Contexto Estudante sempre é responsável por iniciar a comunicação.
4. **Novas conexões de computadores em Contexto Cliente ao computador em Contexto Professor (ações em uma nova conexão):** para cada nova conexão iniciada pelo computador estudante, o computador em Contexto Professor vai criar uma thread cliente. Essas “threads clientes” serão utilizadas pelo computador em Contexto Professor para fazer requisições aos computadores em Contexto Estudante. Através delas será possível separar a parte “servidor” da parte “cliente” no computador em Contexto Professor.
5. **Requisições do professor:** quando uma requisição for feita pelo computador em Contexto Professor, ela será transmitida pela “thread cliente” respectiva ao computador estudante em questão, que tratará essa requisição e fornecerá a sua respectiva resposta ou ação.



## 4 Segunda entrega

### 4.1 Novidades

- Refatoramento do código, professor passa a ser cliente, e o aluno, servidor;
- Nova classe, `BaseClientThread`, responsável pela comunicação assíncrona entre aluno e professor;
- Instalação remota do programa via SSH;
- Possibilidade de controle remoto do computador do aluno através do computador do professor;
- Comunicação assíncrona entre aluno e professor, portanto, quando um aluno envia uma mensagem, ele não necessita ficar bloqueado esperando resposta do professor.

### 4.2 Pré-requisitos

Esta primeira versão do LabSpy possui alguns pré-requisitos que são necessários para que o sistema funcione corretamente.

- 1) Pré-requisitos gerais (necessário para ambos os computadores)
  - Oracle JRE 1.8\_45, disponível no [site oficial](#) do Java;
  - Sistema operacional Ubuntu utilizando Upstart (apenas anteriores ao Ubuntu 15.04).
  - Caso deseje *compilar* o projeto em *.jar*, também é necessário JDK 1.8\_45 e Ant 1.9.4.
- 2) Pré-requisitos específicos (computadores dos estudantes)
  - Um servidor SSH instalado, configurado e rodando (apenas para instalação);
  - Sistema não-*headless*, ou seja, com um servidor X instalado (Xorg Server, por exemplo).
- 3) Pré-requisitos específicos (computador do professor)
  - Acesso aos computadores dos estudantes como *root*;

### 4.3 Comunicação

Há dois processos em comunicação para manter a sincronização e o correto funcionamento do monitoramento remoto. A comunicação é feita entre os dois processos de forma assíncrona, isto é, assim que um envia uma requisição, este não necessita estar bloqueado esperando a resposta do outro processo. Neste projeto, a classe que é responsável pela conversação entre cliente (professor) e servidor (aluno) é a `BaseClientThread`, e a comunicação é feita através de sockets. Por cliente e servidor, entende-se que servidor é o processo que aguarda conexões provenientes de outros processos (ou seja, é passivo), enquanto que cliente é o processo que inicializa a conexão a um servidor (ou seja, é ativo). Destacamos isso pois, após o estabelecimento da comunicação, dados são tanto transmitidos, quanto recebidos, de ambos os lados da conexão. Para o envio das mensagens, buffers são usados como caixa postal para garantir o funcionamento correto de escritas e leituras entre os dois processos; uma requisição é enviada ao buffer, que é enviado ao destinatário, que pode então responder (escrevendo uma

requisição como resposta) ou apenas ler os dados em questão. Essas requisições tem tamanho variável, portanto, limites são traçados de forma que não hajam inconsistências e/ou inconveniências durante a troca de mensagens, que são enviados de maneira sequencial e possuem o tamanho especificado logo no início da requisição.

#### **4.3.1. Buffers**

Para o buffer de leitura de dados, um espaço fixo de 128 bytes é alocado, para indicar o tamanho do dado a ser lido, enquanto o segundo espaço, é reservado à mensagem em si.

Para o buffer de escrita, uma requisição é selecionada de uma queue de requisições, e depois, o socket escreve no buffer de escrita, sequencialmente.

#### **4.3.2 Leitura e escrita**

A leitura é realizada de forma com que, de princípio, haja uma verificação no tamanho do buffer de leitura (readSize), que, se possuir valor diferente de -1, indica que está no meio do processamento de uma mensagem com tamanho igual a readSize. Porém, caso o tamanho (readSize) possua valor diferente de -1, um tamanho é definido a ele de acordo com o início da próxima mensagem (pois uma mensagem está para ser recebida). Sabendo o tamanho da próxima mensagem, a mensagem é carregada em pedaços definidos automaticamente pelo sistema operacional e, quando totalmente carregada, a mensagem a ser lida é descompactada e então interpretada usando uma classe nativa do Java chamada ObjectInputStream, que desserializa a mensagem e passa para o método receiveMessage tratar.

A escrita de uma mensagem é feita com verificação se o buffer do sistema está disponível para escrita, se sim, uma requisição de uma queue de requisições é selecionado. Se não houver mais requisições de escrita, a verificação do buffer de escrita do sistema é desativada, senão, o buffer do sistema operacional é verificado tanto para escrita quanto para a leitura. Se a verificação do buffer de escrita está desativada e o método sendMessage é chamado, ele “acorda” a thread (usando uma operação wakeup) para que esta verifique o queue de requisições e então comece a verificar o buffer tanto para escrita quanto para leitura.

#### **4.3.3. Envio de mensagens**

Uma entrada necessita ser compactada antes de ser enviada ao pool de mensagens a enviar, para isso, o DeflaterOutputStream comprime os dados de entrada, enquanto o ObjectOutputStream escreve a mensagem. O buffer tem seu espaço alocado e, depois, a mensagem é enviada ao pool.

#### **4.3.4 Recebimento de mensagens**

O recebimento de mensagens varia, porque professor e aluno têm de receber mensagens específicas para realizarem as suas tarefas. Para o aluno (student/src/communication/ClientThread.java), ele pode receber instâncias de mensagens e operar de acordo com isto:

- Numa instância de *StartScreenshot*, é iniciada uma thread que envia screenshots de forma periódica, respeitando largura e altura especificados na operação;
- Numa instância de *ResizeScreenshot*, uma solicitação de redimensionamento do screenshot da tela do computador do aluno é feito, de forma que novas screenshots sejam enviados com novas largura e altura;
- Numa instância de *StopScreenshot*, o envio de screenshots é parado;
- Numa instância de *RobotMessage*, é enviada uma mensagem ao RobotThread, que pode ser relacionada a operações como mover o mouse, efetuar um clique ou pressionar uma tecla no teclado, por exemplo;
- Ou, se não for nenhuma das instâncias listadas, ele imprime a string contida na mensagem de forma que o usuário possa verificar o que causou ela.

O professor (teacher/src/communication/ClientThread.java) também receberá instâncias diferentes, atuando de uma forma específica:

- Numa instância de *Screenshot*, um screenshot é recebido e anexado como sendo o último screenshot do cliente em questão;

#### 4.4 Visualizar Tela do Aluno

O professor poderá visualizar a tela do aluno por meio de uma comunicação assíncrona, descrita no tópico “Comunicação”. O aluno enviará mensagens para o professor consecutivamente por meio de sockets, o professor poderá ler essas mensagens e mostrá-las em sua tela, sempre atualizando de acordo com cada mensagem lida. Essa tela de visualização ficará em um grid com várias outras telas de outros computadores, se estes forem configurados, isto é, o professor poderá visualizar em uma única tela, todas as telas dos computadores configurados e poderá selecioná-las para uma visualização com uma dimensão maior.

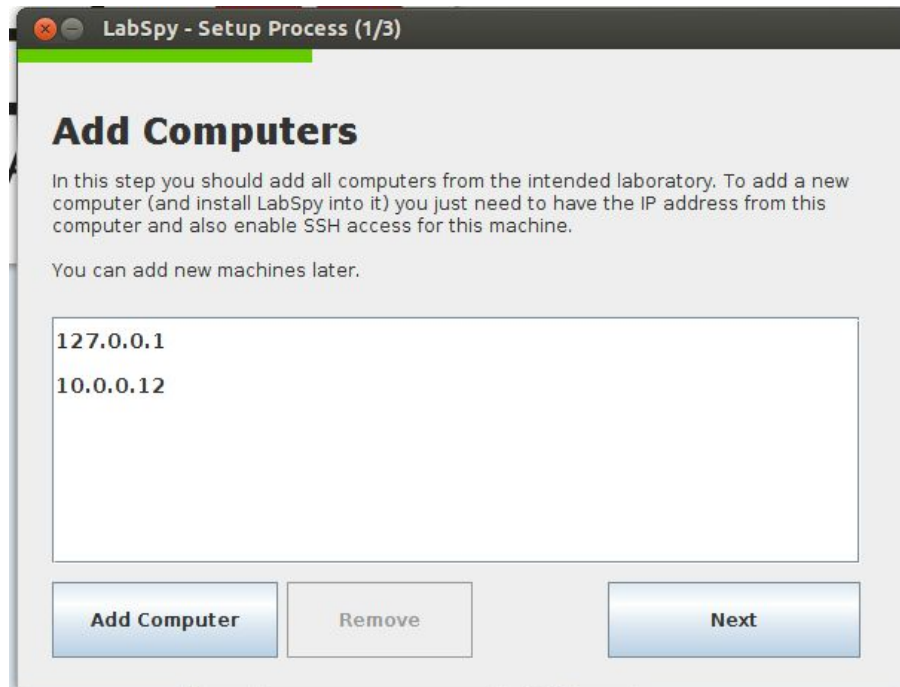
#### 4.5 Controle Remoto

O professor poderá controlar remotamente o computador do aluno por meio de seu mouse e teclado. Ele terá uma grid com as imagens da tela dos alunos, como dito anteriormente, e cada tela será selecionável, possibilitando que seja selecionada apenas o computador que o professor queira controlar remotamente. Cada visualização selecionável é relacionada com o IP da máquina que está enviando screenshots, fazendo uma comunicação direta com a máquina que se quer controlar.

#### 4.6 Processo para iniciar e usar as funcionalidades

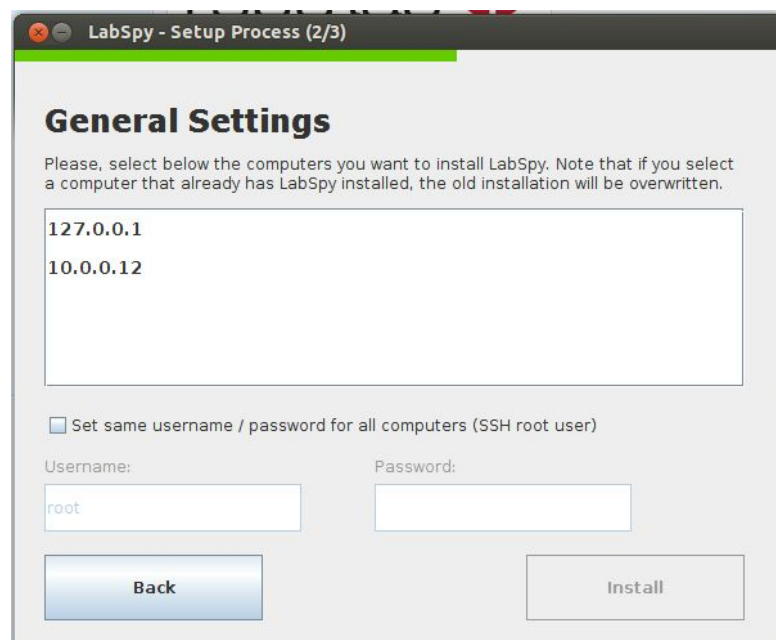
O LabSpy em sua versão atual já fornece suporte para instalação remota via SSH. Para começar a instalação basta utilizar a interface gráfica do programa e clicar em “Functions -> Configuration”. Nota: caso seja a primeira vez que o programa esteja sendo executado, o LabSpy vai perguntar se o usuário deseja configurar os computadores. Ao clicar em “Configuration”, existem 3 etapas para configurar os computadores dos estudantes:

**1ª Etapa:** o usuário adiciona os endereços IP dos computadores desejados, ou seja, o endereço IP dos computadores dos estudantes.



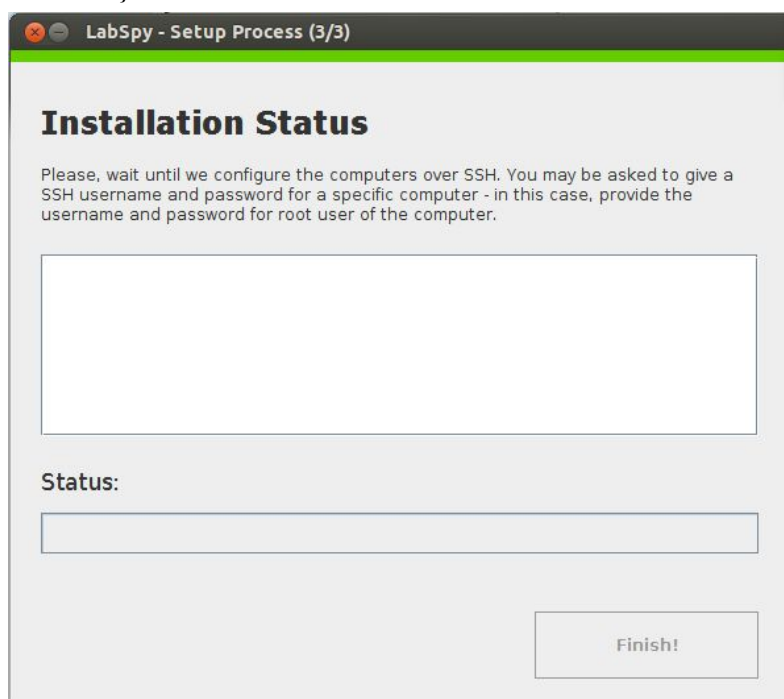
The screenshot shows the 'LabSpy - Setup Process (1/3)' window. The title bar is dark gray with standard window controls. The main content area has a light gray background. At the top, the heading 'Add Computers' is in bold black font. Below it, a paragraph explains that the user should add all computers from the intended laboratory and that they just need the IP address and enable SSH access. A sub-note says 'You can add new machines later.' Below this is a large white text area containing two IP addresses: '127.0.0.1' and '10.0.0.12'. At the bottom, there are three buttons: 'Add Computer' (blue), 'Remove' (gray), and 'Next' (blue).

**2ª Etapa:** permite que o usuário selecione os computadores adicionados que terão o LabSpy adicionado. Além disso é possível definir um *username* e *senha* universal (para todos os computadores) para o processo de login via SSH. Caso os computadores dos estudantes tenham as mesmas credenciais para a conta de root, esta é uma opção que pode economizar muito tempo.



The screenshot shows the 'LabSpy - Setup Process (2/3)' window. The title bar is dark gray with standard window controls. The main content area has a light gray background. At the top, the heading 'General Settings' is in bold black font. Below it, a paragraph asks the user to select computers to install LabSpy and notes that existing installations will be overwritten. Below this is a large white text area containing the same two IP addresses: '127.0.0.1' and '10.0.0.12'. Below the text area is a checkbox labeled 'Set same username / password for all computers (SSH root user)'. Underneath are two input fields: 'Username:' with the value 'root' and 'Password:'. At the bottom, there are two buttons: 'Back' (blue) and 'Install' (gray).

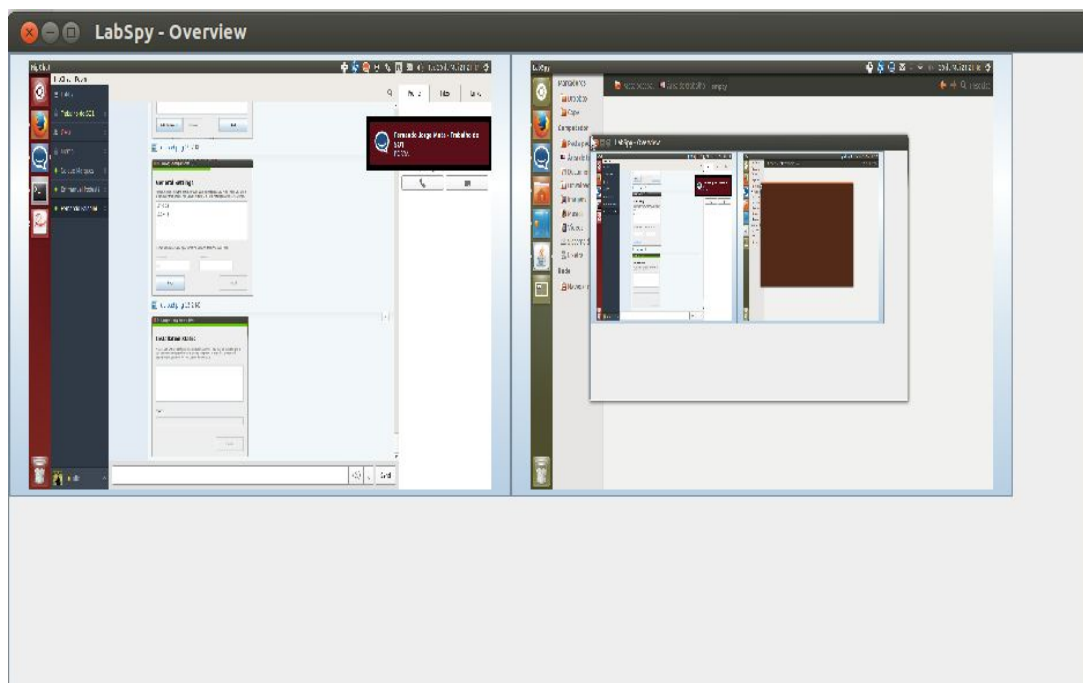
**3ª Etapa:** todos os computadores selecionados terão o LabSpy instalado e configurado. Caso a opção de *username* e *senha* universal não tenham sido selecionados, para cada computador o sistema pedirá informações de *username* e *senha*.

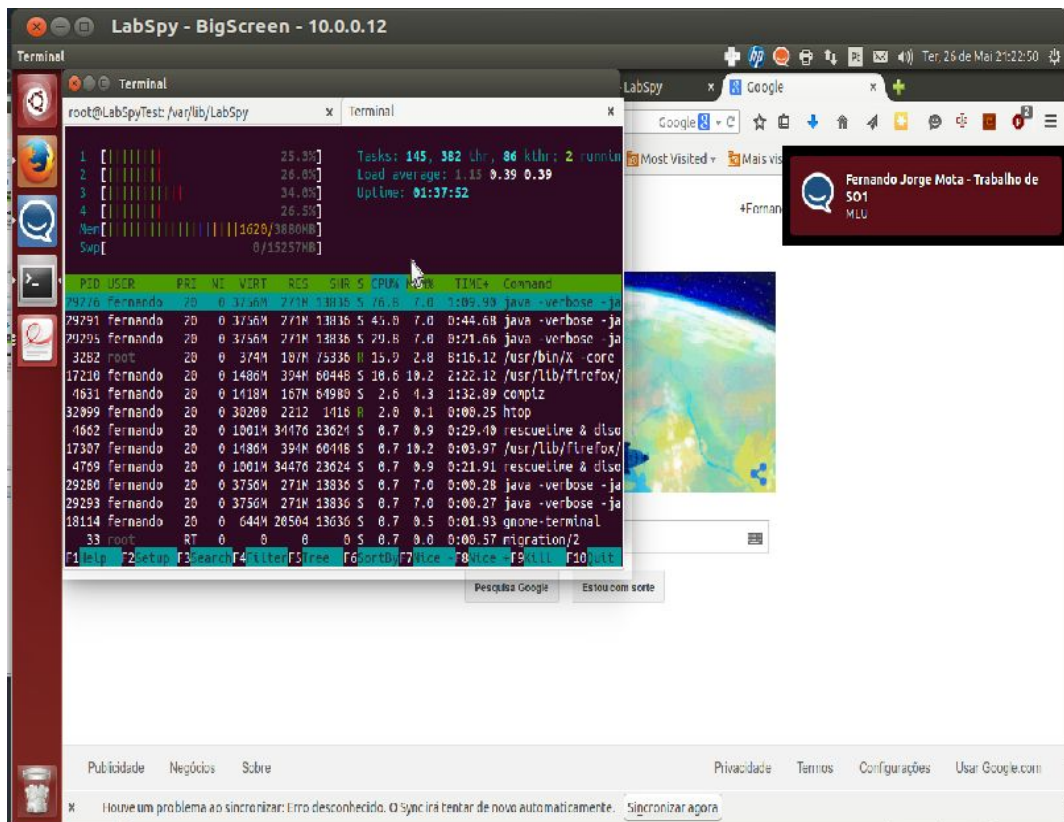


De forma mais técnica, a conexão e instalação via SSH consiste no envio dos arquivos “assets/labspy.sh” e “out/artifacts/Student/Student.jar” para a pasta “/var/lib/LabSpy/” no computador remoto. O primeiro arquivo é um script que permite realizar um comando de *start* e *stop* no Student.jar, que consiste em um servidor cuja o qual o computador do professor se conectará. Após a transferência dos dois arquivos para o computador remoto, um symlink é criado do arquivo /var/lib/LabSpy/labspy.h para /etc/init.d/labspy\_client. Um último comando que utiliza o Upstart é feito para configurar o labspy\_client como um script a ser iniciado junto com o sistema (em seu boot). A partir de agora, toda vez que o computador do estudante iniciar, o Student.jar (que é parte do LabSpy) iniciará junto com o sistema, permitindo assim que o computador do professor possa realizar uma conexão e acessá-lo remotamente.

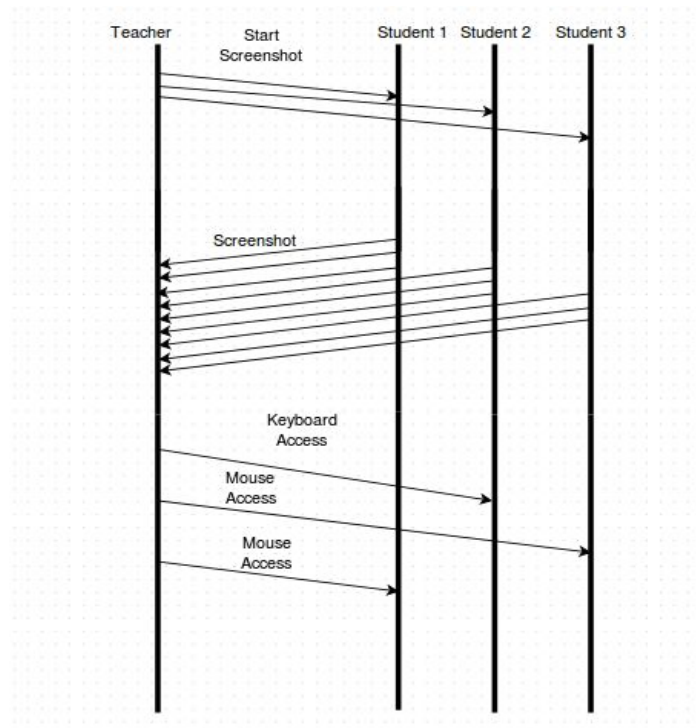
Após realizar a configuração das máquinas (incluindo reinício do sistema para que o Student.jar possa estar rodando) basta clicar em “Overview -> Screenshot” para começar a visualizar a tela dos computadores dos estudantes. Você verá uma nova tela com uma grid com a tela dos computadores configurados já enviando mensagens com suas screenshots, ou seja, mostrando o que está na suas telas para o computador do professor. Caso o docente queira controlar remotamente um computador do aluno, é possível selecionar a tela do computador que se quer controlar e será aberta uma nova janela com a tela do computador selecionado. Após esta ação, o professor já estará controlando o computador do aluno, podendo escrever com o teclado e mexer com o mouse no computador do aluno.

**Imagens do programa executando:**



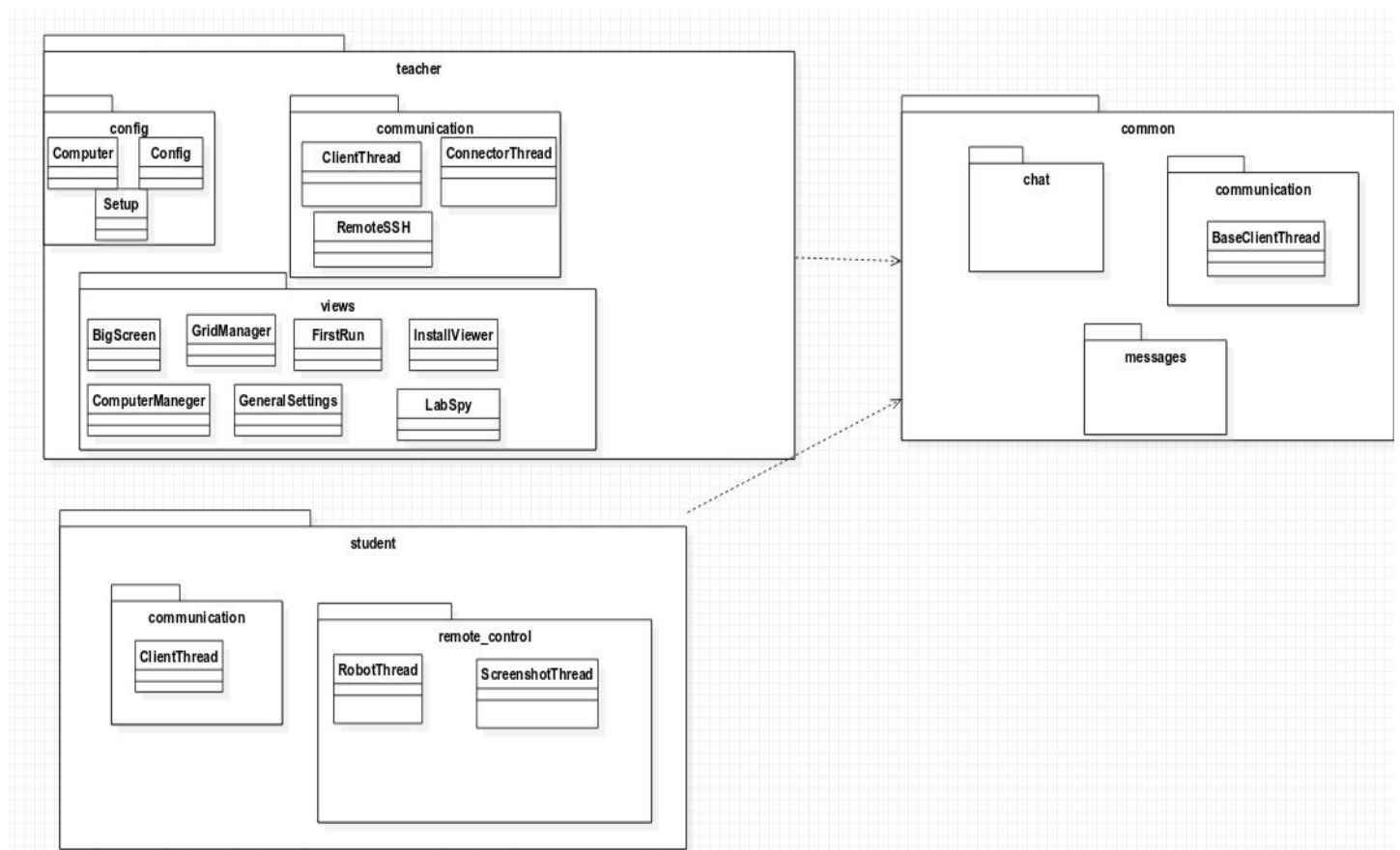


## 4.7 Diagramas



(Figura 1.1) Relação entre requisições e respostas de computadores em Contexto Estudante e em Contexto Professor





(Figura 1.2) Relação entre pacotes e classes do sistema.

## 5 Terceira Entrega

### 5.1 Novidades

- Suporte a *labels* para computadores dos alunos cadastrados. Em outras palavras, agora é possível dar um nome para um computador, não se fazendo mais necessário uma identificação penosa através do endereço IP;
- Grandes melhorias na confiabilidade da instalação remota;
- Computador do estudante se conecta somente a computadores confiáveis (caso tenham sido configurados);
- Suporte à extensão do sistema de instalação remota para outros SOs;
- Adicionada funcionalidade de reiniciar ou desligar computador do estudante remotamente. É possível suportar novos comandos facilmente e também estender para outros SOs;
- Adicionada funcionalidade de enviar comandos específicos a serem executados no computador do estudante;
- Melhorias no sistema de “Overview” (visualizar todos os computadores) e “Big Screen” (visualizar um computador em específico com uma taxa de atualização maior, controle de mouse / teclado, etc.);
- Configurações de frames por segundo, altura e largura dos computadores e número de colunas no Overview.
- Adicionados scripts para instalação local do software (tanto para a versão professor quando para versão estudante);
- Envio de mensagem de texto para alunos (específico ou todos).
- Abrir URL em um navegador no computador dos estudantes (específico ou todos).

### 5.2 Pré-requisitos

A segunda versão do LabSpy possui alguns pré-requisitos que são necessários para que o sistema funcione corretamente.

- 1) Pré-requisitos gerais (necessário para ambos os computadores)
  - a) Oracle JRE 1.8\_45, disponível no [site oficial](#) do Java;
  - b) Sistema operacional Ubuntu utilizando Upstart (apenas anteriores ao Ubuntu 15.04).
  - c) Caso deseje *compilar* o projeto em *.jar*, também é necessário JDK 1.8\_45 e Ant 1.9.4.
- 2) Pré-requisitos específicos (computadores dos estudantes)
  - a) Um servidor SSH instalado, configurado e rodando (apenas para instalação);
  - b) Sistema não-*headless*, ou seja, com um servidor X instalado (Xorg Server, por exemplo).
- 3) Pré-requisitos específicos (computador do professor)
  - a) Acesso aos computadores dos estudantes como *root*;

### 5.3 Instalação Remota

A partir desta versão, o LabSpy permite a instalação e configuração dos computadores remotos de forma mais extensível: agora é possível estender uma classe abstrata de configurações e criar um setup exclusivo para cada sistema operacional, se desejado. Por padrão o LabSpy suporta a instalação do cliente em sistemas operacionais Ubuntu que utilizem o Upstart (ou seja, anteriores ao Ubuntu 15.04), como já mencionado nos pré-requisitos.

Esta versão da instalação remota está (finalmente) funcional e muito mais confiável. Na versão anterior existiam problemas de dependências na geração dos arquivos JARs que foram detectados após a *Entrega 2* e também existia um problema no boot do LabSpy versão estudante, que iniciava antes do servidor X e por isso abortava a sua execução. Tais problemas foram resolvidos e como previamente afirmado a confiabilidade da instalação remota também foi melhorada: sempre é removida a versão antiga do LabSpy da máquina (o processo em execução é parado, o cliente é tirado do processo de bot, arquivos são apagados) e somente após essa etapa é instalado o novo cliente de forma remota.

### 5.4 Instalação Local

Foram criados *scripts* para realizar a instalação local do LabSpy, ou seja, é possível baixar o projeto do LabSpy e realizar uma instalação local, podendo-se escolher entre a instalação do “modo professor” ou “modo estudante”. Para realizar a instalação basta rodar, a partir da pasta do projeto, “*chmod +x install.sh*” e “*./install.sh <modo>*”, onde *<modo>* pode ser “*teacher*” ou “*student*”. Ou seja: “*./install.sh teacher*” para instalar o modo professor neste computador ou “*./install.sh student*” para instalar o modo estudante neste computador. Existe também um *script* para realizar a desinstalação do programa que pode ser chamado com “*./uninstall <modo>*”

Caso tenha instalado o “modo professor”, é possível acessar o LabSpy indo ao *Dashboard* da interface *Unity* (ou *Applications*, do *Gnome* - ou equivalentes em outras interfaces) e buscando por “LabSpy”. Também é possível iniciar o programa a partir do Terminal (em qualquer diretório do sistema) digitando-se “*labspy*”. A instalação é feita na pasta do usuário que executou o script de instalação, mais precisamente em “*.labspy/*” (em outras palavras, em “*~/.labspy/*”).

Caso tenha instalado o “modo estudante”, o cliente já foi configurado para iniciar com o sistema e caso a instalação tenha sido bem sucedida, o cliente já deve estar rodando na máquina (sem ser necessário reinicializar a máquina). Vale notar que é necessário que os pré-requisitos descritos anteriormente tenham sido cumpridos (como por exemplo a instalação e configuração da Oracle JRE mais recente). A instalação na máquina do estudante pode ser encontrada em “*/var/lib/LabSpy/*”.

### 5.5 Configurações

Agora, ao adicionar novos computadores, é possível acrescentar uma *label* a cada computador permitindo assim uma melhor compreensão do que cada endereço IP representa (antes os computadores eram apresentados apenas através do seu endereço IP). Uma nova classe de configuração foi implementada para o lado do estudante, que é uma classe que basicamente consiste do IP da máquina do professor ao qual o estudante deve se conectar.

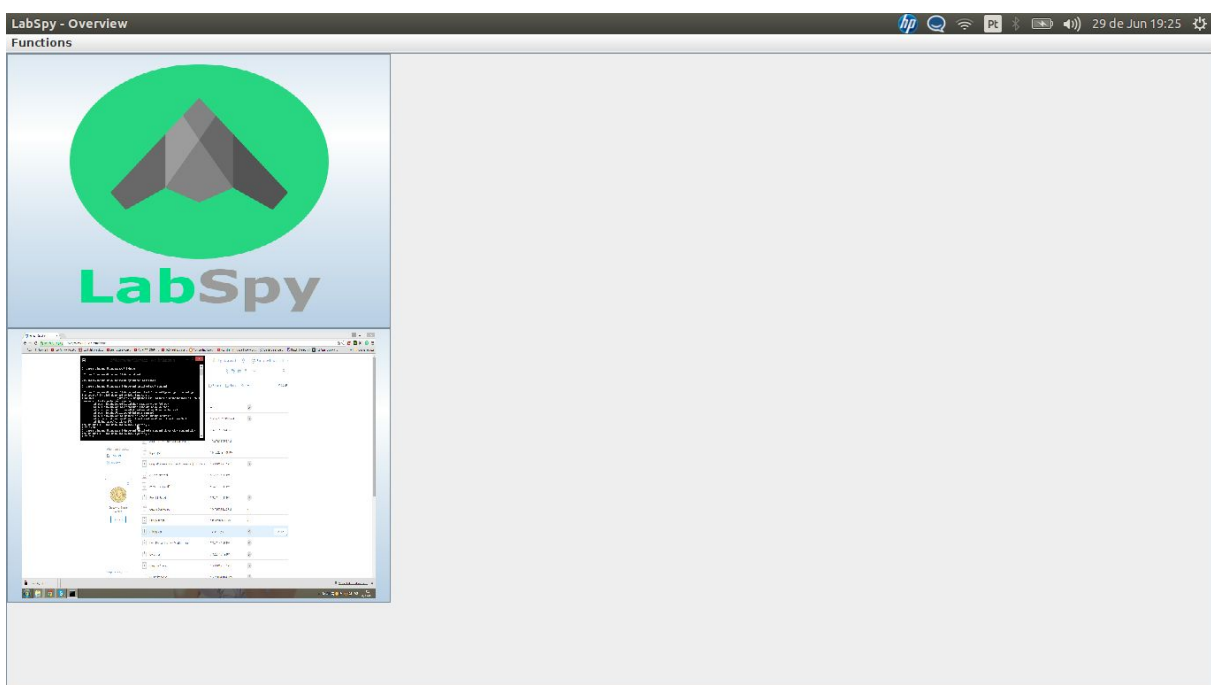
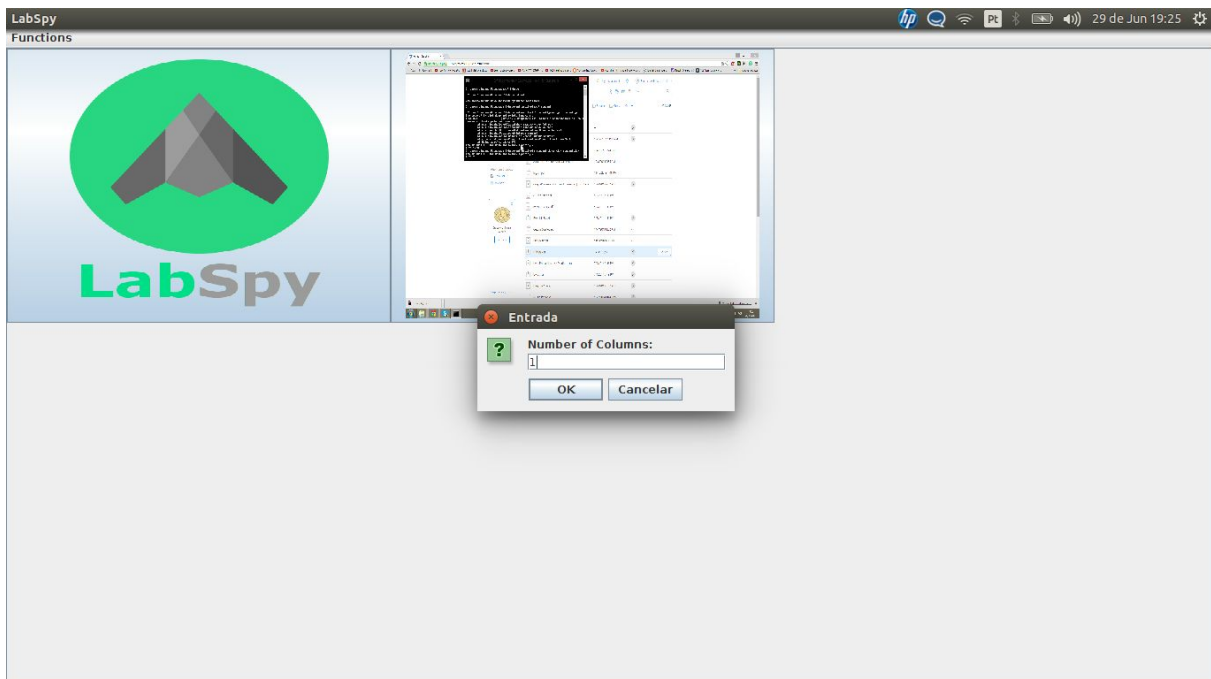
As configurações passaram a responder por ambientes, o que significa que podemos estar em um “ambiente de desenvolvimento” ou em um “ambiente de produção”, permitindo assim maior flexibilidade do programa. O ambiente de produção é o padrão (as configurações do programa são procurada em “*~/labspy/*”); para ativar o ambiente de desenvolvimento é necessário exportar uma variável de ambiente chamada LABSPY\_ENV com o valor “dev” (no Linux: `export LABSPY_ENV=dev`), o que pode ser feito diretamente através do *script* “*./dev.sh*”, que também *constrói* o projeto e cria os arquivos necessários para testar as modificações do ambiente de desenvolvimento.

## 5.6 Atualização da quantidade de frames

A partir de agora é possível atualizar a quantidade de frames de acordo com um input arbitrário. Ao abrir o “Overview” dos computadores, existe a opção “*Functions*” no menu principal. Dentre as opções possíveis temos “*FPS BigScreen*” e “*FPS Overview*”, onde a primeira fica responsável por configurar o número de frames ao controlar remotamente o computador do aluno (essa configuração modifica a taxa de quadros por segundo de todos os computadores na hora de controlar remotamente). Caso a modificação seja feita durante o controle, ou seja, na janela BigScreen, ela modificará apenas aquele computador que está sendo controlado. Já a segunda é responsável por configurar a atualização dos frames dos computadores na tela onde ficam todos os computadores dos alunos (“Overview”) - modificar essa opção muda a frequência com que os alunos mandam screenshots para o professor.

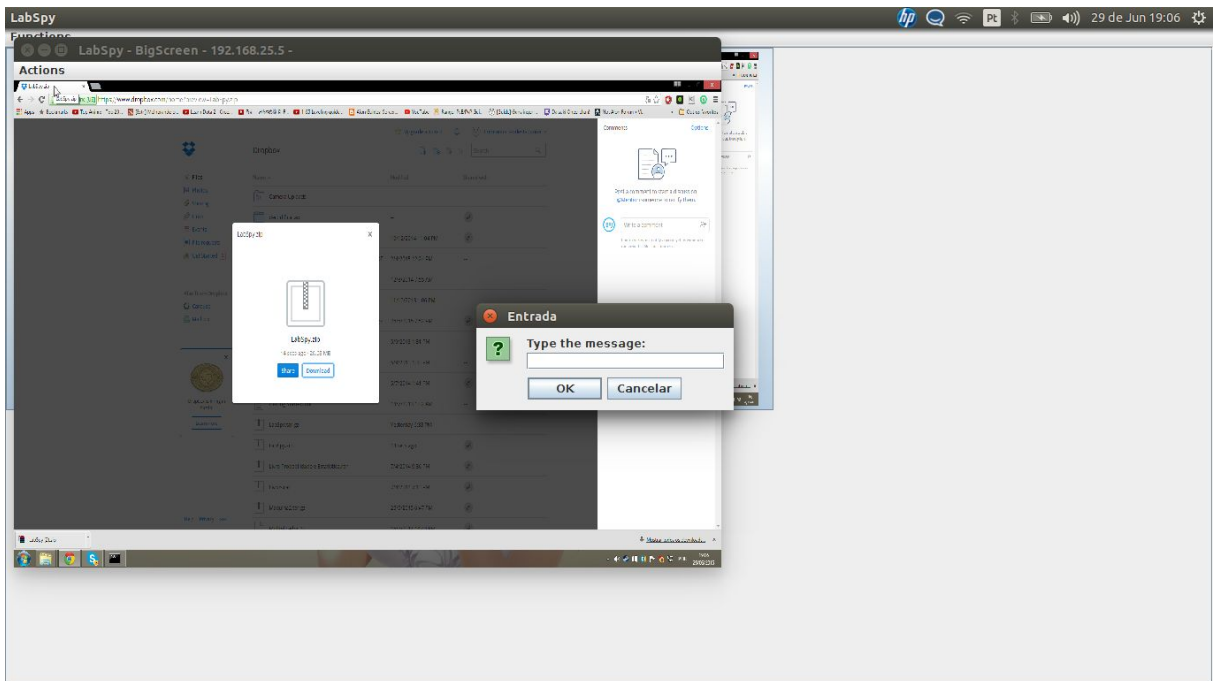
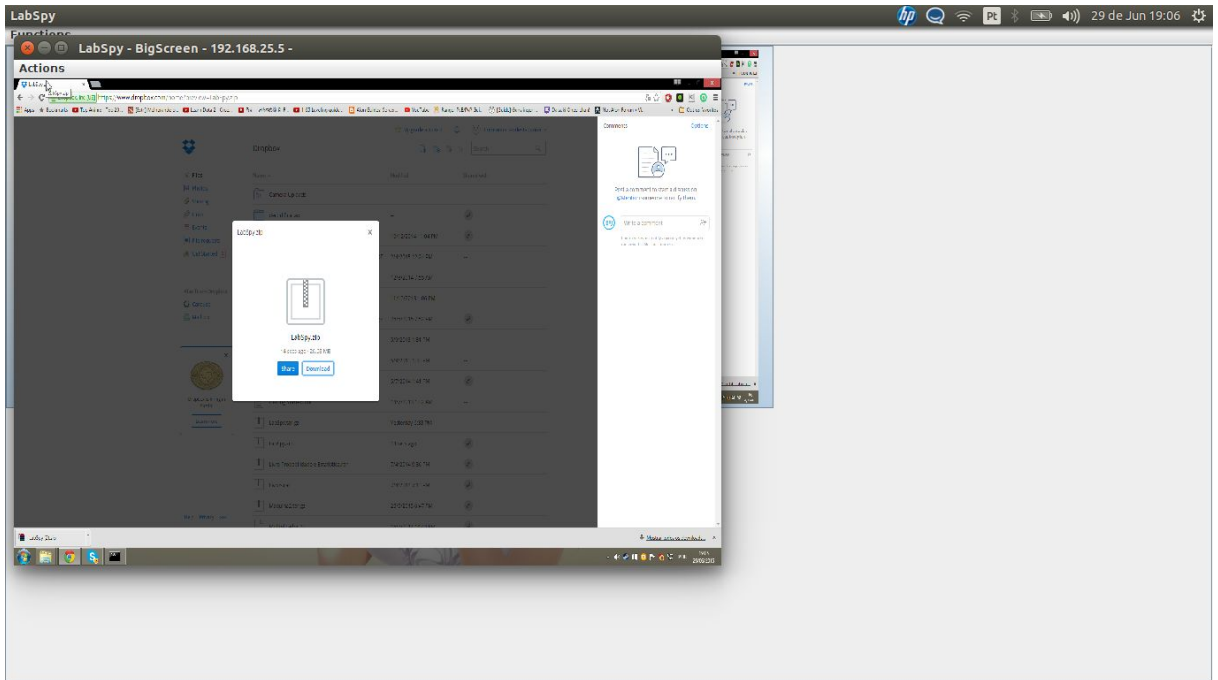
## 5.7 Atualização da quantidade de colunas

Agora é possível modificar o número de colunas que temos na janela com todos os computadores dos alunos (“Overview”). Nesta janela temos o menu “*Functions*”, onde é possível encontrar a opção de “Regrid”, que fornece a opção de modificar o número de computadores em uma linha, modificando a forma em que o Grid é organizado.

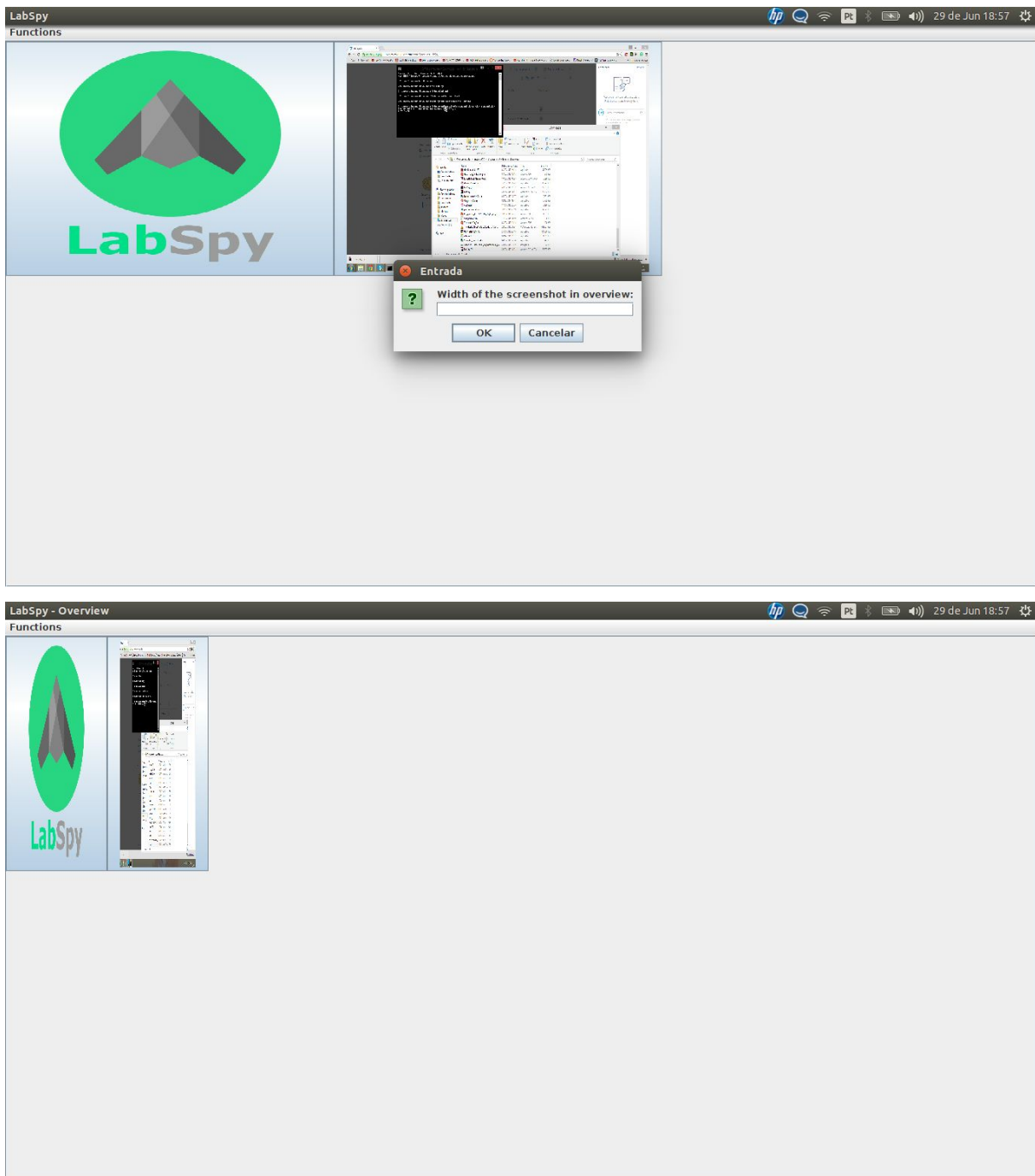


## 5.8 Enviar mensagem para o aluno

Temos a opção de enviar uma mensagem de texto arbitrária para um computador específico. Na janela de “Overview” temos um menu “Functions” que contém a opção “Send Message”, responsável por enviar para todos os computadores de estudantes conectados ao professor uma mensagem de texto. Por conseguinte, computadores que não estejam conectados não poderão receber a mensagem. Essa função também está disponível quando o professor está controlando remotamente o computador de um aluno, ou seja, através da janela “BigScreen”. A partir desta janela um professor pode enviar uma mensagem para aquele computador daquele estudante específico.







Teste de modificação do comprimento

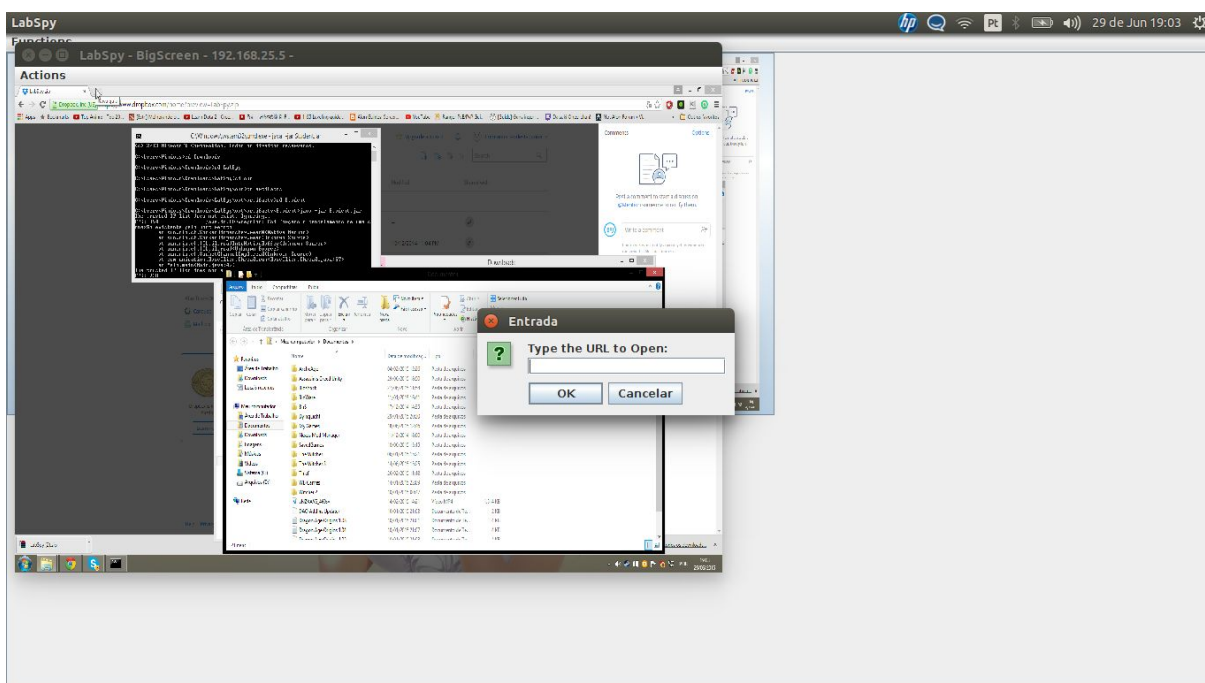
## 5.10 Bloqueio

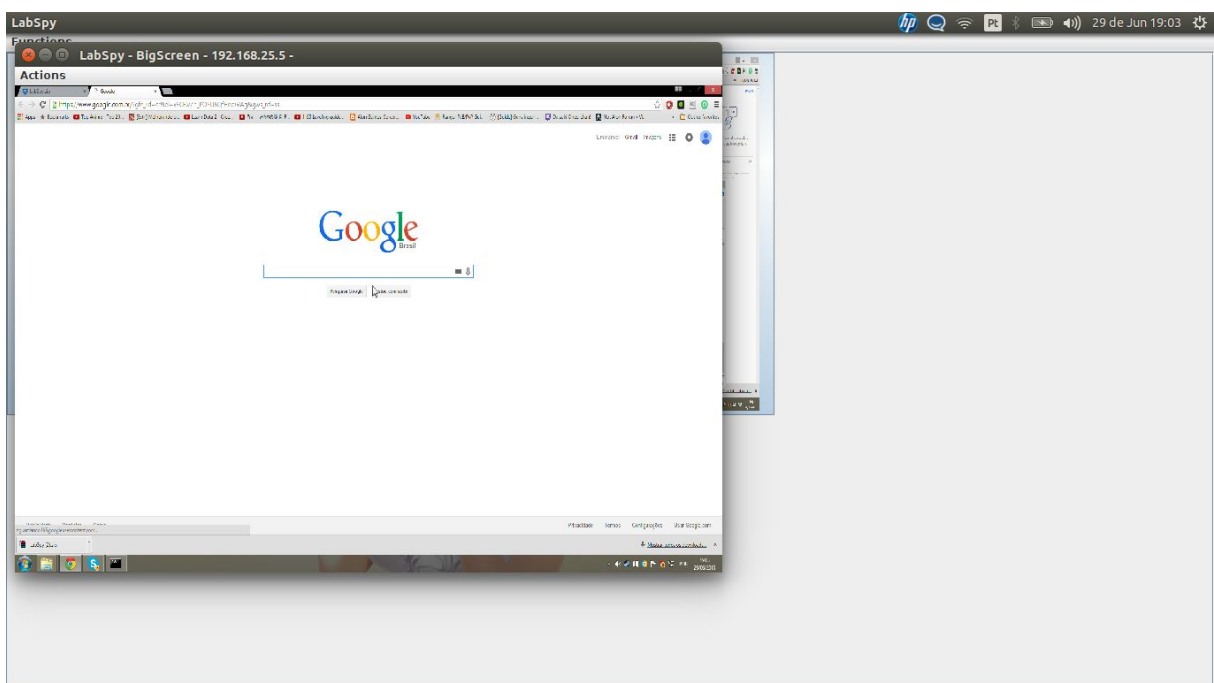
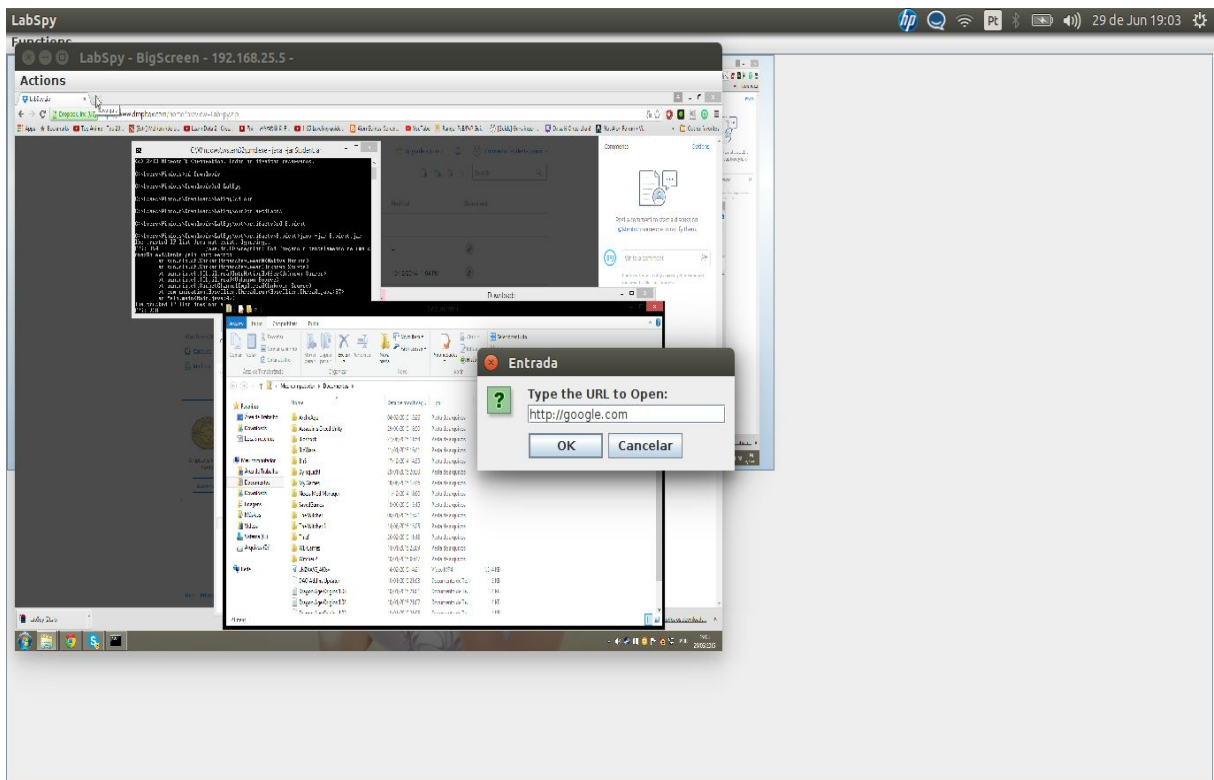
O professor tem a opção de bloquear o computador do aluno, impossibilitando-o de trabalhar no computador. É possível bloquear todos os computadores através dessa função na janela de “Overview” ou ainda bloquear apenas um computador (só que neste caso se faz necessário estar na janela BigScreen. A opção de desbloquear tem as mesmas características citadas anteriormente (serve para todos os computadores ou somente um), entretanto essa opção irá desbloquear o computador do aluno.



## 5.11 Abrir URL

Também foi adicionada a funcionalidade de enviar uma URL que será aberta no navegador de internet do computador do estudante. Dentro da janela “BigScreen”, ao selecionar a opção “Open browser in URL”, será aberto um campo para digitar o URL e, quando o usuário apertar “OK”, o site será aberto no navegador do computador do aluno. É esperado que o usuário utilize `http://` como parte da URL (nitidamente, como um prefixo). Se for necessário o usuário poderá abrir uma URL em todos os computadores conectados, selecionando a opção “Open browser in URL” na janela de “Overview”.





## 5.12 Função de Restart, Shutdown e comandos arbitrários

Nesta versão foi adicionada a funcionalidade de desligar ou reiniciar o computador do aluno remotamente, bem como de enviar comandos arbitrários para serem executados em um *shell* do computador do estudante. As funções de *restart* e *shutdown* podem ser feitas em todos os computadores de uma só vez ou apenas em computadores específicos. Para reiniciar ou desligar todos os computadores, basta ir a tela de “Overview” (onde temos visualização de todos os computadores), ir em “Functions” e então selecionada a opção desejada. Se o usuário

quiser desligar ou reiniciar apenas um computador, ele terá que entrar na janela “BigScreen” daquele computador (ou seja, clicar em computador para fazer controle remoto) para fazer tais ações, que se encontram no menu “*Actions*” da já referida janela.

Um detalhe é que é possível estender a funcionalidade a novos sistemas operacionais, porque a classe `OSCommands` chama uma instância de uma subclasse correspondente ao sistema operacional e executa os comandos próprios, por exemplo, se for no Linux, a classe `LinuxCommands` é instanciada, se for no Windows, a classe `WindowsCommands` é instanciada. Assim sendo, é possível criar novas classes correspondentes a outros sistemas operacionais não listados nesta entrega (nossa meta, até então, está no Windows e no Linux).

### 5.13 Adicionar IPs confiáveis para o aluno

Podemos definir IPs confiáveis aos quais o aluno pode se conectar, para não ocasionar problemas de que outro computador com o mesmo programa possa, também, acessar o computador do aluno. Com essa opção garantimos uma segurança para o aluno. Ao rodarmos a instalação remota para um IP, automaticamente o IP do computador que fez a instalação remota é determinado como confiável, colocando esse IP em uma pasta específica no path “`/var/lib/LabSpy/addressList`”. Para instalações locais essa função pode ser executada através do script `configura.sh`, que precisa ser executado manualmente (ou seja, é preciso configurar manualmente quais computadores são confiáveis). Contudo, esse script faz as mesmas operações que a execução automática na instalação remota.

## 6. Funcionalidades inacabadas

Tivemos várias ideias de funcionalidades para o LabSpy, no entanto, não foi possível concluir todas a tempo devido a atrasos no cronograma ou surpresas negativas durante o desenvolvimento do software (problemas teóricos, problemas não vislumbrados previamente, etc.). Veja a seguir quais foram as funcionalidades planejadas para o LabSpy que não foram concluídas para esta terceira entrega:

### 6.1 Chat

Estávamos pensando em incluir uma funcionalidade de chat, para manter contato entre o professor e o aluno. Devido às prioridades por nós estabelecidas, decidimos incluir apenas a conversação entre um aluno e o professor, e não o professor podendo se conectar com vários alunos como pensávamos inicialmente. No entanto, por problemas prioritários enfrentados durante o desenvolvimento, não foi possível que a implementação ficasse funcional em tempo hábil, de forma que as implementações dessa funcionalidade foram descartadas.

- **O que foi concluído:** Toda a interface está pronta, sendo possível vê-la instanciando um objeto da classe `Messenger` do pacote `common.chat`. Essa classe é composta pela `TextBox`, que está responsável pela caixa de texto e pelo botão de envio.

- **O que falta:** O sistema de envio de mensagens, que funcionaria da seguinte forma: através do sistema de envio de mensagens que o professor é capaz de fazer (feito nesta entrega, seção 5.8), o aluno teria a possibilidade de responder à mensagem e, ao confirmar, uma janela de chat seria aberta. A troca de mensagens seria feita pelo BaseMessage (funcionamento explicado na seção 4.3 na segunda entrega) e a extensão desta, ClientThread responsável por manter o chat funcionando.
- **Objetivos:** Com o chat, o professor poderia obter mais informações sobre o que o aluno está trabalhando no momento ou sugerir auxílio, caso note que o aluno estivesse tendo dificuldades.

## 6.2 Bloquear sites e endereços IPs

Pensamos em diversas possibilidades para realizar o bloqueio de sites e endereços IP dos computadores dos alunos e chegamos em algumas soluções, entretanto acabamos por não implementar nenhuma delas por problemas teóricos ou práticos que surgiram em cada uma das soluções possíveis. Entre as soluções pensadas, estão:

- **Arquivo “/etc/hosts”.** Essa foi uma das primeiras opções que olhamos e percebemos que ela fornecia tudo que precisávamos, pois seria fácil configurar os arquivos “/etc/hosts.allow” e “/etc/hosts.deny” [1]. Após algumas pesquisas e verificadas no código fonte descobrimos que tal arquivo está depreciado [2] e que portanto não devemos implementar, pois a qualquer momento pode deixar de funcionar (o que é algo ruim, visto que planejamos um programa que funcione a longo prazo). De qualquer forma existiam alguns problemas com essa solução, tais como não prevenir o uso dos chamados “proxies web” para acessar sites bloqueados.
- **IPTables.** Sem sombra de dúvidas é uma das soluções mais comentadas e que de certa forma substituiu o uso do arquivo de “hosts”, entretanto possui diversos problemas. Através do IPTables é possível bloquear apenas endereços IPs e não sites, o que resulta em um problema bem grande para bloquear e permitir websites distribuídos. Sites distribuídos como Facebook, todos os serviços do Google e outros sites possuem diversos endereços IPs (que costumam mudar de hora em hora), não sendo possível prever todos os endereços IPs que um site distribuído possui. Dessa forma fica difícil prever, mesmo dentro de uma instituição como a UFSC, os endereços IPs que correspondam a um serviço em específico. Através do IPTables seria possível bloquear todos os sites HTTP e HTTPS com algumas exceções, tais como em um script que fizemos para liberar todos os IPs da UFSC e bloquear todos os demais [4] (versão inicial dele, apenas). Pensamos por mais algum tempo e percebemos que isso ficava longe de resolver o problema, uma vez que só solucionava uma instância específica do problema - e ainda de forma duvidosa, pois não era apenas o Moodle que era liberado dentre todos os sites do mundo, mas todos os IPs utilizados dentro da UFSC, o que constitui um grande problema. Prever diversos endereços IP para cada site, seja para bloquear ou liberar o seu acesso (podendo esses endereços serem

atualizados em intervalos de tempo não previsíveis), não nos pareceu uma ideia convincente e boa o suficiente para ser implementada.

- **Proxy HTTP.** De longe a melhor solução, pois através de um servidor proxy HTTP escrito em Java seria possível bloquear sites não apenas de acordo com a sua URL ou com seu IP, mas também com o seu conteúdo. A implementação de tal solução não é trivial, entretanto não foi isso que nos impossibilitou a sua aplicação. Essa solução requer um *mix* entre o *proxy http* em si e o IPTables do computador de cada estudante. Basicamente o IPTables teria que redirecionar toda e qualquer conexão dos computadores dos estudantes para o *proxy http*, que analisaria se a requisição é válida e aceitaria ou negaria tal requisição. O problema desta solução é que o *proxy http* ficaria localizado no computador do professor, o que incorreria em um tráfego elevado, criando um gargalo de performance no computador do professor, que teria que tomar a decisão de todas as conexões de todos os computadores do laboratório. É possível que a rede local ficasse inclusive sobrecarregada com tanta demanda (basta imaginar uma situação corriqueira onde o professor pede aos alunos para acessar determinado site, ocasionando um alto pico de requisições acontece em um pequeno intervalo de tempo). Não conseguimos encontrar uma solução para esse problema teórico, por isso não implementamos algo nesse sentido.
- **Squid e demais alternativas.** O Squid [4] é uma ferramenta de terceiros que se mostrou muito poderosa e capaz de satisfazer alguns requisitos do nosso projeto com relação ao bloqueio de sites, entretanto também se faria necessário o uso de IPTables (o que não necessariamente constitui um problema) e por uma decisão de projeto decidimos não adotar (ou assumir) essa dependência ou alternativas similares.

## Referências

- [1] <https://jamalahmed.wordpress.com/2010/03/19/using-etchosts-allow-and-etchosts-deny-to-secure-unix/>
- [2] <http://askubuntu.com/a/23225/145168>
- [3] <https://gist.github.com/paladini/abfe36d71cf24a56cc44>
- [4] <http://www.squid-cache.org/>