Aplicação Cliente-Servidor de uma Urna Eletrônica com Socket (TCP)

Caio de Sá Lopes[1],**Igor R. A. Machado**[1], **Lais P. F. Fortes**[1], **Arthur A. Lolico**[1]

[1]Escola de Engenharia de São Carlos - Universidade de São Paulo (USP)
Caixa Postal 668 – CEP 13560-970 – São Carlos – SP – Brasil

{lais.fortes, caio.sa.lopes, igor.ramon.machado}@usp.br, arthur.lolico@gmail.com

Abstract. *This report presents the production process of the design of an Electronic Urn to represent the use of a client-server application with Socket (TCP), which is programmed by Java language. The report details the theories about the matters discussed, the main tools and environments used during its creation, as well as its modeling itself, ie the production and more detailed programming.*

Resumo. *Este relatório apresenta o processo de produção do projeto de um Urna Eletrônica para representar o uso de uma aplicação Cliente-Servidor com Socket(TCP), a qual é programada pela linguagem Java. O relatório descrimina as teorias acerca dos assuntos tratados, as principais ferramentas e ambientes utilizados durante sua criação, assim como sua modelagem propriamente dita, ou seja, a produção e programação mais detalhada.*

1. Introdução e Descrição do Problema

Este relatório tem por objetivo apresentar dados importantes acerca do projeto de uma urna eletrônica com aplicação cliente-servidor, desde sua base teórica até a real implementação da comunicação entre Cliente e Servidor por meio de Sockets e protocolo TCP. Também serão apresentadas a solução do problema proposto, metodologia, ambiente e ferramentas utilizadas, suas especificações e por fim sua utilização, resultados e devidas conclusões.

O problema inicial consiste basicamente no projeto de desenvolvimento de uma aplicação cliente-servidor de uma urna eletrônica utilizando Socket (TPC), obedecendo a alguns requisitos como: implementação em linguagem Java, a qual será dividida numa parte cliente e numa parte servidor, utilizando sockets para comunicação.

A primeira conterá a classe candidato com atributos de código de votação, nome, partido e num de votos, além de uma interfase gráfica para selecionar entre votar, deixar voto em branco ou nulo, e então carregar a lista de candidatos do servidor (com o código 999) e finalizar votações. A urna somente irá permitir votação caso já exista uma lista de candidatos previamente carregada e recebida e também só enviará os resultados com no mínimo um voto já realizado.

Na segunda parte, o lado do servidor será responsável por abrir uma thread para controlar novas conexões de acordo com o código requisitado pelo cliente, sendo as opções para abrir uma conexão e enviar a lista de candidatos para o cliente ou então abrir uma conexão para receber os votos do candidato após este ter decidido encerrar a conexão e enviar os votos.

Todos os códigos desenvolvidos se encontram disponíveis no GitHub no repositório: `https://github.com/caiolopes/client-server-urn`.

2. Fundamentação Teórica

A atual sessão deste relatório apresentará teorias acerca de termos e tópicos utilizados, dando suas devidas explicações para um melhor entendimento desse.

2.1. Urna Eletrônica

A urna eletrônica, ou máquina de votação, é um microcomputador utilizado no Brasil há 20 anos especificamente para votação eleitoral, a qual preza por portabilidade e segurança. Esta é composta pelo terminal do mesário, onde o eleitor é identificado e autorizado a vota, de modo que só possa votar uma vez, e o terminal do eleitor, onde se registra seus votos. Os votos são protegidos e embaralhados, a fim de assegurar o voto secreto.

Figura 1. Urna Eletrônica

A urna é um equipamento composto de partes mecânicas e eletrônicas e seu software é devidamente protegido; a interface do usuário possui teclado numérico e teclas simplificadas (figura 1) para registrar o voto após a confirmação pelo usuário na tela como demonstrado na figura. O projeto será baseado neste tipo de urna, com algumas modificações e simplificações necessárias. [TSE 2016]

2.2. Arquitetura Cliente-Servidor

A arquitetura cliente-servidor vem sendo desenvolvida há vários anos, o termo cliente-servidor diz respeito à distribuição de aplicações computacionais através de diversas plataformas.

Normalmente essas aplicações estão divididas entre um provedor de acesso e uma central de clientes com interface gráfica para usuários acessarem os dados.

Esta arquitetura envolve dois computadores interagindo, requisitando serviços um ao outro, o servidor permanece funcionando e respondendo aos clientes como exemplificado na figura 2. A ligação lógica existente entre cliente e servidor é baseada na comunicação entre processos, ou seja, um programa menor rodando dentro de um sistema final e nesse caso os processos se comunicam em sistemas finais diferentes a partir de trocas de mensagens pela rede. [Kurose and Ross 2010]

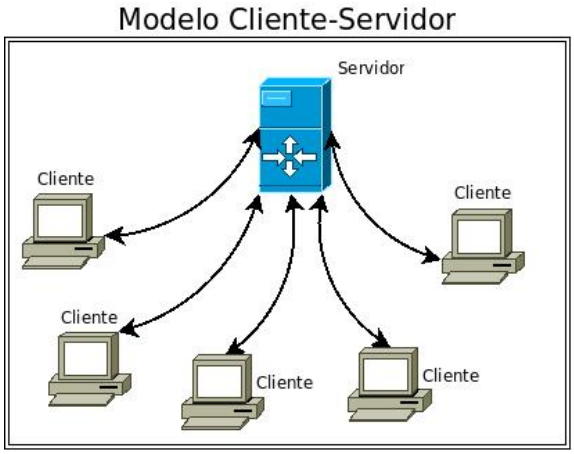

Figura 2. Arquitetura Cliente-Servidor

2.3. Socket

Uma aplicação cliente-servidor normalmente contem uma parte servidora e vários cliente, sendo assim o servidor deve processar as solicitações de todos os clientes. Em Java isso ocorre através de threads e essa conexão precisa ocorrer rapidamente. [Harold 2004]

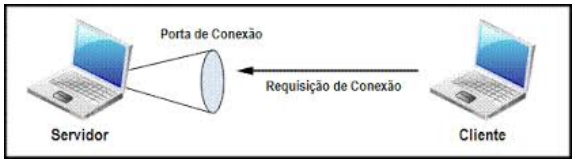

Figura 3. Conexão Socket

Como serviços diversos podem ser oferecidos pela mesma máquina esses devem ser diferenciados não só pelo endereço IP, mas também por um número da porta de conexão como demonstra a figura 3 , logo o objeto do tipo Socket tem a responsabilidade de administrar essas requisições pela rede em certa porta, sendo criado após uma conexão e mantendo a comunicação entre cliente e servidor. [Tanenbaum 2011]

2.4. TCP

O protocolo TCP (Transmission Control Protocol) foi criado designadamente para apresentar um fluxo de bytes fim a fim confiável em uma inter-rede não confiável, ou seja, um rede com partes que podem ter diferentes topologias e definições. É um protocolo mais robusto e menos suscetível a falhas.

O serviço TCP ocorre quando cliente e servidor em pontos opostos se conectam via sockets e seu número fica sendo composto pelo endereço mais o número da porta socket. Todas as conexões realizadas são ponto a ponto e full-duplex, ou seja, tráfego pode ocorrer em ambas direções e cada conexão com dois pontos terminais. O protocolo TCP se comunica através de segmentos com bytes. [Tanenbaum 2011]

3. Metodologia, ambiente e ferramentas utilizadas

Nesta sessão serão apresentadas as ferramentas e os ambientes utilizados no processo de criação deste projeto, assim como uma metodologia mais detalhada.

3.1. IntelliJ IDEA

O projeto da Urna Eletrônica foi produzido utilizando-se a IntelliJ IDEA, ou seja, um ambiente integrado de desenvolvimento em Java (IDE), o qual proporcionava uma unanimidade entre os dois módulos, sem discrepâncias devido a programação em diferentes ambientes ou sistemas operacionais, dessa maneira módulo cliente (cliente) e módulo servidor (server) se comunicariam com mais facilidade, não apresentando problemas devido à utilização de diferentes ambientes.

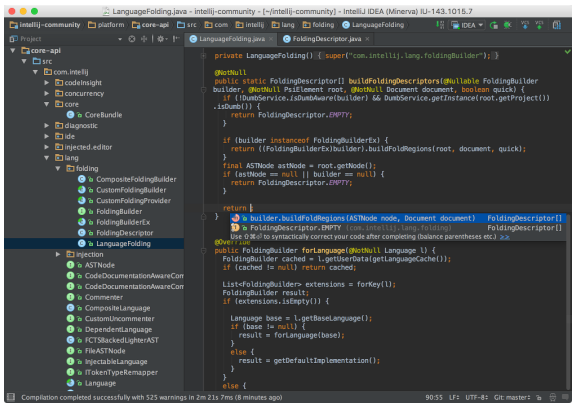

Figura 4. IntelliJ IDEA

Esta IDE é desenvolvida pela JetBrains e existe a cerca de 15 anos, pode ser feito seu download na versão community de forma gratuita ou a ultimate com mais funcionalidades, porém não gratuita. Ambas as versões são como uma assistência de programação eficiente e com diversas funções para auxiliar o programador e suporta diversas ferramentas e estruturas como: Git, JavaScript, Desenvolvimento Android e Web, entre outras. Sua utilização promete desenvolver o máximo de produtividade ao programador, assim como um design (figura 4) mais ergonômico para análise de código para uma experiência mais apreciável. [JetBrains 2016]

3.2. JSON

Um grande problema encontrado durante a produção do projeto da Urna, foi como seria a comunicação entre os módulos do cliente e do servidor, duas aplicações sendo programadas em Java que deveriam se comunicar, o modo encontrado para enviar essas mensagens foi via JSON, pois seria mais flexível que a serialização Java.

JSON (JavaScript Objetct Notation – Notação de Objetos JacaScript) é uma forma de troca de dados leve, de fácil entendimento para pessoas e máquinas, pois acopla todas informações em uma linha, ordenada de maneira simples, com pares de nome em forma de uma hashtable e lista ordenada de valores, ou um vetor, dessa forma padronizada os módulos conseguem se comunicar via essas mensagens do JSON de forma simplificada e flexível [JSON 2016].

3.2.1. GSON

GSON é uma biblioteca de serialização e deserialização que permite converter objetos Java para JSON e JSON para objetos Java.[GSON 2016]. Foi utilizado no nosso trabalho

ao invés da serialização Java padrão (atráves da classe *java.io.Serializable*), pela facilidade, praticidade e escalabilidade que permite como que não apenas seja comunicado de Java para Java, mas para qualquer aplicação independente do formato, além de JSON ser um formato amplamente difundido nos dias de hoje.

3.3. GitHub

Durante a produção do projeto a codificação foi feita com o auxílio dos serviços oferecidos pelo GitHub, para que desta maneira todos programadores do projeto pudessem participar ativamente da codificação de forma compartilhada.

Figura 5. Logo GitHub

O GitHub é um serviço web que traz funcionalidades diversificadas para os programadores, os quais podem, de forma gratuita (também existe opção não gratuita), hospedar seus projetos pessoais e também pode compartilha-los com outros programadores para que estes possam contribuir simultaneamente, e ele faz o controle de modo que todas edições e criações no código possam existir sem alterar outras ou subscreve-las, ocorrendo somente com a permissão do dono. Também há a possibilidade de conectá-lo a diversos outros ambiente e por fim servir como um portfolio do programador. [GitHub 2016]

3.4. JavaFX

JavaFX é uma plataforma de software desenvolvida pela Oracle baseada em Java para a criação e desenvolvimento de aplicações de Desktop que pode rodar em diversos dispositivos diferentes como desktop, browser ou dispositivos móveis. Ela foi criada para substituir o Swing como a biblioteca gráfica padrão GUI, de modo mais eficiente e com mais funcionalidades.

No projeto da Urna Eletrônica foi utilizada a JavaFX par auxiliar a implementação da interface gráfica da Urna, com seus botões, tela e caixas de avisos interativos, podendo obter desta maneira uma interface mais agradável e simples de usar para o usuário, o qual ficará no papel do cliente para a votação dos candidatos.

3.5. LaTeX

Por fim, a composição deste relatório foi criada com o auxílio da ferramenta LaTeX, uma ferramenta muito utilizada em acervos científicos e na produção de textos matemáticos pela sua alta qualidade tipográfica, é um software gratuito e também disponível online, sem necessidade de baixar, como shareLatex, utilizado neste relatório. Seus benefícios vão além de só se preocupar com o texto, enquanto a formatação fica pré-padronizada, mas também a flexibilidade que da ao escritor para futuras modificações e referências, entre outros benefícios. [LaTeX 2015]

4. Modelagem do Sistema

Esta sessão irá apresentar como a solução para a criação da urna eletrônica foi pensada e colocada em prática, bem como apresentar a modelagem do sistema propriamente dita, considerando aspectos importantes que devem ser explorados acerca de sua programação.

4.1. Classe Candidato

Como especificado no trabalho, era necessário utilizar uma classe candidato com os seguintes atributos:

- inteiro código de votação;
- string nome do candidato;
- string partido;
- inteiro número de votos.

Foi definido um modelo em JSON que posteriormente foi convertido para o objeto Java correspondente (classe *Candidate.java*) tanto no cliente como no servidor. Um exemplo do modelo se segue no código 4.1.

```
1 {
2     "code":0,
3     "name":"Lizzie Lynch",
4     "politicalParty":"DEM",
5     "votes":0
6 }
```

Código 1: Exemplo de um modelo de candidato em JSON.

Portanto, o trabalho contemplou a especificação de utilizar uma classe Candidato.

4.2. Servidor

Para implementar o servidor, foi necessário avaliar quais seriam as funcionalidades do sistema. O servidor deve ser capaz de realizar duas tarefas, a primeira consiste em enviar a lista dos candidatos para o cliente, enquanto que a segunda consiste em receber os votos de um cliente ao final das votações. Como o servidor deve ser capaz de receber diversas requisições concorrentemente, cada nova requisição deve ser atendida por uma thread. Além de uma classe para os candidatos, existem três classes para o servidor: Server, CandidatesThread e VotingThread.

A classe Server é a classe responsável por receber as requisições dos clientes. Como foi dito anteriormente, o servidor deve ser capaz de responder a dois tipos de requisição, cada uma delas abre uma thread. Caso o cliente peça pela lista de candidatos, a classe Server abre uma thread do tipo CandidatesThread. Da mesma forma, caso a requisição seja para que o cliente envie os dados da votação, uma thread do tipo VotingThread é criada para atende-la.

A classe CandidatesThread é uma classe simples, responsável por enviar o arquivo que possui a lista de candidatos para o cliente. A principal característica a se destacar

nessa classe é a forma com que a lista é enviada. Para facilitar o envio, e a fim de aproveitar o formato de arquivo utilizado (JSON), os dados são convertidos para um única string, para somente então ser enviada.

Por fim, a classe VotingThread é a classe responsável por receber os votos e atribui-los aos candidatos corretos, somando os votos totais. Assim como a classe CandidatesThread converte os dados do arquivo para uma string antes de enviar, a classe VotingThread deve converter a string recebida em formato JSON de volta para um objeto Java. Desta forma, os votos podem ser somados sem grandes dificuldades.

Conforme o servidor recebe os votos advindos dos cliente, é possível acompanhar as apurações através de um arquivo. Tal arquivo, denominado ”results.data”, é atualizado logo que um cliente envia seus dados para o servidor, e para acompanhar o andamento das votações, basta abri-lo no terminal utilizando o comando ”cat results.data”, além deste também imprimir no terminal.

É interessante comentar que houve um cuidado em relação a possíveis concorrências de Threads, através de métodos syncronized do Java.

4.3. Cliente

Diferentemente do servidor, o cliente deve ser capaz de realizar três tarefas. A primeira tarefa consiste em pedir ao servidor a lista com os candidatos, deixando-os aptos a receber votos. A segunda tarefa consiste na votação propriamente dita, na qual o cliente acumula todos os votos do período na qual a urna ficou disponível, desde que recolheu a lista de candidatos. Por fim, o cliente deve ser capaz de enviar os votos obtidos ao encerrar as votações.

A estrutura do cliente se difere bastante daquela utilizada no servidor. O principal motivo para tal consiste no fato de que enquanto o servidor deve somente lidar com os dados e processamento (back-end), o cliente deve lidar de forma predominante com a interface ao usuário (front-end), além de um certo processamento e formatação dos dados em menor escala. De forma geral, o cliente é simples pelo fato de possuir apenas uma tela principal com as opções de escolher na lista um candidato, votar no mesmo, votar em branco ou em nulo, janelas de alerta e janelas de confirmações. A dificuldade fica por conta de todo o tratamento de eventos inerentes a uma programação com interface gráfica.

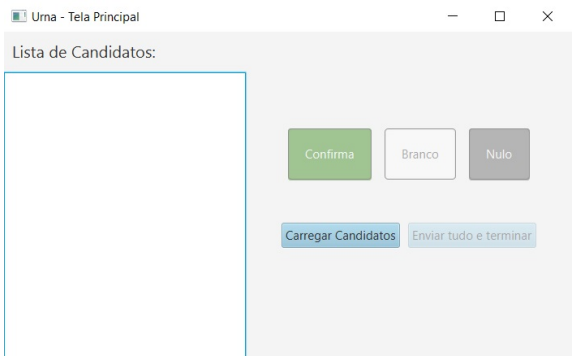

Figura 6. Interface da Urna

O lado cliente possui o seguinte menu, como mostra a 6:

- Votar (através da tecla confirma);
- Votar em branco;
- Votar nulo;
- Carregar lista de candidatos do servidor (opcode 999);
- Enviar tudo e terminar: envia os resultados pro servidor e encerra sessão de votos.

4.3.1. Restrições

Existiu os seguintes cuidados, como especificado pelo trabalho:

- permitir apenas votações caso a lista de candidatos já tivesse sido carregada;
- permitir que apenas fosse enviado os resultados da urna ao servidor caso houvesse no mínimo uma votação realizada.

  Dessa forma, o trabalho contemplou as restrições necessárias da especificação.

4.3.2. Arquitetura do cliente

Para a modelagem do cliente, utilizou-se o padrão de arquitetura MVC (Model-View-Controller), o que foi possível graças ao uso do FXML, que é uma linguagem baseada no XML que provê uma estrutura para construir a interface gráfica separada da lógica da aplicação.

Este padrão permite quebrar a aplicação em três partes. A primeira parte, composta pelo controlador (Controller), é responsável por interpretar as entradas de forma a mapear as ações do usuário, resultando em comandos que serão enviados ao modelo (Model), assim como interagir com a classe $Network$ responsável por interagir com o servidor. O modelo, por sua vez, compõe a segunda parte da aplicação, composta apenas pelo modelo do Candidato.

5. Conclusão

Ao final deste trabalho, obteve-se um sistema completo de uma urna eletrônica, composta de um servidor e de um cliente, ambas implementadas em linguagem Java. Enquanto que o lado do servidor ficou responsável por controlar as novas conexões, bem como por responder ao cliente o que lhe foi requisitado, o lado do cliente, por sua vez, ficou responsável por receber votos dos eleitores e enviá-los ao servidor ao término das votações.

A troca de dados entre o cliente e o servidor foi feita utilizando-se da notação JSON por se mostrar uma solução mais flexível quando comparada com a serialização Java. Desta forma, os dados são enviados em uma string composta por apenas uma linha mas com todos os dados pertinentes, além de permitir uma fácil conversão de string para uma objeto Java.

Também foi adicionada um interface gráfica intuitiva ao usuário, como componente extra do projeto de modo a ser uma composição mais apresentável.

Por fim, é possível afirmar que o projeto foi concluído com sucesso de acordo com os requisitos estabelecidos, assim como alguns adicionais de ferramentas e interface extra, conseguindo desta maneira demonstrar com perfeição o funcionamento de uma aplicação cliente-servidor com socket (TCP) e seus conhecimentos necessários.

Referências

[GitHub 2016] GitHub (2016). About github. https://github.com/. Acessado: 25-06-2016.

[GSON 2016] GSON, G. (2016). Gson. https://github.com/google/gson. Acessado: 26-06-2016.

[Harold 2004] Harold, E. R. (2004). *Java network programming*. O'REILLY, 3th edition.

[JetBrains 2016] JetBrains, I. (2016). Intellij idea. https://www.jetbrains.com/idea/. Acessado: 25-06-2016.

[JSON 2016] JSON (2016). Introducing json. http://www.json.org. Acessado: 25-06-2016.

[Kurose and Ross 2010] Kurose, J. F. and Ross, W. (2010). *Redes de Computadores e a Internet: Uma nova abordagem*. Pearson, 5th edition.

[LaTeX 2015] LaTeX (2015). An introdution to latex. https://latex-project.org/intro.html. Acessado: 25-06-2016.

[Tanenbaum 2011] Tanenbaum, A. S. (2011). *Redes de Computadores*. Pearson, 5th edition.

[TSE 2016] TSE (2016). Tribunal eleitoral superior: Urna eletônica. http://www.tse.jus.br/eleicoes/biometria-e-urna-eletronica/urna-eletronica. Acessado: 25-06-2016.