Universidade Federal do Piauí – UFPI
Campus Senador Helvídio Nunes de Barros – CSHNB
Curso de Bacharelado em Sistemas de Informação

José Mário Alves Monteiro da Silva

Ourdown: Controle de Banda nas Redes Locais usando componentes Colaborativos *Peer-to-Peer*

Picos-PI
2013

José Mário Alves Monteiro da Silva

Ourdown: Controle de Banda nas Redes Locais usando componentes Colaborativos *Peer-to-Peer*

Trabalho de Conclusão de Curso apresentado no Curso de Bacharelado em Sistemas de Informação Campus Senador Helvídio Nunes de Barros da Universidade Federal do Piauí como parte dos requisitos para obtenção do Grau de Bacharel, sob orientação do Prof. M.Sc. Rayner Gomes de Sousa.

Picos-PI

2013

Eu, **José Mário Alves Monteiro da Silva**, abaixo identificado(a) como autor(a), autorizo a biblioteca da Universidade Federal do Piauí a divulgar, gratuitamente, sem ressarcimento de direitos autorais, o texto integral da publicação abaixo discriminada, de minha autoria, em seu site, em formato PDF, para fins de leitura e/ou impressão, a partir da data de hoje.

Picos-PI 23 de setembro de 2013.

José Mário Alves Monteiro da Silva

Assinatura

FICHA CATALOGRÁFICA
Serviço de Processamento Técnico da Universidade Federal do Piauí
Biblioteca José Albano de Macêdo

S586o Silva, José Mário Alves Monteiro da.
Ourdown: controle de banda nas redes locais usando componentes colaborativos peer-to-peer / José Mário Alves Monteiro da Silva. – 2013.
CD-ROM : il. ; 4 ¾ pol. (78 p.)

Monografia(Bacharelado em Sistemas de Informação) – Universidade Federal do Piauí. Picos-PI, 2013.
Orientador(A): Prof. Msc. Rayner Gomes Sousa

1. Compartilhamento. 2. Dowload. 3. Ourdown. I. Título.

CDD 004.68

José Mário Alves Monteiro da Silva

Ourdown: Controle de Banda nas Redes Locais usando componentes Colaborativos *Peer-to-Peer*

Trabalho de Conclusão de Curso apresentado no Curso de Bacharelado em Sistemas de Informação Campus Senador Helvídio Nunes de Barros da Universidade Federal do Piauí como parte dos requisitos para obtenção do Grau de Bacharel, sob orientação do Prof. M.Sc. Rayner Gomes de Sousa.

Data de Aprovação:

16/09/2013

Prof. Msc. Rayner Gomes Sousa _____ UFPI

Prof. Ivenilton Alexandre de Souza Moura _____ UFPI

Prof. Romuere Rodrigues Veloso e Silva _____ UFPI

Picos-PI

2013

A Deus por tudo que me proporciona na vida. A minha mãe e meu pai, os quais amo muito, pelo exemplo de vida e família. Aos meus irmãos por tudo que me ajudaram até hoje. Não conquistaria nada se não estivessem ao meu lado. Enfim a todos que de alguma forma tornaram este caminho mais fácil de ser percorrido.

Hoje, vivo uma realidade que parece um sonho, mas foi preciso muito esforço, determinação, paciência, perseverança, ousadia e maleabilidade para chegar até aqui, e nada disso eu conseguiria sozinho. Minha eterna gratidão a todos aqueles que colaboraram para que este sonho pudesse ser concretizado.

Grato a Deus pelo dom da vida, pelo seu amor infinito, sem Ele nada sou. Agradeço aos meus pais, José Nilton e Maria do Ó, meus maiores exemplos. Obrigado por cada incentivo e orientação, pelas orações em meu favor, pela preocupação para que eu estivesse sempre andando pelo caminho correto. Aos meus irmãos, Helbert e Vanessa, por todo amor e carinho. A minha noiva, Raquel Bezerra, que também desde o início do curso sempre me apoiou e ajudou quando mais precisei.

Aos professores do Curso de Sistemas de Informação pela contribuição na minha vida acadêmica e por tanta influência na minha futura vida profissional. Aos meus amigos por todo apoio e cumplicidade. Porque mesmo quando distantes, estavam presentes em minha vida. Obrigado a todos que, mesmo não estando citados aqui, tanto contribuíram para a conclusão desta etapa e para eu me tornar o homem que sou hoje.

“O tempo é limitado, então não gastes seu tempo vivendo a vida de outro. Não fiques preso no dogma que é viver como os outros pensam que deverias viver. Não deixe que as opiniões dos outros calem sua voz interior. E o mais importante, tenha coragem para fazer aquilo que manda seu coração e intuição.”

(Steve Jobs)

Resumo

Uma das atividades que mais consomem banda de rede é o *download* de arquivos. Realizar *download* é uma atividade comum, pois através da *Internet* há uma infinidade de arquivos, programas, músicas, vídeos e outros disponíveis. Esta atividade extingue meios físicos de distribuição e se tornou uma das principais vias para distribuição de arquivos digitais. Apesar da facilidade da realização de *downloads* no ambiente *WEB*, para os administradores de rede se apresenta como um problema. São notáveis vários *downloads* dos mesmos arquivos sendo descarregados no enlace de rede, consumindo recursos desnecessariamente. Este trabalho apresenta uma nova forma de realizar *downloads*, baseando-se em técnicas para gerenciamento colaborativo entre vários componentes de rede dotados do *software* Ourdown. A principal função do *software* Ourdown é a extinção dos *downloads* redundantes, ou seja, a transferência de arquivos semelhantes já baixados ou em andamento. O Ourdown foi desenvolvido na linguagem Java, portanto funciona em qualquer plataforma. Ele é composto de vários elementos conscientes e desenvolvidos e implementados no modelo *Peer-to-Peer* (P2P) que trabalham de forma colaborativa o compartilhamento de recursos por meio de troca de informações sobre os *downloads* requisitados em toda rede. Por fim, este projeto compara o Ourdown com soluções já existentes e ainda descreve todas as etapas desde a concepção, modelagem, desenvolvimento e testes.

Palavras-chave: Compartilhamento, *Download*, Ourdown, *Peer-to-Peer*

Abstract

One of the activities that consume more network bandwidth is downloading files. Perform download is a common activity, because the Internet is a plethora of files, programs, music, videos and other available, this activity extinguishes physical media distribution and became a major route for distribution of digital files. Despite the ease of performing downloads in a web environment for network administrators is presents as a problem because they are several notable downloads of the same files being downloaded in the link network, consuming resources unnecessarily. This paper presents a new way of performing downloads, relying on techniques for collaborative management between network components endowed Ourdown software. The main function of the software is Ourdown extinction of downloads redundant, other words, the transfer of files downloaded or similar already in progress. The Ourdown was developed in Java, so it works on any platform. It is composed of several elements conscious and developed and implemented in the model Peer-to-Peer (P2P) working collaboratively to share resources through the exchange of information on downloads required throughout the network. Finally, this project compares Ourdown with existing solutions and also describes all the stages from design, modeling, development and testing.

Keywords: *Share, Download, Ourdown, Peer-to-Peer*

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

Segundo previsões da empresa tecnológica Cisco *Systems*, o tráfego total de dados da *Internet* em 2016 será quatro vezes maior que o atual e superará pela primeira vez a casa do *zettabyte* (medida equivalente a um sextilhão de *bytes*), graças ao aumento de *uploads* de vídeos. Estima-se que nesse mesmo ano terão circulado mais dados do que a soma de informação circulada na *Internet* entre 1984 e 2012. Entre 2015 e 2016 o aumento de dados trafegados será de 330 *exabytes* (um *exabyte* equivale a um quintilhão de *bytes*) valor próximo à quantidade de tráfego gerado durante o ano de 2011, que foi de 369 *exabytes*.

Com o rápido crescimento da *Internet*, compartilhar arquivos torna-se cada vez mais rápido e prático. Comumente organizações disponibilizam arquivos digitais nas suas páginas *web*, até porque é uma maneira para que as pessoas que acessam possam conhecer a organização, o trabalho dela, os seus serviços. Por exemplo, um grupo musical pode divulgar suas músicas para seus fãs sem precisar utilizar mídias graváveis.

Antes de ser possível o compartilhamento pelos meios atuais, em ambiente WWW (*World Wide Web*), o principal meio de compartilhamento de arquivos foi o serviço FTP (*File Transfer Protocol*) que viabilizava o compartilhamento de arquivos na *Internet* permitindo que o usuário transfira, renomeie, mova ou remova arquivos remotos. Para organizações onde o tráfego da rede precisa ser controlado, *downloads* tornam-se restritos e os arquivos são disponibilizados em um servidor FTP local, onde administradores de rede executam uma conexão ao servidor e armazenam os arquivos necessários para serem disponibilizados para os usuários na rede local.

O FTP foi criado em uma época a qual usava-se a *Internet* com a disponibilidade de poucos serviços e acessos de usuários, e a tarefa de *download* era pouco realizada. Em dias atuais, gerenciar o tráfego torna-se uma tarefa minuciosa devido a três aspectos: (i) Nova geração de usuários com acesso irrestrito; (ii) Liberdade dos usuários em utilizar a *Internet*; (iii) Pouco conhecimento sobre características do "conjunto" que concedem o acesso à *Internet*.

A maioria dos *downloads* são executados através de *browsers*, ou simplesmente navegadores *web*. Pela facilidade de encontrar um conteúdo desejado e pela praticidade que se tem de transferir-lo para a sua máquina, muitas vezes o usuário não está ciente das condições da rede local. O perfil de cada usuário não generaliza o comportamento de todos os usuários daquela rede local. Todavia, as rede locais (LAN – *Local Area Network*) possuem uma taxa de trans-

missão bem maior do que as redes externas (WAN – *Wide Area Network*) o *download* de um mesmo arquivo torna-se um problema, reduzindo a quantidade de banda disponível.

É muito difícil para um administrador de rede monitorar *downloads* sem saber a natureza dos arquivos transferidos, principalmente se por um acaso uma réplica do arquivo é transferido ao mesmo tempo que o outro. Os servidores FTP necessitam de uma supervisão rígida e centralizada. Os *softwares* gerenciadores de *download* não apresentam transparência nos trabalhos realizados, pois numa rede local, são restritos às máquinas individuais dos usuários. Evitar réplicas de *download*, é de grande importância, pois a transferência de arquivos representam perda de banda, e vale ressaltar que eles concorrem e consomem recursos de rede: *buffers* de recepção, *buffers* de transmissão, *links* de dados, processador e memória de roteadores e comutadores de rede.

O presente trabalho, irá apresentar todos os processos de desenvolvimento de um *software* distribuído que juntamente com o navegador *web*, Firefox, gerenciará *downloads* dentro de uma rede local. Afim de aperfeiçoar os enlaces, o *software* trabalhará de tal forma a compartilhar recursos e trocar informações, com outras instâncias do mesmo, presente em outras máquinas na rede de forma colaborativa, monitorando as solicitações de *download* para que não haja réplicas de arquivos sendo transferidos.

Este trabalho está organizado em mais cinco capítulos além desta introdução. O Capítulo 2 aborda os principais conceitos da área da pesquisa. O Capítulo 3 é constituido de diagramas de Caso de uso, de Classe e de Sequência que permitem a compreensão do modelo concebido. O Capítulo 4 apresenta a construção do modelo, destacando as tecnologias utilizadas, infraestrutura de rede utilizada, a codificação e desenvolvimento. No Capítulo 5 serão apresentados os testes realizados com a aplicação, assim como também o ambiente em que foram realizados, os equipamentos e recursos utilizados. Por fim, no Capítulo 6, encontra-se as considerações finais do projeto destacando o conhecimento adquirido durante o desenvolvimento, as dificuldades e melhorias que podem ser implementadas posteriormente.

2 Compartilhamento de Arquivos

Com o avanço tecnológico que propicia o uso da *Internet*, muitas ferramentas e serviços emergem facilitando a vida de quem depende dela no seu dia-a-dia. Com a *Internet* é possível comunicar-se com outras pessoas, realizar trabalho, obter distração e informação, além de disponibilizar seus arquivos de diversificadas mídias (áudio, vídeo, documentos, etc.) para que outras pessoas tenham acesso.

O compartilhamento de arquivos na *Internet* é uma das atividades mais comuns que os usuários realizam hoje. Na era dos *mainframes* e dos primeiros computadores pessoais, já se fazia compartilhamento de arquivos em redes locais. Desde então com o surgimento da *Internet*, tornava-se possível que os computadores acessassem arquivos remotos utilizando o sistema de arquivos virtual, tais como: *Bulletin Board System* (1978), o USENET (*Unix User Network*, 1979) e servidores FTP (*File Transfer Protocol*, 1985) e a comunicação através de *chats* que era possível com o uso do IRC (*Internet Relay Chat*, 1988) e do Hotline (1997) que também permitiam a troca de arquivos entre usuários.

2.1 *Internet*

Desenvolvida há quase cinco décadas, a *Internet* foi incialmente desenvolvida para fins militares. O projeto foi financiado pelo Departamento de Defesa dos Estados Unidos durante a Guerra Fria, era o meio de os militares manterem a comunicação caso os meios de comunicações convencionais fossem destruídos.

Nas décadas de 70 e 80, além do uso militar, a *Internet* tinha por finalidade conectar sistemas de cerca de uma dúzia de universidades e organizações voltadas para a pesquisa científica. No início da década de 90, o engenheiro inglês Tim Bernes-Lee desenvolveu a *World Wide Web* dotando-se do uso de interface gráfica e sites dinâmicos, sendo tolerante para que a *Internet* expandisse à proporções globais. Em meio a expansão, surgiam os navegadores *web*, permitindo a interação do usuário com a *Internet*.

A medida que a *Internet* crescia, provedores de acesso e serviços surgiam desencadeando novas formas de se utilizar a *Internet*. Contemporaneamente utiliza-se a *Internet* para localização, comunicação, fonte de pesquisa, entretenimento, trabalho e etc. Através dela o usuário obtém exposição mundial que favorece diversificadas vantagens. Ele pode comunicar-se

com algum parente ou amigo distante, obter ou compartilhar informação e notícias de qualquer lugar do mundo em tempo real, divertir-se com *games* online, divulgar serviços ou produtos de empresas, comprar ou vender, assistir vídeos, ouvir música, obter ou compartilhar arquivos multimídia. A partir disso, podemos observar que a *Internet* torna-se um dos principais meios de comunicação que existem.

2.2 FTP (*File Transfer Protocol*)

É de grande necessidade compartilhar informações de forma ágil para viabilizar uma melhor performance nas tarefas desenvolvidas em conjunto por usuários de computador e *Internet*. Anteriormente ao ambiente WWW de compartilhamento, o FTP destacava-se sendo o principal meio que permitia a troca de arquivos na *Internet*.

O FTP, segundo SOARES (1995), viabiliza o compartilhamento de arquivos na Internet, permitindo que um usuário em um computador transfira, renomeie ou remova arquivos remotos. O serviço FTP foi criado numa época que o uso da *Internet* era limitado a poucos serviços.

Caracterizado como uma forma bastante rápida e versátil de transferir arquivos, o FTP baseia-se no TCP (*Transmission Control Protocol*), sendo padrão da pilha TCP/IP para transferir arquivos. Todavia, independe de *hardware* e sistema operacional.

O uso do FTP conta com restrições de acesso e propriedades de cada usuário, onde este estabelece uma conexão – conexão de controle – com o servidor através da porta 21, onde que durante o acesso ela permanecerá aberta. Posteriormente outra conexão – conexão de dados – é estabelecida entre a porta 20 do servidor e alguma outra no cliente, porta está definida durante a conexão de controle para a transferência de arquivos.

A conexão de controle citada anteriormente, manipula a sessão iniciada pelo usuário que por sua vez irá fornecer identificação, senha e comandos ao servidor. O serviço FTP pode ser executado em dois modos: passivo e ativo. No modo ativo, o cliente fornecerá na requisição de uma conexão o endereço IP e a porta na qual ele irá escutar e o servidor iniciará a conexão TCP com o cliente. Já no modo passivo, que é quando o cliente se encontra por trás de um *firewall* ou inapto, o cliente envia o comando PASV e recebe um endereço IP e um número de porta para abrir uma conexão com o servidor.

Na transferência, quatro representações de dados podem ser usadas:

- Modo ASCII: converte a representação de caracteres do *host* para 8 *bits* em ASCII e depois é desconvertido.
- Modo Imagem: o emissor envia o arquivo *byte* por *byte*, e o receptor irá armazenar o fluxo de *bytes*.
- Modo EBCDIC: utilizado para texto simples entre *hosts*.

- Modo Local: permitirá que duas máquinas enviem dados em um formato proprietário sem precisar de conversão.

Além das representações dos dados, a transferência pode ser feita de três modos:

- Modo Fluxo: um dado é enviado em fluxo continuo, sendo processado pelo TCP.
- Modo de Bloqueio: o FTP quebra o dado em blocos (cabeçalho, contagem de *byte* e campo de dado) e passa para o TCP.
- Modo Comprimido: o dado é comprimido utilizando um algoritmo simples.

2.3 *Web Cache*

Segundo Ibope (2012), são 94,2 milhões de internautas brasileiros, sendo o Brasil o quinto país com mais usuários conectados. Com o constante crescimento da quantidade de acessos, torna-se preocupante a qualidade de desempenho no acesso de serviços na *Internet*. No intuito de minimizar as consequências deste crescimento, vários métodos podem ser aplicados de várias formas, como por exemplo aumento da largura de banda, reposição de sistemas com servidores mais rápidos, melhorias nos enlaces e etc. Porém, estas alternativas podem não resolver os problemas, e não ser as melhores alternativas econômicas. Sistemas como *web cache*, espelhamento de arquivos, entre outros são alternativas que podem ser usados para evitar gargalos na rede.

Um sistema de *cache* é um sistema que armazena temporariamente objetos (páginas *web*, imagens e arquivos) da *Internet*. Contudo, pode melhorar a qualidade de acesso à *Internet* fornecida aos usuários. Computadores modernos usam este conceito afim de melhorar o desempenho de processadores e a velocidade de acesso ao disco e etc.

Geralmente, *browsers* conectam-se diretamente aos servidores os quais são requisitados, podendo também serem configurados para conectar-se a um servidor *proxy*. Quando uma página é requisitada, uma consulta é feita ao *cache* local, caso a página não seja encontrada, a requisição será atendida imediatamente, caso contrário, a requisição será feita ao servidor e a transferência é feita armazenando uma cópia no *cache* e enviando outra para o usuário.

De acordo com suas características, um serviço *cache* é classificado de acordo com os seguintes itens:

- *Browser Cache*: geralmente os *browsers* possuem um *cache* próprio pelo fato de várias páginas serem acessadas frequentemente pelo usuário. Todavia, este serviço não é compartilhado entre usuários.

- *Proxy Cache*: é acessado e compartilhado por vários usuários e age como intermediador entre *web* e usuários.
- *Transparent Proxy Cache*: intercepta o tráfego transparente ao *browser*. É usado por ISP e não necessita de configurar junto ao *browser* do usuário.

Os servidores *proxy* podem compartilhar informações entre si, visando a redução da banda utilizada nas WANs e o tempo de acesso às páginas. Contudo, eles apresentam diversas vantagens:

- Redução de tráfego - menos requisições e menos respostas.
- Redução de carga dos servidores - menos requisições para o servidor *web* atender.
- Redução de latência - as respostas de requisições a objetos "cacheados"são feitas a partir da *cache* local.
- Possibilidade de acesso - caso um servidor *web* requisitado esteja indisponível, é possível visualiza-la caso ela esteja armazenada, porém impossível de ser atualizada.

2.4 Redes *Peer-to-Peer*

Durante o surgimento da *Internet* alguns servidores de endereços IP fixos comunicavam-se entre si para compartilhar recursos. Todavia, incialmente quando a *Internet* passou a existir, apenas instituições militares e universidades tinham acesso a ela, compartilhando informações entre si. Não sendo um conceito novo, a computação *Peer-to-Peer* (P2P) foi uma das primeiras formas de compartilhar algo na *Internet*. Alguns autores questionam argumentos pelo fato de que quando dois computadores foram conectados pela primeira vez, formaram uma rede *Peer-to-Peer*.

Segundo COULOURIS (2007), sistemas *Peer-to-Peer* representam um paradigma para a construção de sistemas distribuídos e aplicações em que dados e recursos computacionais são compartilhados por muitos *hosts* na *Internet*, os quais, participam de um serviço uniforme. O surgimento desses sistemas se dá pelo constante e rápido crescimento do uso de recursos compartilhados na *Internet*. Mesmo em localidades diferentes do planeta, os sistemas *Peer-to-Peer* garantem acesso a dados mantendo a disponibilidade dos recursos. Aplicações *Peer-to-Peer* são utilizadas para compartilhar arquivos, *web cache*, serviços de domínio, entre outros, por meios de *hosts* conectados à *Internet*.

2.4.1 Redes de Sobreposição

Processos que constituem um sistema *Peer-to-Peer* são todos semelhantes. Então cada função a ser realizada é representada por todo processo presente no sistema distribuído. Todavia, a interação feita entre os processos é feita simetricamente. Cada processo se comporta como um cliente e um servidor.

Por conta desse comportamento simétrico, as arquiteturas *Peer-to-Peer* são desenvolvidas no interesse de como organizar os processos em uma rede de sobreposição (rede sobreposta ou de *overlays*), que de acordo com TANENBAUM (2007), é uma rede na qual os nós são formados pelos processos e os enlaces representam os canais de comunicação possíveis (geralmente são realizados como conexões TCP).

Para localizar os nós e objetos na rede de sobreposição, é preciso implementar um algoritmo de roteamento distribuído sobre a camada de aplicação. Com isso as requisições dos clientes são encaminhadas para um *host* que possui o objeto do qual a requisição está endereçada, objetos estes que são colocados e movidos em qualquer nó sem envolver o cliente. Geralmente em um sistema *Peer-to-Peer* são armazenadas múltiplas replicas de objetos, garantindo a disponibilidade do mesmo. O algoritmo de roteamento permite que qualquer objeto seja acessado por qualquer nó na rede de sobreposição e também procura manter o mínimo de informação possível sobre a localização das réplicas enviando as requisições para nós ativos com menor distância. Existem dois tipos de redes de sobreposição: as que são estruturadas e as que não são. A Figura 1 mostra a organização de uma rede de sobreposição estruturada.

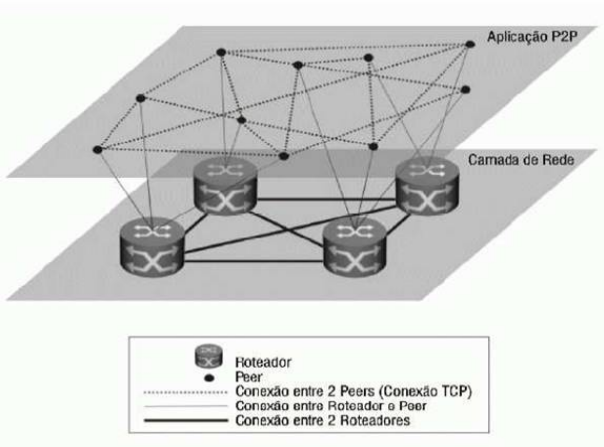

Figura 1 *– Rede de sobreposição ou overlay*

2.4.2 Arquitetura *Peer-to-Peer* estruturada

Segundo TANENBAUM (2007), em uma arquitetura *Peer-to-Peer* estruturada, a rede de sobreposição é construída com a utilização de um procedimento determinístico que é organizar

os processos por meio de uma tabela *hash* distribuída (*Distributed Hash Table* - DHT).

No sistema que se baseia em DHT, os itens de dados são identificados com uma chave aleatória, com um identificador de 128 ou 160 *bits* e da mesma maneira, os nós do sistema. Perante a isso, o método busca implementar um esquema eficiente e determinístico para mapear exclusivamente a chave de um item de dado para o identificador de um nó baseando-se na distância métrica.

O sistema *Chord*, por exemplo, tem a função de mapear nós de acordo com a chave. De acordo com TANENBAUM (2007), neste sistema os nós estão logicamente organizados em um anel de tal modo que um item de dado k (variável de *hashing* para atribuir chaves aos nós) seja mapeado para os nós que tenham o menor identificador id$\geq$k. Cada nó *Chord* pode resolver a função *hash* ao comunicar-se com outros nós, já que a tabela de roteamento é distribuída.

O *Chord* roteia uma chave através de uma série de outros nós para o destino, onde cada nó manterá atalhos para outros nós de modo que as consultas em geral possam ser feitas em O(log(N)) números de etapas, em que N é o número de nós que participam da rede de sobreposição. Mas as informações sobre estes outros nós devem estar atualizadas para que haja um roteamento eficiente. Em uma rede dinâmica, nós entram e saem a qualquer momento, mesmo assim para manter a habilidade de localizar toda chave na rede, o sistema garante que o sucessor de cada nó seja mantido corretamente, e que para toda chave k, o sucessor(k) – succ(k) ou id$\geq$k - seja responsável por k.

2.4.3 Arquitetura *Peer-to-Peer* não estruturada

Para TANENBAUM (2007), sistemas *Peer-to-Peer* não estruturados baseiam-se em algoritmos aleatórios para construir uma rede de sobreposição, sendo que a ideia principal é que cada nó mantenha uma lista de vizinhos, mas que essa lista seja construída de modo mais ou menos aleatório.

Quando um nó necessita localizar um objeto específico, a única coisa a se fazer é inundar a rede com uma consulta de busca. Deste modo muitas consultas podem não ser respondidas, pois o cliente e o *host* podem estar muito afastados na rede, o que se torna uma desvantagem. Outra desvantagem que pode ser destacada é que mecanismos de inundação ocasionando grande tráfego de sinalização e lentidão. Exemplos de sistemas *Peer-to-Peer* que se dispõe de arquitetura não estruturada são: Kazaa e Napster.

2.5 Gerenciadores de *Download*

A cada dia o número de usuários com acesso à *Internet* aumenta assim como também a infinidade de arquivos compartilhados entre eles. Para cada arquivo é feita uma requisição

de *download* para obtê-los. As requisições de *download* são feitas através dos *browser* para servidores que hospedam os arquivos compartilhados pelos usuários.

Muitas vezes a transferência de um arquivo pode ser realizada tanto pelo *browser* como por gerenciadores de *download*. Os gerenciadores de *download* são *softwares* desevolvidos para realizar *download* de arquivos compartilhados na *Internet*. Estes *softwares* permitem ao usuário: (i) Pausar *download* de arquivos grandes; (ii) Reiniciar *downloads* parados ou corrompidos; (iii) Transferir arquivos em conexões lentas; (iv) Agendar *downloads*; (v) Quebrar os arquivos em partes, baixando-as simultaneamente. Nas próximas subseções serão apresentados três gerenciadores de *download* populares na *Internet*: *Download Accelerator Plus*, *Internet Download Manager* e o JDownloader.

2.5.1 *Download Accelerator Plus*

Figura 2 *– Gerenciador Download Accelerator Plus*

O *Download Accelerator Plus* é um *software* gerenciador de *downloads* pago e desenvolvido exclusivamente para ser executado em sistemas operacionais *Windows*. O gerenciador tem a capacidade de estabelecer múltiplas conexões para aumentar a velocidade do descarrecamento dos *downloads*. Permite que o usuário o agendamento de *downloads* e reinicia-los quando a conexão é perdida e quando os *downoads* estão pausados ou com o arquivo corrompido. O DAP interage com a maioria dos *browsers*. A Figura 2 mostra uma instancia do gerenciador de *downloads Download Accelerator Plus*.

2.5.2 IDM - *Internet Download Manager*

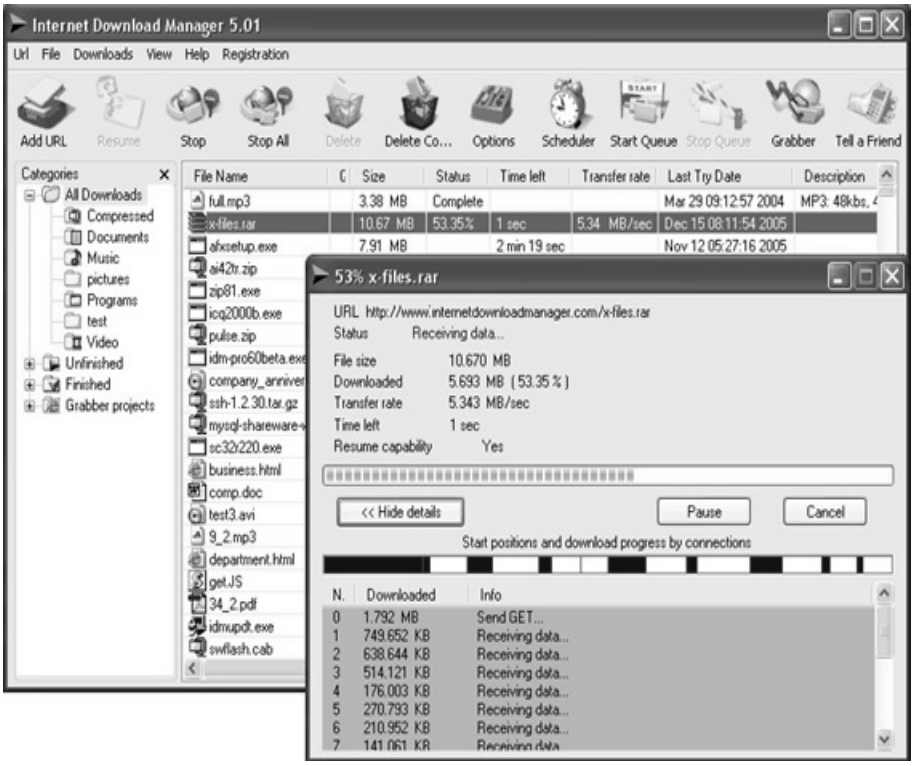

Figura 3 *– Gerenciador Internet Download Manager*

O IDM é um software gerenciador de *download* desenvolvido por ToneC. É um software pago e somente é utilizado em sistemas operacionais *Windows*. Ele estabelece múltiplas conexões para aumentar a velocidade do descarrecamento dos *downloads*. Permite o agendamento de *downloads* e reinicia-los quando a conexão é perdida e quando os *downoads* estão pausados ou com o arquivo corrompido. O gerenciador IDM integra com a maioria dos *browsers*. A Figura 3 mostra janelas do IDM. Segundo plano se encontra a tela principal do *software* e no primeiro plano a tela representando uma instância de um *download*.

2.5.3 *JDownloader*

O *JDownloader* é um gerenciador de *download* desenvolvido pela equipe JD e disponibilizado de forma gratuita na *Internet*. O gerenciador é um *software* livre de código-fonte aberto, desenvolvido em linguagem de programação JAVA e permite ser executado em diferentes sistemas operacionais. Ele estabelece múltiplas conexões para aumentar a velocidade do descarrecamento dos *downloads*. Permite o agendamento de *downloads* e reinicia-los quando a conexão é perdida e quando os *downoads* estão pausados ou com o arquivo corrompido. O gerenciador IDM integra com a maioria dos *browsers*A Figura 4 mostra a janela principal do *JDownloader*.

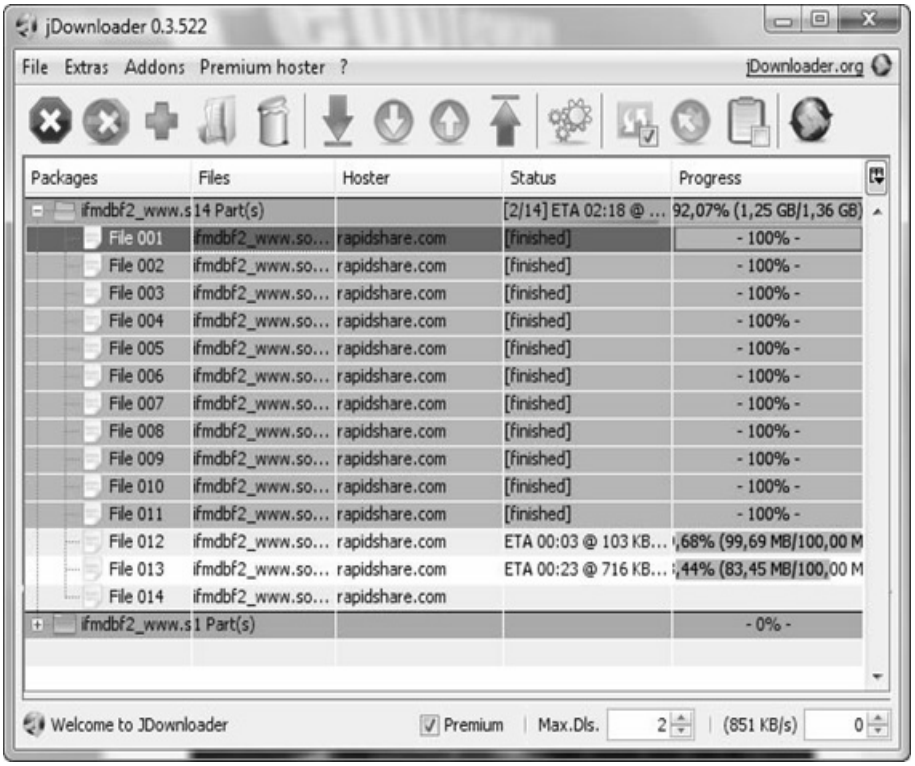

Figura 4 *– Gerenciador JDownloader*

2.6 Ourdown

O Ourdown não acelera os *downloads* como outros gerenciadores fazem, até porque, uma vez que esses gerenciadores de *download* estabelecem múltiplas conexões para transferir o arquivo desejado. Vale ressaltar, que múltiplas conexões é uma das causas da lentidão do tráfego na rede.

Desenvolvido em JAVA, o *software* permite ser executado em ambientes Linux e Windows. A integração com o *browser* é feita através de uma extensão desenvolvida para o navegador Firefox. Extensão esta que ao momento que o usuário clicar com o botão direito do mouse sobre o *link* de *download* a requisição será enviada para o Ourdown. A maioria dos outros gerenciadores se integram com a maioria *browsers* existentes, e isso faz com que haja o interesse de se pesquisar novas formas para que o Ourdown possa interagir com outros *browsers*.

O Ourdown é um *software* distribuído, baseado no modelo de *Peer-to-Peer*. Comunica-se com outros componentes através da rede local trabalhando colaborativa com o objetivo de controlar as requisições de *download* evitando o download de arquivos já baixados ou com sua transferêcnai em andamento. A Figura 5 mostra a interface de uma instância do Ourdown. O painel de mensagens do sistema é onde o usuário obterá informações das ocorrências do sistema como por exemplo as requisições de *download* recebidas do navegador e dos outros *hosts* e informação quando um *download* é concluído. A tabela de *downloads* é onde as transferências são representadas graficamente para que o usuário seja melhor informado quanto a progresso da

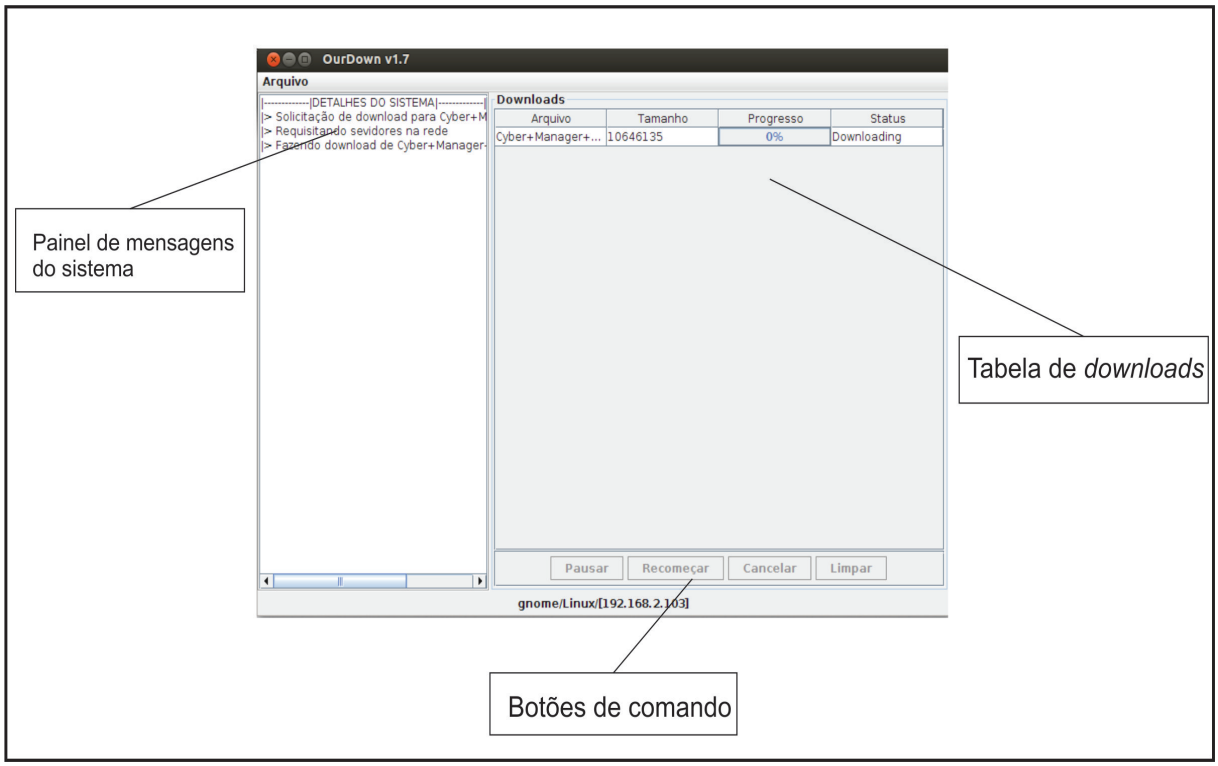

Figura 5 *– Gerenciador de download Ourdown*

transferência, o tamanho do arquivo e o nome do arquivo no servidor. Os botões de comando, são os botões que farão o controle do progresso do *download*: Pausar, Recomeçar, Cancelar e Limpar (retira o *download* selecionado da tabela).

Tabela 1 *– Comparativo entre o Ourdown e outros gerenciadores de downloads*

	Ourdown	IDM	DAP	JDownloader
Pausar *downloads*	X	X	X	X
Reiniciar *downloads*	X	X	X	X
Agendar *downloads*		X	X	X
Múltiplas conexões por *download*		X	X	X
Integração com mais de um *browser*		X	X	X
Multiplataforma	X			X
Restringe *downloads* de arquivos existentes	X			

A Tabela 1, permite fazer um comparativo entre o Ourdown e os gerenciadores de *download* que foram conhecidos durante o desenvolvimento do projeto. É visto que só são duas características que fazem o Ourdown semelhante aos outros gerenciadores de *download*. Porém, as outras funções podem serem estudadas e incrementas em trabalhos futuros. A função de restringir *downloads* de arquivos existentes somente o Ourdown apresenta porque este é o diferencial no projeto.

3 Projeto Ourdown

Este capítulo tem o objetivo de apresentar a modelagem do sistema Ourdown. Primeiramente é apresentado o modelo conceitual de seu funcionamento. Em seguida são utilizados dois artefatos da UML para ilustrar sua função (Diagramas de Caso de Uso e de Sequência) e as Classes usadas na implementação do código.

3.1 Modelagem do sistema Ourdown

Esta subseção apresenta a proposta computacional de modelagem de objetos utilizados na implementação dos componentes.

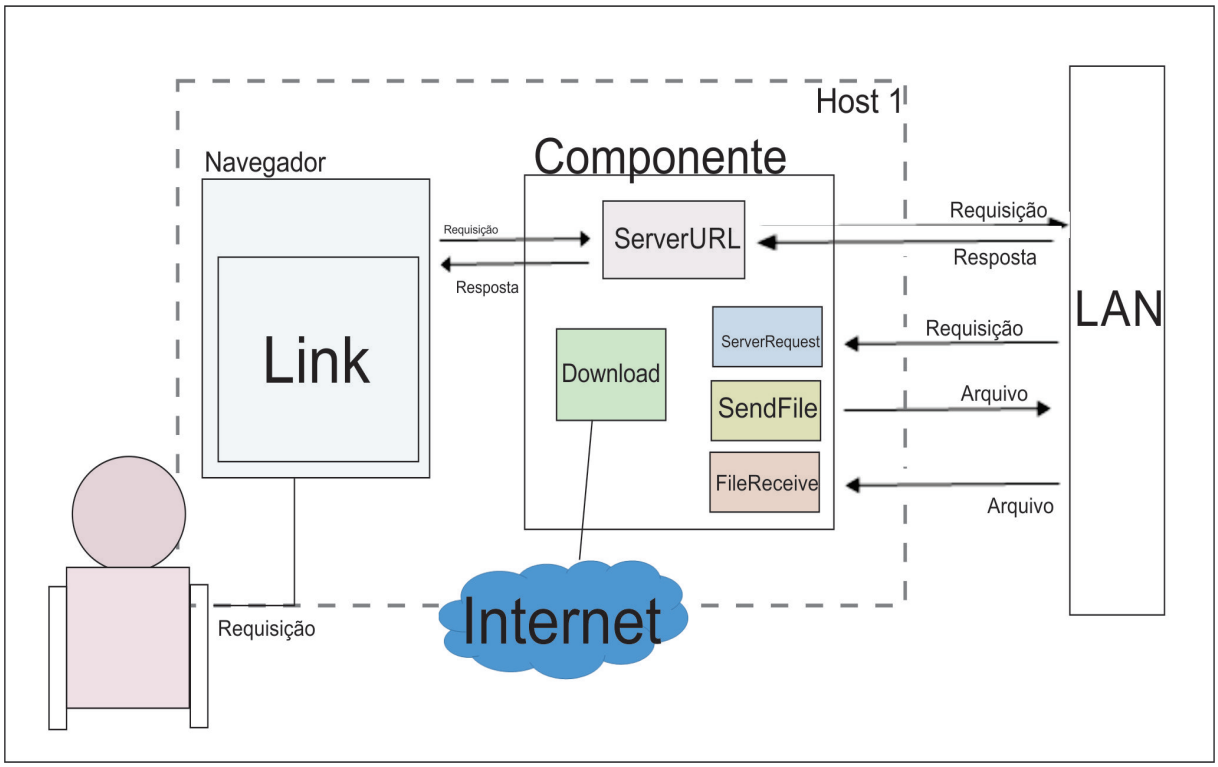

Figura 6 *– Objetos do componente*

A Figura 6 apresenta o Ourdown como um Componente detalhado de acordo com seus principais objetos e sua localização no ambiente. O Componente atua como *servant* e assume os papéis de cliente e servidor. Ele foi decomposto em cinco objetos afim de facilitar a apresentação e explicação onde as quais foram referenciados por ServerURL, Download, ServerRequest,

SendFile e FileReceive. É importante observar na figura que o componente tem acesso à LAN e à *Internet* ao mesmo tempo.

O *software* possui suas funcionalidades apresentadas na Figura 6 através dos cinco objetos que o constitui. O objeto ServerURL é responsável por recepcionar as requisições vindas do navegador e também por fazer o tratamento da requisição extraindo o nome do arquivo e sua localização para verificar a existência da execução de outro *download* semelhante ao requisitado. Este objeto ainda acumula a responsabilidade de encaminhar requisições a outros componentes com o objetivo de descobrir se algum deles está realizando o *download* do arquivo requisitado pelo usuário. Caso nenhum componente na rede esteja realizando o *download* do arquivo requisitado, o objeto Download inicia a transferência do arquivo. Feito isso, informações do *download*, como por exemplo o nome e o endereço do arquivo, são armazenadas em uma lista para eventualidades futuras.

A qualquer momento o Ourdown está apto a receber requisições de consultas de outros componentes. O responsável por executar esta tarefa é o objeto ServerRequest. Ele recebe as requisições e busca junto às informações gravadas pelo ServerURL o arquivo solicitado. Caso encontre o arquivo, uma mensagem de resposta será reencaminhada ao componente solicitante e as informações deste serão gravadas (por exemplo, endereço IP, porta de comunicação, nome e endereço do arquivo solicitado).

Quando um *download* termina, o objeto SendFile, que é responsável por enviar arquivos para outros componentes, inspeciona as informações dos componentes gravadas pelo ServerRequest afim de encontrar algum componente interessado no arquivo recém baixado, caso encontre, o objeto estabelece uma conexão ao outro componente e envia o arquivo através da rede. O Ourdown é configurado para estar sempre permitindo conexões com os outros componentes para receber os arquivos requisitados. O objeto FileReceive é o responsável por desempenhar esta função.

É perceptível a existência de um protocolo de alto nível entre os componentes. Existem dois tipos de mensagem: a) Mensagem de requisição e b) Mensagem de resposta. A Mensagem de requisição é uma mensagem *broadcast*, ou seja, é enviada para todos os componentes contendo o nome do arquivo e sua origem, após o envio desta mensagem o componente aguardará um *timeout*, quando este *timeout* se esgota o componente inicia o *download*. A mensagem resposta é uma mensagem *unicast*, enviada diretamente ao componente requisitante que aguardará o envio do arquivo.

3.2 Diagrama de Caso de Uso

O diagrama de caso de uso representa graficamente as funcionalidades que são observáveis do sistema e de elementos externos que interagem com ele. Visa descrever ações e

variantes executadas para produzir um resultado de valor que seja observado por um ator ou entidade. A Figura 7 motra o diagrama de casos de uso do Ourdown com 6 casos de usos: Ourdown, AguardaURL, AguardaConsulta, RecebeArquivos, Download e EnviaArquivos.

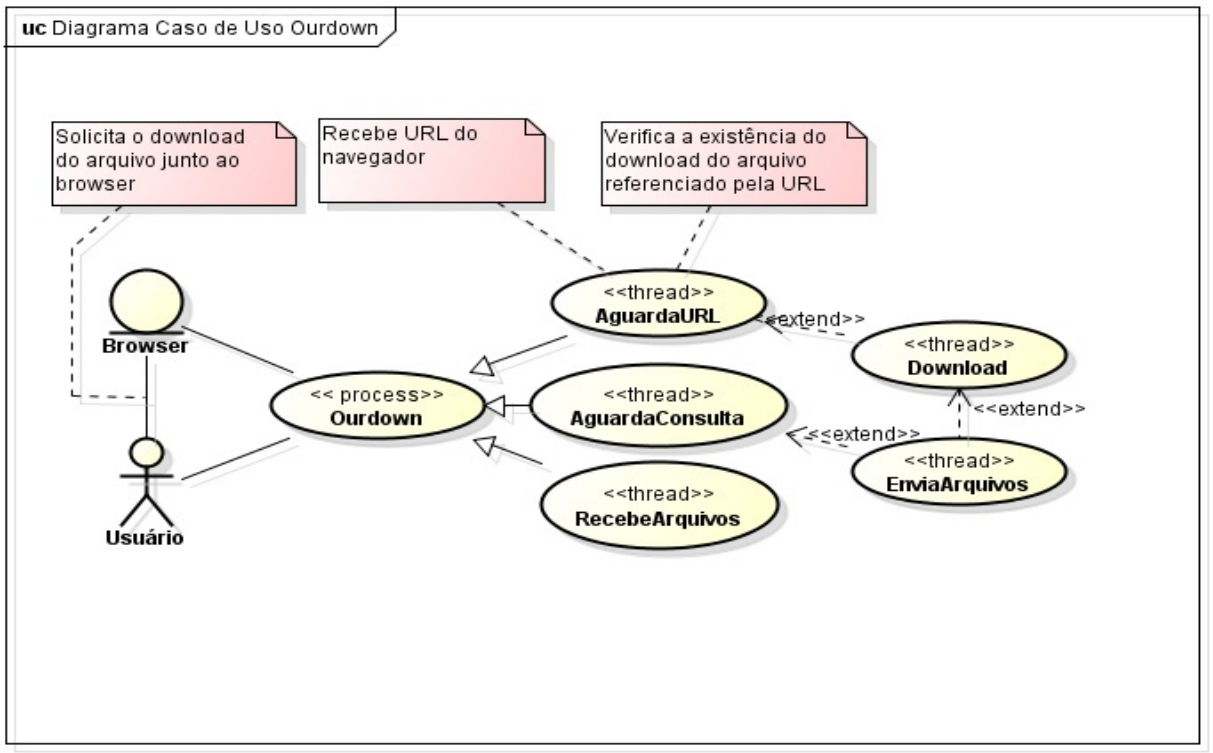

Figura 7 *– Diagrama de Caso de Uso Ourdown*

Uma vez feita a descrição de cada caso de uso, as informações colhidas contribuíram para o desenvolvimento do sistema permitindo identificar as funções que o sistema deve desempenhar e como deverão serem elaboradas.

O caso de uso Ourdown representa o momento em que o ator, usuário, altera o estado do *software*, quando passa a estar em execução. Contudo inicializará todos os componentes do *software*.

O caso de uso AguardaURL, representa o objeto destinado a receber as requisições de *download* do *browser*. O fluxo principal deste caso de uso: receber requisições do *browser*; verificar a existência do arquivo baixado ou com o *download* em andamento; enviar requisições para outros componentes através da rede local; responder as requisições para o *browser*.

O caso de uso RecebeArquivos tem como objetivo receber arquivos de outros componentes conectados à rede local. É acionado quando o software é iniciado pelo usuário e permanece ativo aguardando conexões para a receber arquivos solicitados a outros componentes.

O AguardaConsulta foi desenvolvido para desempenha a função de recpetor de requisições oriundas da rede local. O fluxo atividade neste caso de uso limita-se a receber requisições emitidas por outros componentes, comparar o arquivo solicitado junto a lista de *downloads*

afim certificar aexistência do arquivo solicitado, se a existência for confirmada, armazenar informações do componente solicitante e encaminhar uma mensagem resposta ao componente que enviou a requisição.

O caso de uso Download é destinado a executar os *downloads* requisitados pelo usuário. Este caso de uso é acionado logo após a requisição do *download* ser processado em AguardaURL e informa ao usuário informções da transferência através de uma interface gráfica.

O caso de uso EnviaArquivos tem como objetivo encaminhar arquivos a outros componentes na rede local. Após a tarefa de *download* terminada é iniciada as atividades deste caso de uso: verificar a existência de requisições pelo arquivo; preparar o envio do arquivo de acordo com as informações do componente solicitante; e por fim, efetuar a transferência do arquivo.

3.3 Diagrama de Sequência

O diagrama de sequência enfatiza o tempo de sequência mostrando objetos participando em interações de acordo com suas linhas de vida e as mensagens que trocam enquanto realizam uma operação. Nele podem ser encontrados os seguintes elementos:

- Linhas verticais que representam o tempo de vida de um objeto (*lifeline*). Indicam quando um objeto passa a existir;
- Linhas horizontais ou diagonais representam as mensagens trocadas pelos objetos. Nelas são embutidas o nome da mensagem e os seu parâmetros, estes não obrigatórios.

As mensagens existentes podem ser enviadas para o mesmo objeto, representando uma interação. Uma condição é representada na mensagem entre colchetes. As mensagens de retorno são representadas através de linhas tracejadas, o que não é frequentemente utilizada nos diagramas. A Figura 8 representa o diagrama de sequência para exemplificar o caso de uso utilizado para o desenvolvimento do Ourdown.

O digrama da Figura 8 mostra a sequência de ações decorrentes a uma requisição de *download*. O usuário requisita o *download* de um arquivo junto ao *browser*, que encaminha a requisição ao Ourdown. AguardaURL recebe a requisição do *browser*. VerifDownload busca indícios da existência do *download* do arquivo no *host* e também faz uma consulta a outros *hosts* na rede. Se AguardaURL obtém qualquer resposta de outro *host*, RecebeArquivos é acionado com a finalidade receber o arquivo do outro *host*. Qualquer resultado obtido em VerifDownload, AguardaURL encaminhará uma mensagem para o *browser*. Na medida em que não é detectada a existência do *download* requisitado, Download iniciará a transferência do arquivo. Ao terminar o *download* EnviaArquivos procurará alguma requisição do arquivo e por fim se houver enviará o arquivo ao destinatário requisitante.

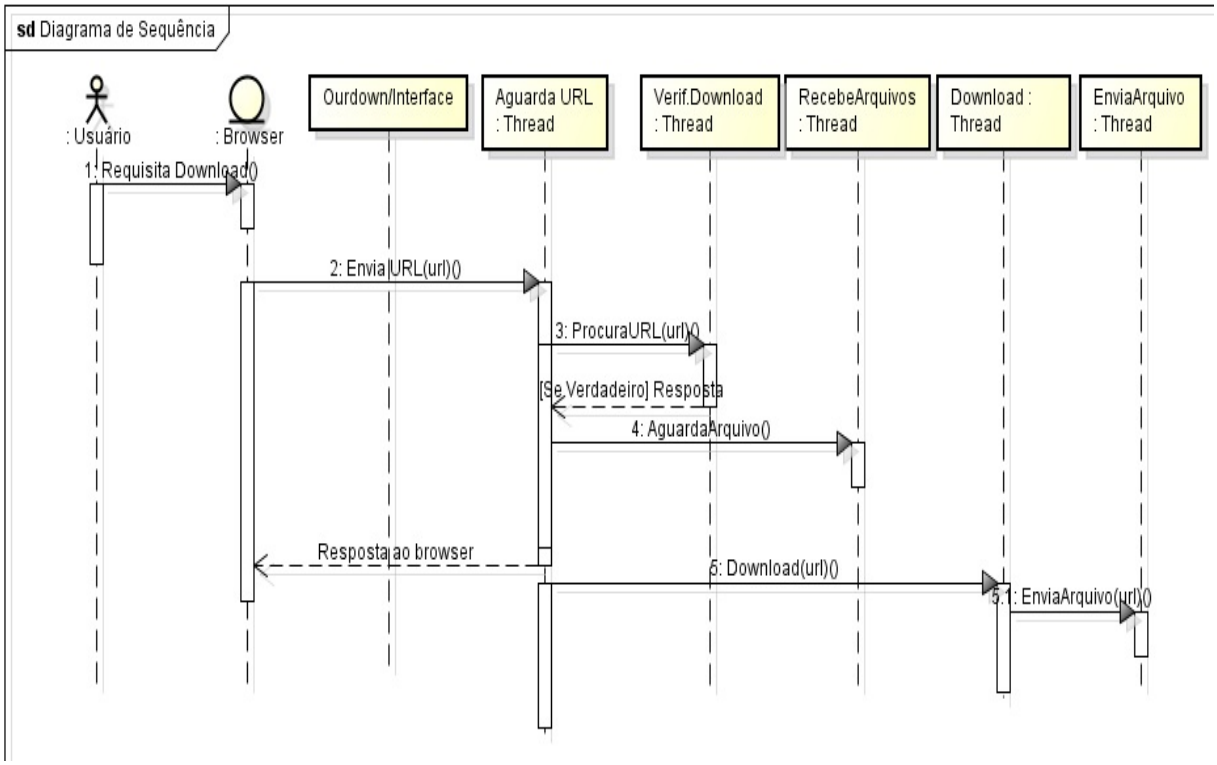

Figura 8 *– Diagrama de Sequência*

3.4 Diagrama de Classes

Enquanto o diagrama de caso de uso fornece uma perspectiva do sistema do ponto de vista do ator, internamente, objetos colaboram para atender às funcionalidades do sistema. O diagrama de classes demonstra a estrutura da colaboração entre os objetos, apresenta as classes do sistema e seus relacionamentos, atributos e operações. A Figura 9, representa a estrutura e relações entre as classes, sendo base para o desenvolvimento do Ourdown. No diagrama de classes da Figura 9, estão definidas as classes, os atributos, tipos de atributos e operações referidas a um objeto. No diagrama de classes da Figura 8, pode-se observar a existência de 11 classes: Ourdown, ServerURL, Download, FileReceive, OurdownScreen, Progresso, SendFile, ServerRequest, Config, TabelaDownloads, TratamentoDeUrl.

- Classe Ourdown: é a classe principal do projeto;
- Classe Config: possui um construtor e quatro métodos. Cada método ao ser executado fará tarefas distintas o que resultará em dados processados que serão utilizados por objetos de outras classes;
- Classe Download: nessa classe será manipulada a vida do *download*. As funções pausar, continuar e cancelar, serão implementadas pelos métodos métodos *pause()*, *resume()* e *cancel()* respectivamente. Esta classe dependerá de atributos provenienes da classe Config

como da classe Ourdown;

- Classe FileReceive: trata do recebimento de arquivos enviados pelo Ourdown presentes em outros *hosts*.
- Classe OurdownScreen: determinada para gerenciar a janela da aplicação além de fornecer alguns métodos para que outras classes realize o tratamento da URL;
- Classe Progresso: implementa a barra de progresso de *download* presente na tabela dos *downloads*;
- Classe SendFile: responsável pelo envio dos arquivos baixados além de fornecer métodos para que outras classes possa fazer esta operação;
- ServerRequest: gerencia as mensagens recebidas através da rede, de outros *hosts*;
- Classe ServerURL: implementa o gerenciador de mensagens que são recebidas do *browser*. Faz o tratamento, verifica a existência de *download* referente à URL recebida além de enviar resposta para o *browser*;
- TabelaDownloads: implementa a tabela que mostrará a lista de *downloads* na tela do Ourdown;
- TratamentoDeUrl: faz o tratamento de URLs para as outras classes.

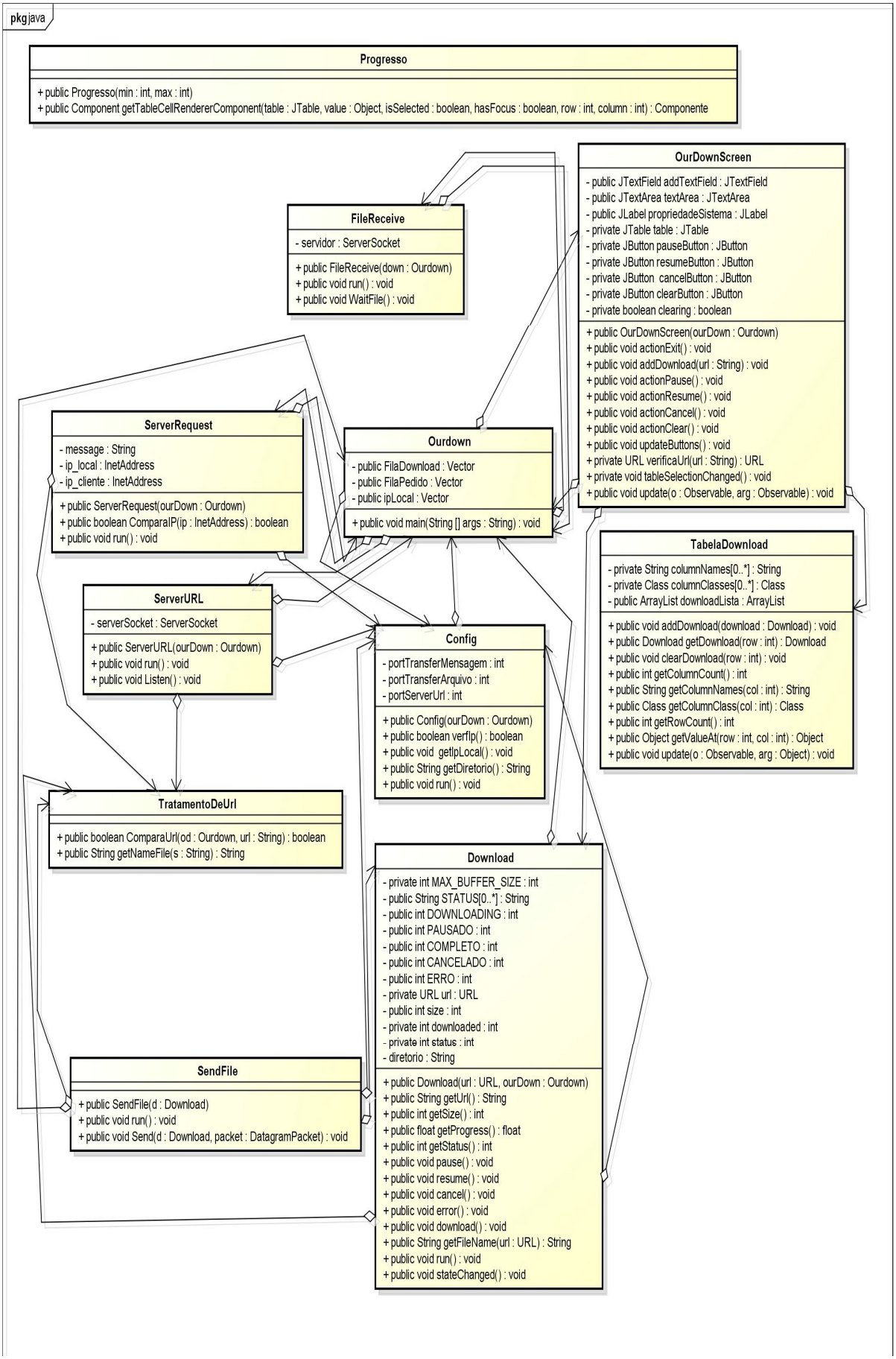

Figura 9 *– Diagrama de Classes*

4 Tecnologias, Infraestrutura e Desenvolvimento

Este capítulo dedica-se a descrever as tecnologias utilizadas para a implementação do projeto, como também a infraestrutura e o desenvolvimento.

4.1 Tecnologias e Infraestrutura

Seguindo a modelagem do sistema, é dada a partida para início da implementação da aplicação. Durante esta etapa foram avaliadas tecnologias que serão utilizadas e contribuirão para o desenvolvimento do Ourdown.

4.1.1 Firefox

O navegador *web*, ou *browser*, permite que o usuário acesse conteúdo virtual da *Internet*, onde tal conteúdo é requisitado pelo usuário através de um endereço que referencia uma máquina a qual armazena o conteúdo. É também com os *browsers* que o usuário solicita a transferência de arquivos guardados em um servidor.

O Firefox tornou-se importante para o projeto por ser um multiplataforma, ou seja, pode ser executado em diferentes sistemas operacionais além de permitir a adição de componentes que fazem mudar as suas funcionalidades e características, o que se torna prático e fácil, a elaboração de um mecanismo para a captura das solicitações de *download* feitas pelo usuário. O Firefox é escrito com linguagem de marcação desenvolvida pela própria Mozilla, o XUL, Linguagem de Interface XML. Segundo a Mozilla (1995) a linguagem XUL foi criada para o desenvolvimento de interfaces de usuários portáteis, ou seja, para ser usada independente de plataforma. O Firefox é um software livre de código aberto, todavia o usuário poderá alterar suas características e funcionamento através de extensões.

De acordo com a Mozilla (1995) as extensões do Firefox, são pacotes contendo arquivos que podem modificar o comportamento e interface do Firefox. Podem ainda monitorar eventos do browser, como por exemplo, o clique do mouse sobre algum link, e acessar e modificar os módulos do navegador. As extensões assim como o próprio navegador são codificadas com a linguagem XUL. A extensão ou pacote, é contida por documentos que descrevem a extensão

baseando-se no seu funcionamento, contem também um arquivo manifesto pelo qual o Firefox sabe que é uma extensão e que quer ser instalada, tem ainda a pasta *chrome* na qual estão inseridos os conteúdos que personificam a aparência da extensão, idioma, documentos e arquivos de proprietário para localização e *scripts* definem o funcionamento da extensão que por padrão da Mozilla é usado a linguagem JavaScript.

É através de uma extensão que as solicitações de *download* serão encaminhadas para o *Ourdown*, e este tratará de gerenciar os *downloads* requisitados.

4.1.2 Java e Sockets

Por ser uma linguagem multi-plataforma, pois a portabilidade é um dos pontos mais importantes do projeto, o Ourdown foi desenvolvido por meio da plataforma de desenvolvimento Java. Além disso, tornou-se ainda mais atrativo para o desenvolvimento com essa linguagem, o suporte a *Threads*, a programação para *Internet*, controle de concorrência e paralelismo como também suporte a *Sockets* TCP e UDP.

Em redes de computadores, *Sockets*, são abstrações de canais de comunicação estabelecidos entre processos, todo servidor pretendido a ser acessado deve criar um *Socket Server*. Uma vez criado um *Socket Server*, este pode ser acessado por outro programa que inicie um *Socket* com o servidor. A conexão é estabelecida por dois *sockets*, o do cliente e a do servidor. Os dois *sockets* são endereçados pela combinação de um endereço IP e um número da porta do protocolo de transporte (TCP ou UDP). Com base nesse endereçamento, os *sockets* entregam pacotes de dados de entrada para o processo ou *thread* de aplicação apropriado.

4.2 Desenvolvimento e Codificação

Após analisar as tecnologias a serem utilizadas e adquirir conhecimento sobre elas, é dada a partida para codificar e implementar as funcionalidades do gerenciador.

4.2.1 Extensão do Firefox

O trecho de código abaixo é do arquivo *browser.xul*. O conteúdo deste arquivo, é o que faz alterar as funcionalidades do Firefox, adicionar botões e janelas.

```
1 <overlay id="ourdown"
2   xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.
3   is.only.xul">
4   <script src="ourdown.js" />
5   <popup id ="contentAreaContextMenu">
6     <menuitem label="OurDown"
```

```
7       oncommand="linkTargetFinder.run()"/>
8     </popup>
9   </overlay>
```

Quando o usuário clicar sobre algum *link* com o botão direito do *mouse*, o código do arquivo *browser.xul* faz com que na *pop-up*, que sempre aparece neste evento, adicione a propriedade ourdown entre os menus, como mostrado na Figura 10.

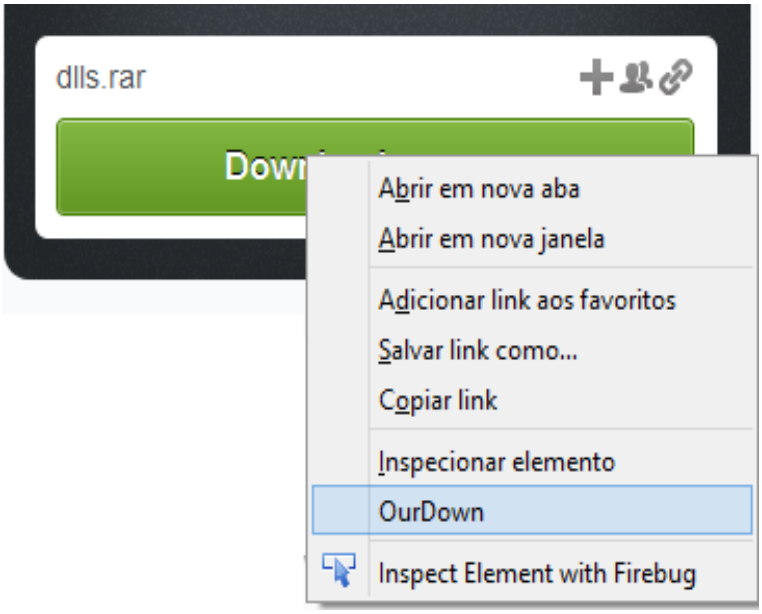

Figura 10 *– Opção Ourdown no menu pop-up no Firefox*

O processo de envio da solicitação de *download* é trabalhado pelo arquivo *oudown.js* contido na extensão, onde nele é escrito o código da função em AJAX. O trecho de código abaixo descreve este processo.

```
1  try
2    {
3    // Firefox, Opera 8.0+, Safari
4      xmlHttp=new XMLHttpRequest();
5    }
6  catch (e)
7    xmlHttp.open("GET","http://localhost:2121?="+arquivo,true);
8    xmlHttp.send(null);
```

O método *open* do objeto xmlHttp estabelece uma conexão para porta 2121 destinando-se à própria máquina, permitindo que o *browser* encaminhe as requisições para o Ourdown, onde existe um componente dedicado para a recepção destas conexões.

4.2.2 Recebendo requisições de *download* do Firefox

Para receber as requisições de *download* do navegador, foi elaborado um mecanismo implementado por um *thread* executando um *Server Socket* vinculado para o endereço da própria máquina e para porta 2121, também utilizada pela extensão do Firefox. O trecho de código abaixo é do arquivo *ServerURL.java* implementa o mecanismo.

```
1  public void run(){
2    try{
3      //cria um socket servidor não acoplado
4      serverSocket = new ServerSocket();
5      serverSocket.bind(new InetSocketAddress("localhost",
6        Config.portServerUrl));
7      //Vincula o ServerSocket para um endereço específico
8      //(endereço IP e número de porta).
9      Listen();
10   }catch(Exception e){
11     e.printStackTrace();
12   }
13 }
```

Além de receber as requisições, o mesmo mecanismo é capaz de enviar mensagens ao navegador, contendo informações do *download* requisitado.

4.2.3 Enviando e recebendo mensagens de outros componentes através da rede local

A mensagem para a rede local destinada a todos os componentes presentes é encapsulada por um datagrama UDP, como o destino a priori é desconhecido o datagrama é enviado para o endereço de *broadcast* da rede. O trecho de código do arquivo *ServerURL.java* abaixo mostra como ocorre esta tarefa.

```
1  //criar um datagrama socket
2  ds = new DatagramSocket();
3  //cria um buffer do tamanho da url
4  byte[] bufout = new byte[url.length()];
5  //prepara pacote para ser enviado em broadcast
6  InetAddress ip = InetAddress.getByName("255.255.255.255");
7  bufout = url.getBytes();//
8  DatagramPacket packet = new DatagramPacket(bufout, bufout.length,
9    ip, Config.portTransfMensagem);//prepara um datagrama
10 ds.send(packet);//envia uma datagrama na rede
```

```
System.out.println("->A pesquisar servidores na rede");
od.ods.textArea.append("\n|> Requisitando servidores na rede");
//estancia para que seja esperado 2s para a espera de resposta
ds.setSoTimeout(5000);
byte [] bufin = new byte[1024];//cria um novo buffer
DatagramPacket dsin = new DatagramPacket(bufin, bufin.length);
ds.receive(dsin);
```

Esse código apresenta a construção de um datagrama representado pelo objeto Java packet, o qual é configurado com um *buffer* e o seu tamanho, um endereço destinado para 255.255.255.255, que representa o endereço broadcast em uma rede local, e também a porta do canal de comunicação. O código o mecanismo dedicao ao envio de datagramas UDP em broadcast na rede local. Esta solução implica que o código é dependente da Arquitetura TCP/IP e seu escopo de atuação restringe a uma rede local, haja vista que datagramas UDP de broadcast são descartados nos roteadores.

O artifício utilizado para enviar uma mensagem para todos os componentes da rede é uma característica particular da camada de rede da Arquitetura TCP/IP. Assim, este código depende de como a camada de rede trata os datagramas com endereço de broadcast. O uso do UDP também se fundamenta por ser um protocolo com baixo overhead de rede e preferível em relação ao TCP neste aspecto.

No código abaixo, do arquivo *ServerRequet.java*, um datagrama também é representado pelo objeto Java packet. Este datagrama serve, para receber mensagens oriundas da rede, encaminhadas por outros componentes. Nele está contido o endereço do solicitante, a porta, e também a mensagem.

```
byte[] buffer = new byte[1024];
System.out.println("<<--Servidor iniciado"+od.ipLocal+"-->");
DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
socket.receive(packet);
ip_cliente = packet.getAddress();
int client_port = packet.getPort();
message = new String(packet.getData()).substring(0, packet.getLength());
```

4.2.4 Enviando arquivos solicitados por outros componentes

Quando o *download* é terminado, é realizada uma tarefa de verificação para saber se há alguma solicitação pelo arquivo baixado, caso haja, um mecanismo de envio é acionado, para que o arquivo seja transferido pela rede até ao componentente solicitante. O trecho de código do arquivo *SendFile.java* a seguir mostra como o trabalho é realizado.

```
try {
  System.out.println("Download de"+
    download.getFileName(new URL(download.getUrl()))+" terminado");
  if(od.FilaPedido.isEmpty()){
    System.out.println("Fila de pedidos vazia!");
  }else{
    Iterator <DatagramPacket> pedido = od.FilaPedido.iterator();
    //procura o arquivo referente ao lista de pedidos
    while(pedido.hasNext()){
      DatagramPacket dp = pedido.next();
      String file_request = trataUrl.getNameFile(new String(dp.getData()));
      tring file_download = trataUrl.getNameFile(download.getUrl());
      if(file_request.equals(file_download)){
        System.out.println("Cliente: "+
          dp.getAddress().getHostAddress()+" pedido: "+download.getUrl());
        Send(download, dp);
        //quando o pedido for enviado, o mesmo será deletado
        od.FilaPedido.remove(dp);
      }
    }
  }
} catch (Exception e) {
  e.printStackTrace();
}
```

4.2.5 Recebendo arquivos de outros componentes

O *software* Ourdown é programado para estar sempre a espera por conexões para que os arquivos sejam recebidos sem interferir outros processos do componente. O trecho de código a seguir do arquivo *FileReceive.java* mostra como é implementada esta função do *software*.

```
try {
  System.out.println(
    "<<--Servidor iniciado para recepção de arquivos-->>");
  servidor = new ServerSocket(Config.portTransfArquivo);
  //inicia um thread para espera por transferencia de algum arquivo requisitado
  WaitFile();
} catch (Exception e) {
  e.printStackTrace();
  System.exit(1);
}

```

No código é criado um objeto do tipo *ObjectInputStream* para obter o arquivo no canal estabelecido através de *sockets*. Para este evento é configurado um *Server Socket* para aceitar conexões na porta 6000.

5 Testes e Simulações

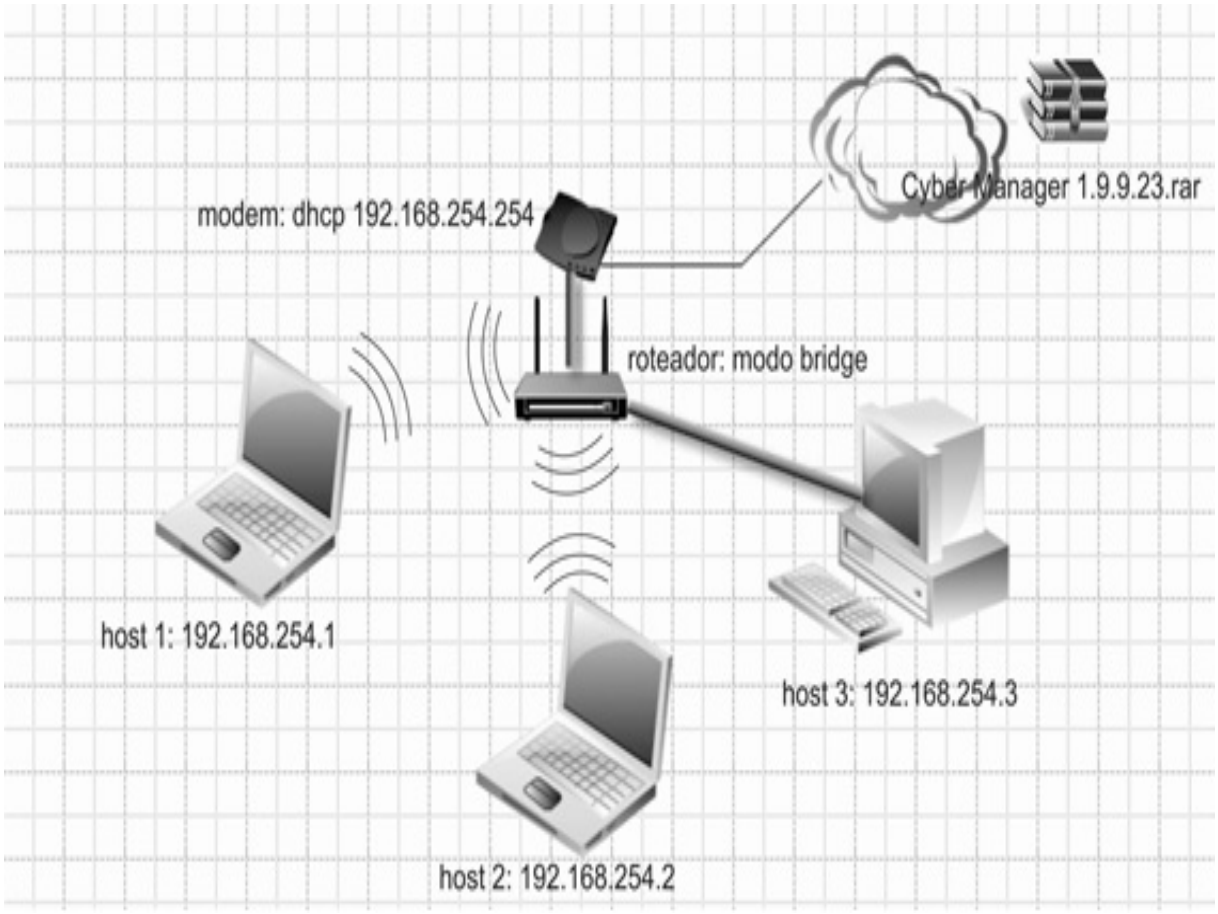

Figura 11 *– Ambiente de rede para simulação de testes*

Segundo INTHURN (2001), teste é uma das áreas de engenharia de *software* e tem como objetivo aprimorar a produtividade e fornecer evidências da confiabilidade e da qualidade de *software*, em complemento a outras atividades de qualidade ao longo do processo de desenvolvimento de software e também descobrir um erro através do processo de executar o programa. Durante a fase de testes, o desenvolvedor deve simular situações onde devem haver entrada de dados, operações inválidas, exceções e erros possíveis. Erros e deficiências deverão ser corrigidos.

Os testes com o Ourdown transcorrem em uma rede local com acesso à *Internet* com largura de banda de 1MB disponibilizado pela empresa de telefonia contratada. Vale ressaltar

que a qualidade do acesso depende dos enlaces presentes na rede, dos dispositivos e da disponibilidade e qualidade de serviço do provedor.

A Figura 11 mostra a organização do ambiente, rede local, e os dispositivos envolvidos: (i) Acesso à *Internet*, como largura de banda de 1MB; (ii) Modem Zte Zxdsl 831 Series; (iii) Roteador Wireless Intelbras 150 Mbps – WRN 240; (iv) Host 1 – Notebook LG N450, Intel® Core™ i5-3210M CPU @ 2.50GHz, 4 GB RAM DDR3, HD 500 GB; (v) Host 2 – Notebook Samsung RV420, Intel® Core™ i3-2310M CPU @ 2.10GHz, 3 GB RAM DDR3, HD 500 GB; e (vi) Host 3 – Desktop Sim+ A885, Pentium Dual Core CPU 4GHz, 4 GB RAM DDR2, HD 750 GB.

5.1 Primeito Teste

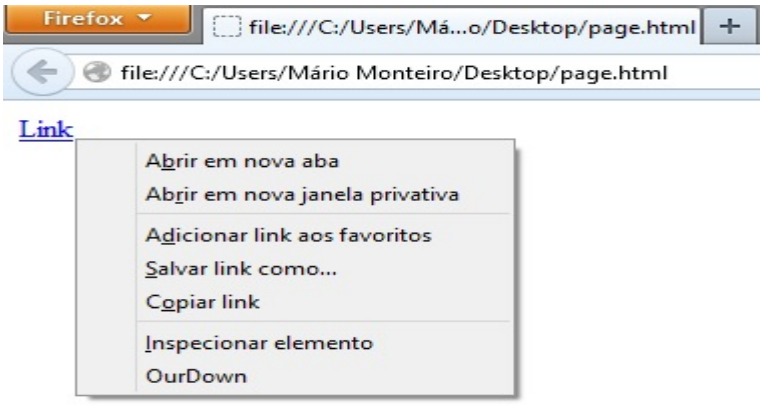

Figura 12 *– Opção ourdown no menu pop-up após instalar a extensão no Firefox*

Esse teste visa a instalação da extensão do Ourdown no Firefox para se certificar de que o *browser* desempenhará as funções estabelecidas na função. Espera-se que ao instalar a extensão, apareça no menu *pop-up* do Firefox a opção ourdown. A Figura 12 mostra o que resulta após a instalação da extensão do ourdown no Firefox.

5.2 Segundo Teste

Certificado que a extensão está instalada no *browser*, agora realiza-se teste de comunicação do Ourdown com o *browser*. O intuito é que as requisições de *download* sejam repassadas neste processo. Para isso, o componente ServerURL do Ourdown está aguardando as mensagens enviadas pelo Firefox.

O resultado deste teste pode ser visto na Figura 13 onde é mostrado o momento em que o componente ServerURL recebe a requisição do *browser*. O evento ocorre ao clicar sobre o *link* teste.com contido numa página *web*.

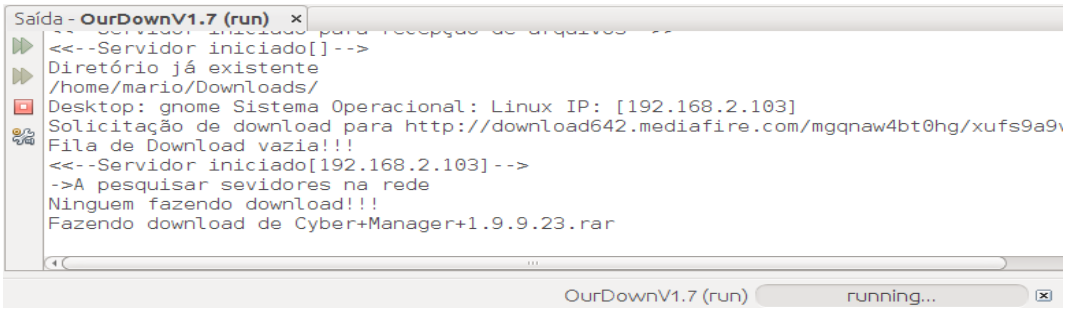

Figura 13 *– URL recebida pelo componente ServerURL enviada pela extensão do Firefox*

5.3 Terceiro Teste

Uma requisição recebida do *browser* implica em uma solicitação de *download* e logo o gerenciador executará a transferência do arquivo solicitado. Mas no Ourdown, o que se espera é que nem sempre esta tarefa seja executada, pois o Ourdown tem a função de impedir que um *download* de um mesmo arquivo seja executado, caso o arquivo solicitado não esteja sendo feito o *download* no *host* ou em qualquer outro interligado na rede, o *download* do arquivo é feito normalmente.

O teste apresenta resultados positivos quanto ao esperado. Como não havia nenhuma tarefa de *download* no *host* e nem nos outros, o *download* é executado como pode ser visto na Figura 14.

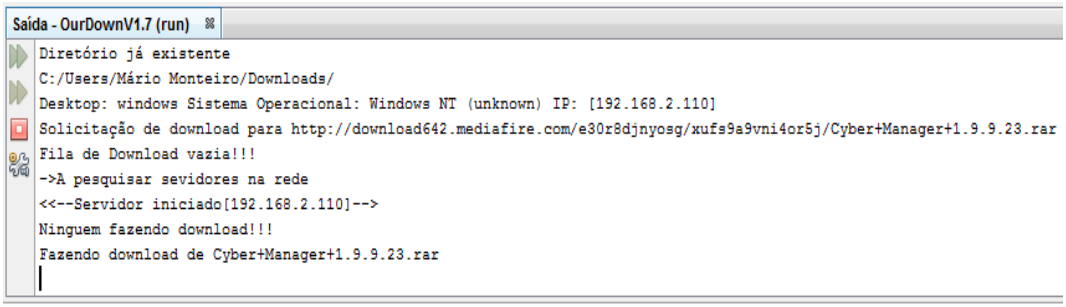

Figura 14 *– Url recebida do Firefox mostra no painel de saída do IDE Netbeans*

5.4 Quarto Teste

A partir do momento que um *download* já se encontra em andamento, é requisitado o *download* do mesmo arquivo no mesmo *host* utilizado. A ideia é de que com este experimento o Ourdown barre a nova requisição não iniciando um novo *download*.

A Figura 15 mostra o painel de saída do IDE Netbeans no momento em que o Ourdown recebe a requisição do *browser*. Ao receber a requisição, o componente ServerURL trata de procurar indícios da existência do *download* ativo do arquivo ou do próprio arquivo no diretório, encontrando-o, o Ourdown não executa o *download*, e portanto com as ações observadas do

software o resultado almejado é alcançado.

***Figura 15** – Solicitação de download negado pelo Ourdown*

5.5 Quinto Teste

Com um *download* sendo descarredado, é realizado uma tentativa de realizar o *download* do mesmo arquivo em outro *host* conectado à mesma rede local. O objetivo é se certificar de que o Ourdown não permita fazer um novo *download* do mesmo arquivo, pois já está sendo feito conforme o terceiro teste e também se haverá comunicação entre os componentes através da rede local.

A Figura 16 mostra o resultado do teste, onde a requisição feita no segundo *host* é encaminhada ao primeiro, afim de verificar a existência do *download*. Como já se sabe que existe, o primeiro *host* responde a solicitação do segundo, implicando dizer que já está a fazer o *download* impedindo que o novo *download* seja iniciado. Assim também conclui-se que a comunicação entre os componente através da rede local funciona, que se dá através dos componentes ServerURL (encaminhar as requisições) e ServerRequest (receber as requisições).

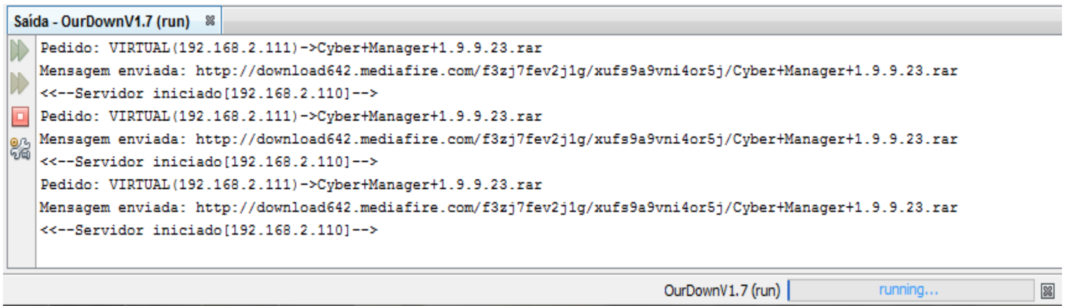

***Figura 16** – Solicitação de download negado pelo Ourdown*

6 Conclusões

Este trabalho aborda o desenvolvimento de um *software* distribuído para gerenciar *downloads* com a função especial de extinguir *downloads* redundantes, em uma rede local, contribua para uma melhor utilização dos enlaces da rede.

O Ourdown é um sistema distribuído cuja função é exercida cooperativamente por componentes complexos constituídos de objetos com funções bem definidas. Seu propósito geral torna a rede local de computadores conscientes da dinamicidade de *downloads* de forma descentralizada. A descentralização está em consonância com o novo modo de utilização dela por usuários acostumados com o compartilhamento e a descentralização de serviços. O *software* foi desenvolvido utilizando a linguagem Java possibilitando a sua utilização em diferentes plataformas de sistemas operacionais. A linguagem possibilita o uso de *Sockets*, que por meio deles é possível que os componentes se comuniquem através da rede local, todavia, o sistema é dependente da arquitetura TCP/IP e utiliza *Broadcast* para que os componentes se encontrem.

Usualmente, *downloads* são solicitados através dos *browsers*, todavia foi desenvolvida uma extensão para o Firefox, para que as requisições fossem transferidas para o Ourdown. Para que isso aconteça, uma nova tecnologia foi conhecida, o XUL (Linguagem de Interface XML), que é a linguagem utilizada para o desenvolvimento do Firefox. As extensões permitem a modificação de funcionalidades e interface do *browser*.

Durante o desenvolvimento do código foram implementados: componentes para assegurar a comunicação entre os componentes; um motor de busca por *downloads* ativos utilizando como método de busca o nome do arquivo e a sua URL; sistema de *cache* local para que fosse executado o papel de *proxy cache*. Porém é encontrada uma limitação pelo fato da possibilidade do usuário excluir ou retirar o arquivo do diretório usado pelo Ourdown, assim poderá atrapalhar a lógica do *software*. O desenvolvimento consumia algumas horas por dia, durante dias. Porém era necessário conciliar o desenvolvimento com outras atividades, o que implica dizer a existência de dificuldades em atender aos prazos estimados.

Atualmente o Ourdown interage apenas com Firefox, tornando-se uma pequena desvantagem em relação a outros gerenciadores de *download*. Futuramente é pretendido estender os trabalhos afim de buscar outras soluções para a integração do componente com outros *browsers*, com a visão da interação ser “universal”. É considerável destacar o aperfeiçoamento da interface do *software* deixando-a mais atrativa para os usuários. A solução é buscar uma melhor compreensão dos conceitos e fundamentos de IHC (Interação Humano-Computador).

Referências Bibliográficas

CISCO SYSTEMS. [S.l.], 2011. Disponível em: <http://www.cisco.com>.

COSTA, Daniel Gouveia. *Java em Rede: Recursos avançados de programação*. [S.l.]: Brasport, 2008.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Sistemas Distribuídos: Conceitos e Projeto. 4ª Edição*. [S.l.]: Bookman, 2007.

DEITEL, H. M. *Java TM: Como programar. 6ª Edição*. [S.l.]: Pearson Prentice Hall, 2005.

FUNDAÇÃO MOZILLA. [S.l.], 2011. Disponível em: <http://www.mozilla.org>.

IBOPE MEDIA. [S.l.], 2011. Disponível em: <http://www.ibope.com.br>.

INTERNET DOWNLOAD MANAGER. [S.l.], 2011. Disponível em: <http://www.internetdownloadmanager.com>.

INTHURN, C. *Qualidade & teste de software*. [S.l.]: Visual Books, 2001.

JDOWNLOADER. [S.l.], 2011. Disponível em: <http://jdownloader.org/>.

ORBIT. [S.l.], 2011. Disponível em: <http://www.orbitdownloader.com/>.

SOARES, Luiz Fernando G. *Redes de Computadores: Das LANs, MANs e WANs às Redes ATM 2ª Edição*. [S.l.]: Elsevier, 1995.

TANENBAUM, Adrew S. *Redes de Computadores. 4ª Edição*. [S.l.]: Elsevier, 2003.

TANENBAUM, Adrew S. *Sistemas Distribuídos: princípios e paradigmas. 2ª Edição*. [S.l.]: Pearson Prentice Hall, 2007.

APÊNDICE A – Class Config

```
1  package ourdown;
2
3  import java.io.File;
4  import java.io.RandomAccessFile;
5  import java.net.*;
6  import java.util.Enumeration;
7  import java.util.Vector;
8  import java.util.logging.Level;
9  import java.util.logging.Logger;
10 import javax.print.DocFlavor;
11
12 public class Config extends Thread{
13   OurDown od;
14   //define porta padrão para transferencia de mensagens entre os hosts na rede
15   static int portTransfMensagem= 5000;
16   //define porta padrão para transferencia de arquivos entre os hosts na rede
17   static int portTransfArquivo = 6000;
18   //define porta do escutador de url's do browser
19   static int portServerUrl = 2121;
20   //Construtor da Classe
21   public Config(OurDown ourDown){
22     od = ourDown;
23   }
24   //Verifica se o IP é válido
25   public static boolean verfIp(String ip){
26   boolean retorno = false;
27   String teste = "0123456789";
28   //quebra o endereço IP onde tiver '.'
29   String[] ipSplit;
30   //Se no endereço IP tiver ':', ele é inválido para a aplicação
31   if(ip.contains(":")){
32     retorno = false;
33   }else{
34     int count = 0;
35     //quebra o endereço IP onde tiver '.'
```

```
36      ipSplit = ip.split(".");
37      for(int i=0;i<ipSplit.length;i++){
38         //Verifica se no endereço IP só contém números
39         if(ipSplit[i].contains(teste)){
40           //para cada numero encontrado count é incrementada
41           count++;
42         }
43      }
44      //Após quebra o endereço IP, se o mesmo apresentar quatro partes
45      //e o número de números for >= 4 ou <= 12 o IP é válido para a aplicação
46      if(ipSplit.length!=4&&(count>=4||count<=12)){
47        retorno = true;
48      }
49    }
50    return retorno;
51  }
52    //Obtem os endereços IPs atribuidos à máquina
53    public void getIpLocal(){
54      Enumeration nis = null;
55      try {
56        //obtém propriedades das interfaces de rede da máquina
57        nis = NetworkInterface.getNetworkInterfaces();
58      }catch (SocketException e) {
59        e.printStackTrace();
60      }
61      while (nis.hasMoreElements()) {
62        NetworkInterface ni = (NetworkInterface) nis.nextElement();
63        Enumeration ias = ni.getInetAddresses();
64        InetAddress name = null;
65        try {
66          name = InetAddress.getLocalHost();
67        } catch (UnknownHostException ex) {
68          ex.printStackTrace();
69        }
70        while (ias.hasMoreElements()) {
71          InetAddress ia = (InetAddress) ias.nextElement();
72          if(verfIp(ia.getHostAddress())&&
73            !ia.getHostName().equals("localhost")&&
74            !ia.getHostName().equals("127.0.0.1")){
75            od.ipLocal.add(ia.getHostAddress());
76          }
77        }
78      }
```

```
79    }
80    //Define o diretório para downloads de acordo com o sistema operacional
81    public String getDiretorio(){
82      String user = new String(System.getProperty("user.home"));
83      char[] dir = user.toCharArray();
84      //Torca a '\' (barra invertida) por '/'
85      for(int i=0;i<dir.length;i++){
86        if(dir[i]=='\\')
87          dir[i]='/';
88      }
89      String diretorio = new String(dir)+"/Downloads/";
90      return diretorio;
91    }
92    //Cria pasta Downloads e mostra valores como So, nome do host e endereço ip
93    public void run(){
94      getIpLocal();
95      String desktop = System.getProperty("sun.desktop");
96      String os = System.getProperty("os.name");
97      File diretorio = new File(getDiretorio());
98      //se o diretorio ainda não foi criado
99      if (!diretorio.exists()) {
100       //mkdir() cria somente um diretório
101       diretorio.mkdirs();
102       System.out.println(getDiretorio());
103     }else{
104       System.out.println("Diretório já existente");
105       System.out.println(getDiretorio());
106     }
107     InetAddress ip;
108     try {
109       ip = InetAddress.getLocalHost();
110       //retorna para a tela do gerenciador os valores
111       //com SO, nome do host e ip
112       System.out.println("Desktop: "+desktop+
113       " Sistema Operacional: "+os+" IP: "+od.ipLocal);
114       od.ods.propriedadeSistema.setText(desktop+"/"+os+"/"+od.ipLocal);
115     } catch (UnknownHostException ex) {
116       Logger.getLogger(Config.class.getName()).log(Level.SEVERE, null, ex);
117     }
118   }
119 }
```

APÊNDICE B – Class Download

```
1  package ourdown;
2
3  import java.io.*;
4  import java.net.*;
5  import java.util.Iterator;
6  import java.util.Observable;
7
8  public class Download extends Observable implements Runnable{
9    //Tamanho máximo do buffer
10   private static final int MAX_BUFFER_SIZE = 1024;
11   //Estados do Download
12   public static final String STATUS[] =
13      {"Downloading","Pausado","Completo","Cancelado","Erro"};
14   public static final int DOWNLOADING = 0;
15   public static final int PAUSADO = 1;
16   public static final int COMPLETO = 2;
17   public static final int CANCELADO = 3;
18   public static final int ERRO = 4;
19   private URL url; //url do download
20   private int size;//tamanho do download em bytes;
21   private int downloaded;//numero de byte baixados
22   private int status;//estado atual do download
23
24   OurDown od;
25   TratamentoDeUrl trataUrl;
26   Config config;
27   String diretorio = "";
28   //inicia o download
29   public Download(URL url, OurDown ourDown){
30     this.url = url;
31     size = -1;
32     downloaded = 0;
33     status = DOWNLOADING;
34     od = ourDown;
35     trataUrl = new TratamentoDeUrl();
```

```
36      download();
37      diretorio = new Config(ourDown).getDiretorio();
38    }
39    //retorna a url do download
40    public String getUrl(){
41      return url.toString();
42    }
43    //retorna o tamanho do download
44    public int getSize(){
45    return size;
46    }
47    //retorna o progresso do download
48    public float getProgress(){
49      return ((float)downloaded/size)*100;
50    }
51    //retorna o status do download
52    public int getStatus(){
53      return status;
54    }
55    //pausa o download
56    public void pause(){
57      status = PAUSADO;
58      stateChanged();
59    }
60    //reinicia o download
61    public void resume(){
62      status = DOWNLOADING;
63      stateChanged();
64      download();
65    }
66    //cancela o download
67    public void cancel(){
68      status = CANCELADO;
69      Iterator index = od.FilaDownload.iterator();
70      while(index.hasNext()){
71        if(trataUrl.ComparaUrl(od, getUrl())){
72          //remove a url de FilaDownload
73          od.FilaDownload.remove(getUrl());
74        }
75      }
76      stateChanged();
77    }
78    //retorna possível erro no download
```

```
79    private void error(){
80      status = ERRO;
81      stateChanged();
82    }
83    //inicia ou continua um download
84    private void download(){
85      Thread thread = new Thread(this);
86      thread.start();
87    }
88    //obtem o nome do arquivo
89    public String getFileName(URL url){
90      String fileName = url.getFile();
91       return fileName.substring(fileName.lastIndexOf("/")+1);
92    }
93
94    @Override
95    public void run() {
96      RandomAccessFile file = null;
97      InputStream stream = null;
98      try{
99        //abre a conexão para a url
100       HttpURLConnection connection = (HttpURLConnection) url.openConnection();
101       //especifica qual porção do arquivo a ser baixada
102       connection.setRequestProperty("Range", "bytes="+downloaded+"-");
103       //conecta ao servidor
104       connection.connect();
105       //assegura o codigo de resposta na gama de 200
106       if(connection.getResponseCode() / 100 != 2){
107         error();
108       }
109       //assegura se o tamanho do conteudo e valido
110       int contentLength = connection.getContentLength();
111       if (contentLength < 1){
112         error();
113       }
114       //Define o tamanho do download caso não seja definido
115       if(size == -1){
116         size = contentLength;
117         stateChanged();
118       }
119
120       file = new RandomAccessFile(diretorio+getFileName(url),"rw");
121       //Tamanho do buffer de acordo com o tamanho do arquivo baixado
```

```
122         stream = connection.getInputStream();
123         while(status == DOWNLOADING){
124           byte buffer[];
125           if(size - downloaded > MAX_BUFFER_SIZE){
126             buffer = new byte[MAX_BUFFER_SIZE];
127           }else{
128             buffer = new byte[size - downloaded];
129           }
130           //le do servidor dentro do buffer
131           int read = stream.read(buffer);
132           if(read == -1)
133             break;
134           //escreve do buffer para o arquivo
135           file.write(buffer,0,read);
136           downloaded += read;
137           stateChanged();
138         }
139         //Altera o status para completo se este ponto foi
140         //alcancado porque o download concluiu
141         if(status == DOWNLOADING){
142           status = COMPLETO;
143           stateChanged();
144         }
145         //Se o status é completo, então enviará o arquivo caso haja
146         //alguma requisição
147         if(status==COMPLETO){
148           SendFile sf = new SendFile(this, od);
149           sf.start();
150         }
151       }catch(Exception e){
152          error();
153       }finally{
154          if(file!=null){ //fecha o arquivo
155            try{
156              file.close();
157            }catch(Exception e){}
158       }
159       if(stream!=null){ //fecha a conexão com o servidor
160         try{
161           stream.close();
162         }catch(Exception e){}
163       }
164     }
```

```
165   }
166    //notifica observer que o status do download foi alterado
167    private void stateChanged(){
168      setChanged();
169      notifyObservers();
170    }
171  }
```

APÊNDICE C – Class FileReceive

```
1  package ourdown;
2
3  import java.io.BufferedOutputStream;
4  import java.io.FileOutputStream;
5  import java.io.InputStream;
6  import java.io.ObjectInputStream;
7  import java.net.ServerSocket;
8  import java.net.Socket;
9
10 public class FileReceive extends Thread{
11   ServerSocket servidor;
12   OurDown od;
13   public FileReceive(OurDown down){
14     od = down;
15   }
16   public void run() {
17     try {
18       System.out.println("<<--Servidor iniciado para +
19         "recepção de arquivos-->>");
20       servidor = new ServerSocket(Config.portTransfArquivo);
21       //inicia um thread para espera por transferencia
22       //de algum arquivo requisitado
23       WaitFile();
24     } catch (Exception e) {
25       e.printStackTrace();
26       System.exit(1);
27     }
28   }
29   public void WaitFile(){
30     try{
31       while(true){
32         //sempre estára a espera de um arquivo
33         Socket ss = servidor.accept();
34         ReceiveFile rf = new ReceiveFile(ss,od);
35         rf.start();
```

```
36        }
37      }catch(Exception e){
38        e.printStackTrace();
39        }
40    }
41
42    class ReceiveFile extends Thread {
43      Socket sock;
44      Config config;
45      OurDown od;
46      public ReceiveFile(Socket _s, OurDown down) {
47        sock = _s;
48        config = new Config(down);
49        od = down;
50      }
51      public void run() {
52        // Receber o Arquivo
53        // filesize temporary hardcoded
54        int filesize = 6022386;
55        long start = System.currentTimeMillis();
56        int bytesRead;
57        int current = 0;
58        // receive file
59        byte[] mybytearray = new byte[filesize];
60        try {
61          ObjectInputStream ois = new ObjectInputStream(sock.getInputStream());
62          String nome_arquivo = (String) ois.readObject();
63          InputStream is = sock.getInputStream();
64          //gravará o arquivo no diretorio escolhido pelo gerenciador
65          FileOutputStream fos =
66                  new FileOutputStream(config.getDiretorio()+nome_arquivo);
67          BufferedOutputStream bos = new BufferedOutputStream(fos);
68          bytesRead = is.read(mybytearray, 0, mybytearray.length);
69          current = bytesRead;
70          do {
71            bytesRead =
72            is.read(mybytearray, current, (mybytearray.length - current));
73            if (bytesRead >= 0) {
74              current += bytesRead;
75            }
76          } while (bytesRead > -1);
77          bos.write(mybytearray, 0, current);
78          bos.flush();
```

```
79            long end = System.currentTimeMillis();
80            System.out.println(end - start);
81            od.ods.textArea.append("\n|> "+nome_arquivo+
82                    " recebido com sucesso - "+(end-start)+"ms");
83            bos.close();
84            sock.close();
85          } catch (Exception e) {
86            e.printStackTrace();
87          }
88       }
89     }
90   }
```

APÊNDICE D – Class OurDown

```
1  package ourdown;
2
3  import java.net.MalformedURLException;
4  import java.net.URL;
5  import java.util.Iterator;
6  import java.util.Vector;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9  //Classe principal do gerenciador
10 //Aqui iniciará todos os componentes do sistema
11 public class OurDown {
12   public Vector FilaDownload = new Vector();
13   public Vector FilaPedido = new Vector();
14   public Vector ipLocal = new Vector();
15   ServerURL serverURL = new ServerURL(this);
16   ServerRequest serverRequest = new ServerRequest(this);
17   OurDownScreen ods = new OurDownScreen(this);
18   FileReceive fileReceive = new FileReceive(this);
19   Config config = new Config(this);
20   public static void main(String[] args) {
21     OurDown od = new OurDown();
22     od.config.start();
23     od.ods.show();
24     od.serverURL.start();
25     od.fileReceive.start();
26     od.serverRequest.start();
27   }
28 }
```

APÊNDICE E – Class OurDownScreen

```
1  package ourdown;
2
3  import java.awt.BorderLayout;
4  import java.awt.Color;
5  import java.awt.event.ActionEvent;
6  import java.awt.event.ActionListener;
7  import java.awt.event.KeyEvent;
8  import java.net.URL;
9  import java.util.Observable;
10 import java.util.Observer;
11 import javax.swing.*;
12 import javax.swing.event.ListSelectionEvent;
13 import javax.swing.event.ListSelectionListener;
14 //Nesta classe é implementada toda a interface do sistema
15 public class OurDownScreen extends JFrame implements Observer{
16   // Adiciona campo de texto
17   public JTextField addTextField;
18   //Adiciona uma área de texto
19   public JTextArea textArea;
20   //Adiciona um label
21   public JLabel propriedadeSistema;
22   private TabelaDownloads tableModel;
23   // Tabela listando os downloads
24   private JTable table;
25   //Botoes usados para gerenciar os downloads
26   private JButton pauseButton, resumeButton;
27   //Seleciona o download atual
28   private JButton cancelButton, clearButton;
29   //Sinaliza quando a linha da tabela é selecionada ou não
30   private Download selectedDownload;
31   private boolean clearing;
32   OurDown od;
33   public OurDownScreen(OurDown ourDown){
34     od = ourDown;
35     setTitle("OurDown v1.7");
```

```
36      setSize(800, 600);
37      setDefaultCloseOperation(this.EXIT_ON_CLOSE);
38      //Implementa barra de menu e opções do menu
39      //cria uma barra de menu
40      JMenuBar menuBar = new JMenuBar();
41      //cria a opção arquivo no menu
42      JMenu fileMenu = new JMenu("Arquivo");
43      fileMenu.setMnemonic(KeyEvent.VK_F);
44      //cria a opção sair
45      JMenuItem fileExitMenuItem =
46        new JMenuItem("Sair", KeyEvent.VK_X);
47      fileExitMenuItem.addActionListener(new ActionListener() {
48        public void actionPerformed(ActionEvent e) {
49          actionExit();
50        }});
51      //adiciona a opção sair ao menu
52      fileMenu.add(fileExitMenuItem);
53      menuBar.add(fileMenu);
54      setJMenuBar(menuBar);
55      //Define a tabela de downloads
56      tableModel = new TabelaDownloads();
57      table = new JTable(tableModel);
58      table.getSelectionModel().addListSelectionListener(
59        new ListSelectionListener() {
60          @Override
61          public void valueChanged(ListSelectionEvent e) {
62            tableSelectionChanged();
63          }});
64      //permite somente uma linha ser selecionada por vez
65      table.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
66      //Configura ProgressBar como renderer para coluna de progresso
67      Progresso renderer = new Progresso(0, 100);
68      renderer.setStringPainted(true);
69      table.setDefaultRenderer(JProgressBar.class, renderer);
70      //define a largura da linha na tabela
71      table.setRowHeight((int)renderer.getPreferredSize().getHeight());
72      //define painel de downloads
73      JPanel downloadsPanel = new JPanel();
74      downloadsPanel.setBorder(BorderFactory.createTitledBorder("Downloads"));
75      downloadsPanel.setLayout(new BorderLayout());
76      downloadsPanel.add(new JScrollPane(table),BorderLayout.CENTER);
77      //Define o painel de botoes
78      JPanel buttonsPanel = new JPanel();
```

```
79      pauseButton = new JButton("Pausar");
80      pauseButton.addActionListener(new ActionListener() {
81        @Override
82        public void actionPerformed(ActionEvent e) {
83          actionPause();
84        }});
85      pauseButton.setEnabled(false);
86      buttonsPanel.add(pauseButton);
87      resumeButton = new JButton("Recomeçar");
88      resumeButton.addActionListener(new ActionListener() {
89        @Override
90        public void actionPerformed(ActionEvent e) {
91          actionResume();
92        }});
93      resumeButton.setEnabled(false);
94      buttonsPanel.add(resumeButton);
95      cancelButton = new JButton("Cancelar");
96      cancelButton.addActionListener(new ActionListener() {
97        @Override
98        public void actionPerformed(ActionEvent e) {
99          actionCancel();
100       }});
101     cancelButton.setEnabled(false);
102     buttonsPanel.add(cancelButton);
103     clearButton = new JButton("Limpar");
104     clearButton.addActionListener(new ActionListener() {
105       @Override
106       public void actionPerformed(ActionEvent e) {
107         actionClear();
108       }});
109     clearButton.setEnabled(false);
110     buttonsPanel.add(clearButton);
111     downloadsPanel.add(
112       new JScrollPane(buttonsPanel),BorderLayout.SOUTH);
113     //texteArea mostra as mensagens e detalhes do sistema
114     textArea =
115       new JTextArea("|-------------|DETALHES DO SISTEMA|-------------|");
116     JPanel prompt = new JPanel();
117     prompt.setBorder(BorderFactory.createEtchedBorder());
118     prompt.setLayout(new BorderLayout());
119     prompt.add(new JScrollPane(textArea), BorderLayout.CENTER);
120     //Mostra as propriedades do sistema desktop/So/Nome do host/endereço ip
121     propriedadeSistema = new JLabel();
```

```
122     JPanel detalhes = new JPanel();
123     detalhes.add(propriedadeSistema,BorderLayout.CENTER);
124     //Posicionamento dos paineis
125     getContentPane().setLayout(new BorderLayout());
126     getContentPane().add(detalhes, BorderLayout.SOUTH);
127     getContentPane().add(downloadsPanel, BorderLayout.CENTER);
128     getContentPane().add(prompt,BorderLayout.WEST);
129   }
130   //Actions dos botoes
131   private void actionExit(){ //Sair do programa
132     System.exit(0);
133   }
134   //addDownload é chamada em ServerURL
135   public void addDownload(String url){
136     URL verifiedUrl = verificaUrl(url);
137     if(verifiedUrl!=null){
138       tableModel.addDownload(new Download(verifiedUrl,od));
139     }else{
140       JOptionPane.showMessageDialog(
141         this, "URL Invalida!!!", "Error", JOptionPane.ERROR_MESSAGE);
142     }
143   }
144   //Pausa o download selecionado
145   private void actionPause(){
146     selectedDownload.pause();
147     updateButtons();
148   }
149   //Continua o download selecionado
150   private void actionResume(){
151     selectedDownload.resume();
152     updateButtons();
153   }
154   //Cancela o download selecionado
155   private void actionCancel(){
156     selectedDownload.cancel();
157     updateButtons();
158   }
159   //limpa o download selecionado
160   private void actionClear(){
161     clearing = true;
162     tableModel.clearDownload(table.getSelectedRow());
163     clearing = false;
164     selectedDownload = null;
```

```
165       updateButtons();
166     }
167     //atualiza cada estado dos botões baseado no atual download selecionado
168     private void updateButtons(){
169       if(selectedDownload!=null){
170         int status = selectedDownload.getStatus();
171         switch(status){
172           case Download.DOWNLOADING:
173             pauseButton.setEnabled(true);
174             resumeButton.setEnabled(false);
175             cancelButton.setEnabled(true);
176             clearButton.setEnabled(false);
177             break;
178           case Download.PAUSADO:
179             pauseButton.setEnabled(false);
180             resumeButton.setEnabled(true);
181             cancelButton.setEnabled(true);
182             clearButton.setEnabled(false);
183             break;
184           case Download.ERRO:
185             pauseButton.setEnabled(false);
186             resumeButton.setEnabled(true);
187             cancelButton.setEnabled(false);
188             clearButton.setEnabled(true);
189             break;
190           //Completo ou cancelado;
191           default:
192             pauseButton.setEnabled(false);
193             resumeButton.setEnabled(false);
194             cancelButton.setEnabled(false);
195             clearButton.setEnabled(true);
196             break;
197         }
198       }else{
199           pauseButton.setEnabled(false);
200           resumeButton.setEnabled(false);
201           cancelButton.setEnabled(false);
202           clearButton.setEnabled(false);
203       }
204     }
205     //valida a url
206     //Verifica a url para download
207     private URL verificaUrl(String url){
```

```
208       //permite somente url's http
209       if(!url.toLowerCase().startsWith("http://"))
210         return null;
211       URL verifiedUrl = null;
212       try{
213         verifiedUrl = new URL(url);
214       }catch(Exception e){
215         return null;
216       }
217       if(verifiedUrl.getFile().length()<2)
218         return null;
219       return verifiedUrl;
220     }
221     //a função abaixo e chamada quando a seleção na tabela e alterada
222     private void tableSelectionChanged(){
223       //Notificações não registradas do ultimo download selecionado
224       if(selectedDownload!=null)
225         selectedDownload.deleteObserver(OurDownScreen.this);
226       if(!clearing){
227         selectedDownload = tableModel.getDownload(table.getSelectedRow());
228         selectedDownload.addObserver(OurDownScreen.this);
229         updateButtons();
230       }
231     }
232     //update é chamado quando um download notifica o observer dele
233     //de alguma alteração
234     @Override
235     public void update(Observable o, Object arg) {
236       //atualiza os botoes se o download selecionado foi alterado
237       if(selectedDownload!=null && selectedDownload.equals(o))
238       updateButtons();
239     }
240   }
```

APÊNDICE F – Class Progresso

```
1  package ourdown;
2
3  import java.awt.Component;
4  import javax.swing.JProgressBar;
5  import javax.swing.JTable;
6  import javax.swing.table.TableCellRenderer;
7
8  public class Progresso extends
9        JProgressBar implements TableCellRenderer {
10         public Progresso(int min, int max){
11           super(min, max);
12         }
13
14        public Component getTableCellRendererComponent(J
15          Table table, Object value,
16             boolean isSelected, boolean hasFocus, int row, int column) {
17               setValue((int)((Float)value).floatValue());
18             return this;
19        }
20 }
```

APÊNDICE G – Class SendFile

```
1  package ourdown;
2
3  import java.io.*;
4  import java.net.*;
5  import java.util.Iterator;
6  import java.util.logging.Level;
7  import java.util.logging.Logger;
8
9  public class SendFile extends Thread{
10   Download download;
11   OurDown od;
12   TratamentoDeUrl trataUrl;
13   Config config;
14   public SendFile(Download d,OurDown ourDown){
15     download = d;
16     od = ourDown;
17     trataUrl = new TratamentoDeUrl();
18     config = new Config(ourDown);
19   }
20   public void run(){
21     try {
22       System.out.println("Download de"+download.getFileName(
23         new URL(download.getUrl()))+" terminado");
24       od.ods.textArea.append("\n|> Download de"+download.getFileName(
25         new URL(download.getUrl()))+" terminado");
26       if(od.FilaPedido.isEmpty()){
27         System.out.println("Fila de pedidos vazia!");
28       }else{
29         Iterator <DatagramPacket> pedido = od.FilaPedido.iterator();
30         //procura o arquivo referente ao lista de pedidos
31         while(pedido.hasNext()){
32           DatagramPacket dp = pedido.next();
33           String file_request = trataUrl.getNameFile(new String(dp.getData()));
34           String file_download = trataUrl.getNameFile(download.getUrl());
35           if(file_request.equals(file_download)){
```

```
36              System.out.println("Cliente: "+dp.getAddress().getHostAddress()+
37                " pedido: "+download.getUrl());
38              od.ods.textArea.append("\n|> Cliente: "
39                +dp.getAddress().getHostAddress()+" pedido: "+download.getUrl());
40              Send(download, dp);
41              //quando o pedido for enviado, o mesmo será deletado
42              od.FilaPedido.remove(dp);
43            }
44          }
45        }
46      }catch (Exception e) {
47        e.printStackTrace();
48      }
49    }
50    public void Send(Download d, DatagramPacket packet){
51      try {
52        Socket sock = new Socket(packet.getAddress().getHostAddress(), 6000);
53        String arquivo = trataUrl.getNameFile(download.getUrl());
54        //Enviando o nome do arquivo
55        System.out.println("Enviando "+arquivo+
56          " para "+packet.getAddress().getHostAddress());
57        od.ods.textArea.append("\n|> Enviando "+
58          arquivo+" para "+ packet.getAddress().getHostName()+"("
59            +packet.getAddress().getHostAddress()+")");
60        ObjectOutputStream oos = new ObjectOutputStream(sock.getOutputStream());
61        oos.writeObject(arquivo);
62        File myFile = new File(config.getDiretorio()+arquivo);
63        byte[] mybytearray = new byte[(int) myFile.length()];
64        FileInputStream fis = new FileInputStream(myFile);
65        BufferedInputStream bis = new BufferedInputStream(fis);
66        bis.read(mybytearray, 0, mybytearray.length);
67        OutputStream os = sock.getOutputStream();
68        System.out.println("Enviando...");
69        os.write(mybytearray, 0, mybytearray.length);
70        os.flush();
71        sock.close();
72      } catch (Exception e) {
73        e.printStackTrace();
74        System.exit(0);
75      }
76    }
77  }
```

APÊNDICE H – Class ServerRequest

```
1  package ourdown;
2
3  import java.io.*;
4  import java.net.DatagramPacket;
5  import java.net.DatagramSocket;
6  import java.net.InetAddress;
7  import java.net.Socket;
8  import java.util.Iterator;
9  import sun.awt.image.OffScreenImage;
10
11 class ServerRequest extends Thread{
12   OurDown od;
13   String message;
14   InetAddress ip_local, ip_cliente;
15   TratamentoDeUrl trataUrl;
16   Config config;
17   public ServerRequest(OurDown ourDown){
18     od = ourDown;
19     trataUrl = new TratamentoDeUrl();
20     config = new Config(ourDown);
21   }
22   public boolean ComparaIP(InetAddress ip){
23     Iterator index = od.ipLocal.iterator();
24     int count=0;
25     while(index.hasNext()){
26       if(ip.getHostAddress().equals(index.next())){
27         count++;
28       }
29     }
30     if(count>0)
31       return true;
32     return false;
33   }
34   public void run(){
35     String message;
```

```
36      try{
37        DatagramSocket socket =
38          new DatagramSocket(Config.portTransfMensagem);
39        ip_local = InetAddress.getLocalHost();
40        while(true){
41          byte[] buffer = new byte[1024];
42          System.out.println("<<--Servidor iniciado"
43            +od.ipLocal+"-->");
44          DatagramPacket packet =
45            new DatagramPacket(buffer, buffer.length);
46          socket.receive(packet);
47          ip_cliente = packet.getAddress();
48          int client_port = packet.getPort();
49          message =
50            new String(packet.getData()).substring(0,
51              packet.getLength());
52          if(!ComparaIP(ip_cliente)){
53            System.out.println("Pedido: "+
54              ip_cliente.getHostName()+
55                "("+ip_cliente.getHostAddress()+")->"+
56                   trataUrl.getNameFile(message));
57            if(!od.FilaDownload.isEmpty()){
58              if(trataUrl.ComparaUrl(od, message)){
59                od.ods.textArea.append("\n|> Pedido: "+
60                  ip_cliente.getHostName()+"("+
61                    ip_cliente.getHostAddress()+
62                      ")->"+trataUrl.getNameFile(message));
63                buffer = message.getBytes();
64                packet =
65                  new DatagramPacket(buffer, buffer.length,
66                      ip_cliente, client_port);
67                socket.send(packet);
68                od.FilaPedido.add(packet);
69                System.out.println("Mensagem enviada: "+
70                  new String(packet.getData()));
71              }
72            }else{
73               String arquivo = trataUrl.getNameFile(message);
74               File file =
75                 new File(config.getDiretorio()+
76                   trataUrl.getNameFile(message));
77               if(file.exists()){
78                 System.out.println("Arquivo salvo no diretorio");
```

```
79                  od.ods.textArea.append("\n|> Pedido: "+
80                    ip_cliente.getHostName()+"("+
81                      ip_cliente.getHostAddress()+")->"+
82                        trataUrl.getNameFile(message));
83                  buffer = message.getBytes();
84                  packet =
85                     new DatagramPacket(buffer, buffer.length,
86                       ip_cliente, client_port);
87                  socket.send(packet);
88                  try {
89                    Socket sock =
90                      new Socket(packet.getAddress().getHostAddress(), 6000);
91                    //Enviando o nome do arquivo
92                    od.ods.textArea.append("\n|> Enviando "
93                      +arquivo+" para "+packet.getAddress().getHostName()+
94                        "("+packet.getAddress().getHostAddress()+")");
95                    ObjectOutputStream oos =
96                      new ObjectOutputStream(sock.getOutputStream());
97                    oos.writeObject(arquivo);
98                    byte[] mybytearray = new byte[(int) file.length()];
99                    FileInputStream fis = new FileInputStream(file);
100                   BufferedInputStream bis = new BufferedInputStream(fis);
101                   bis.read(mybytearray, 0, mybytearray.length);
102                   OutputStream os = sock.getOutputStream();
103                   System.out.println("Enviando...");
104                   os.write(mybytearray, 0, mybytearray.length);
105                   os.flush();
106                   sock.close();
107                 } catch (Exception e) {
108                   e.printStackTrace();
109                   System.exit(0);
110                 }
111                 System.out.println("Mensagem enviada 2: "+
112                   new String(packet.getData()));
113               }
114             }
115           }
116       }
117     }catch(Exception e){
118       e.printStackTrace();
119     }
120   }
121 }
```

APÊNDICE I – Class ServerURL

```
1  package ourdown;
2
3  import java.io.*;
4  import java.net.*;
5  import java.util.logging.Level;
6  import java.util.logging.Logger;
7  import sun.nio.ch.DatagramSocketAdaptor;
8
9  class ServerURL extends Thread{
10   OurDown od;
11   TratamentoDeUrl trataUrl;
12   Config config;
13   public ServerURL(OurDown ourDown){
14     od = ourDown;
15     trataUrl = new TratamentoDeUrl();
16     config = new Config(ourDown);
17   }
18   ServerSocket serverSocket;
19   public void run(){
20     try{
21       //cria um socket servidor não acoplado
22       serverSocket = new ServerSocket();
23       //Vincula o ServerSocket para um endereço
24       //específico (endereço IP e número de porta).
25       serverSocket.bind(new InetSocketAddress(
26         "localhost",Config.portServerUrl));
27       Listen();
28     }catch(Exception e){
29       e.printStackTrace();
30     }
31   }
32   //Aguarda requisições do navegador
33   public void Listen(){
34   try{
35     //espera eternamente por mensagens do navegador
```

```
36      while(true){
37        Socket socket = serverSocket.accept();
38        BrownserRequest br = new BrownserRequest(socket);
39        br.start();
40      }
41    }catch(Exception e){
42      e.printStackTrace();
43    }
44  }
45  //Trata das requisições do navegador
46  class BrownserRequest extends Thread{
47    Socket cliente;
48    private BrownserRequest(Socket socket){
49      cliente = socket;
50    }
51    public void run(){
52      try{
53        byte [] b = new byte[200];
54        DataInputStream dataInputStream =
55          new DataInputStream(cliente.getInputStream());
56        dataInputStream.readFully(b);
57        String url;
58        url = new String(b);
59        url = url.substring(url.indexOf(
60          "http://"),url.indexOf(" HTTP/1.1"));
61        System.out.println("Solicitação de download para "+url);
62        od.ods.textArea.append("\n|> Solicitação de download para "+
63          trataUrl.getNameFile(url));
64        SolicitacaoDeDownload sdd = new SolicitacaoDeDownload(url);
65        sdd.start();
66      }catch(Exception e){
67        e.printStackTrace();
68      }
69    }
70    class SolicitacaoDeDownload extends Thread{
71      String url;
72      public SolicitacaoDeDownload(String requisicao){
73        url = requisicao;
74      }
75      public void run(){
76        DatagramSocket ds = null;
77        //Verifica se o vetor de url's está vazia
78        if(od.FilaDownload.isEmpty()){
```

```
79          try{
80            System.out.println("Fila de Download vazia!!!");
81            //criar um datagrama socket
82            ds = new DatagramSocket();
83            //cria um buffer do tamanho da url
84            byte[] bufout = new byte[url.length()];
85            //prepara pacote para ser enviado em broadcast
86            InetAddress ip = InetAddress.getByName("255.255.255.255");
87            bufout = url.getBytes();
88            //prepara um datagrama
89            DatagramPacket packet =
90              new DatagramPacket(bufout, bufout.length,
91                ip, Config.portTransfMensagem);
92            ds.send(packet);//envia uma datagrama na rede
93            if(ds==null){}
94            System.out.println("->A pesquisar sevidores na rede");
95            od.ods.textArea.append("\n|> Requisitando sevidores na rede");
96            //Timeout de 2s para a espera de resposta
97            ds.setSoTimeout(5000);
98            //cria um novo buffer
99            byte [] bufin = new byte[1024];
100           DatagramPacket dsin = new DatagramPacket(bufin, bufin.length);
101           ds.receive(dsin);
102           InetAddress inetAddress = dsin.getAddress();
103           System.out.println("Servidor->"+
104             inetAddress.getHostName()+"("+inetAddress.getHostAddress()+"): "
105               +trataUrl.getNameFile(new String(dsin.getData()).substring(
106                  0,dsin.getLength()))+" encontrado");
107           od.ods.textArea.append("\n|> "+
108             inetAddress.getHostName()+
109               "("+inetAddress.getHostAddress()+"): "+
110                 trataUrl.getNameFile(new String(
111                   dsin.getData()).substring(
112                     0,dsin.getLength()))+" encontrado");
113            ds.close();
114            DataOutputStream dataOutputStream;
115            dataOutputStream =
116              new DataOutputStream(cliente.getOutputStream());
117            dataOutputStream.write((
118              "Download de "+trataUrl.getNameFile(url)+
119                 " em "+inetAddress.getHostName()+"("+
120                   inetAddress.getHostAddress()+")").getBytes());
121            dataOutputStream.flush();
```

```
122             dataOutputStream.close();
123           }catch(SocketTimeoutException ste){
124             System.out.println("Ninguem fazendo download!!!");
125             od.FilaDownload.add(url);
126             od.ods.addDownload(url);
127             System.out.println(
128               "Fazendo download de "+trataUrl.getNameFile(url));
129             od.ods.textArea.append(
130               "\n|> Fazendo download de "+trataUrl.getNameFile(url));
131             try{
132               DataOutputStream dataOutputStream;
133               dataOutputStream =
134                 new DataOutputStream(cliente.getOutputStream());
135               dataOutputStream.write((
136                 "Fazendo o download de: "+
137                    trataUrl.getNameFile(url)).getBytes());
138               dataOutputStream.flush();
139               dataOutputStream.close();
140             }catch(IOException ex) {
141               Logger.getLogger(
142                 ServerURL.class.getName()).log(
143                   Level.SEVERE, null, ex);
144             }
145           }catch(IOException ioe){
146              ioe.printStackTrace();
147           }
148         }else{
149           //Verifica se ja existe o download do arquivo na máquina
150           if(trataUrl.ComparaUrl(od, url)){
151             System.out.println("Achei!!!");
152             od.ods.textArea.append(
153               "\n|> Download de "+trataUrl.getNameFile(url)+
154                  " existente");
155             try {
156               DataOutputStream dataOutputStream;
157               dataOutputStream =
158                new DataOutputStream(cliente.getOutputStream());
159               dataOutputStream.write((
160                 "Download de "+trataUrl.getNameFile(url)+
161                    " existente").getBytes());
162               dataOutputStream.flush();
163               dataOutputStream.close();
164             } catch (IOException ex) {
```

```
165             Logger.getLogger(
166               ServerURL.class.getName()).log(
167                 Level.SEVERE, null, ex);
168             }
169           }else{
170             try{
171               System.out.println("Fila de Download vazia!!!");
172               //criar um datagrama socket
173               ds = new DatagramSocket();
174               //cria um buffer do tamanho da url
175               byte[] bufout =
176                 new byte[url.length()];
177               //prepara pacote para ser enviado em broadcast
178               InetAddress ip =
179                 InetAddress.getByName("255.255.255.255");
180               bufout = url.getBytes();
181               //prepara um datagrama
182               DatagramPacket packet =
183                 new DatagramPacket(
184                   bufout, bufout.length, ip, Config.portTransfMensagem);
185               //envia uma datagrama na rede
186               ds.send(packet);
187               if(ds==null){}
188               System.out.println("->A pesquisar sevidores na rede");
189               od.ods.textArea.append("\n|> Requisitando sevidores na rede");
190               //Timeout de 2s para a espera de resposta
191               ds.setSoTimeout(5000);
192               //cria um novo buffer
193               byte [] bufin =
194                 new byte[1024];
195               DatagramPacket dsin =
196                 new DatagramPacket(bufin, bufin.length);
197               ds.receive(dsin);
198               InetAddress inetAddress = dsin.getAddress();
199               System.out.println(
200                 "Servidor->"+inetAddress.getHostName()+
201                   "("+inetAddress.getHostAddress()+"): "
202                      +trataUrl.getNameFile(new String(
203                        dsin.getData()).substring(
204                          0,dsin.getLength()))+" encontrado");
205               od.ods.textArea.append(
206                 "\n|> "+inetAddress.getHostName()+
207                    "("+inetAddress.getHostAddress()+"): "+
```

```
208                     trataUrl.getNameFile(new String(
209                       dsin.getData()).substring(
210                         0,dsin.getLength()))+" encontrado");
211             ds.close();
212             DataOutputStream dataOutputStream;
213             dataOutputStream =
214               new DataOutputStream(cliente.getOutputStream());
215             dataOutputStream.write((
216               inetAddress.getHostName()+"("+inetAddress.getHostAddress()+
217                 ") fazendo o download de: "+trataUrl.getNameFile(
218                   url)).getBytes());
219             dataOutputStream.flush();
220             dataOutputStream.close();
221           }catch(SocketTimeoutException ste){
222             System.out.println("Ninguem fazendo download!!!");
223             od.FilaDownload.add(url);
224             od.ods.addDownload(url);
225             System.out.println("Fazendo download de "+
226               trataUrl.getNameFile(url));
227             od.ods.textArea.append("\n|> Fazendo download de "+
228               trataUrl.getNameFile(url));
229             try{
230               DataOutputStream dataOutputStream;
231               dataOutputStream =
232                 new DataOutputStream(cliente.getOutputStream());
233               dataOutputStream.write(("Fazendo o download de: "+
234                 trataUrl.getNameFile(url)).getBytes());
235               dataOutputStream.flush();
236               dataOutputStream.close();
237             }catch(IOException ex) {
238               Logger.getLogger(ServerURL.class.getName()).log(
239                 Level.SEVERE, null, ex);
240             }
241           }catch(IOException ioe){
242             ioe.printStackTrace();
243           }
244         }
245       }
246     }
247   }
248  }
249 }
```

APÊNDICE J – Class TabelaDownloads

```
1  package ourdown;
2
3  import java.util.ArrayList;
4  import java.util.Iterator;
5  import java.util.Observable;
6  import java.util.Observer;
7  import javax.swing.JProgressBar;
8  import javax.swing.table.AbstractTableModel;
9  //Esta classe gerencia os dados da tabela de downloads
10 public class TabelaDownloads extends
11  AbstractTableModel implements Observer{
12  //Estes são os nomes para as colunas da tabela
13  private static final String[] columnNames =
14    {"Arquivo","Tamanho","Progresso","Status"};
15  //Estas são as classes para cada valor da coluna
16  private static final Class[] columnClasses =
17    {String.class,String.class,JProgressBar.class,String.class};
18  //A lista de downloads da tabela
19  public ArrayList downloadLista = new ArrayList();
20  //adiciona um novo download a tabela
21  public void addDownload(Download download){
22    //registra para ser notificado quando alterar o download
23    download.addObserver(this);
24    downloadLista.add(download);
25    //grava a notificaçao em uma linha da tabela
26    fireTableRowsInserted(getRowCount()-1, getRowCount()-1);
27  }
28  //obtem um download para uma linha especifica
29  public Download getDownload(int row){
30    return (Download) downloadLista.get(row);
31  }
32  //remove o download da lista
33  public void clearDownload(int row){
34    downloadLista.remove(row);
35    fireTableRowsDeleted(row, row);
```

```
36   }
37   //obtem o tamanho da coluna
38   public int getColumnCount(){
39     return columnNames.length;
40   }
41   //retorna o nome da coluna
42   public String getColumnName(int col){
43     return columnNames[col];
44   }
45   //retorna uma classe da coluna
46   public Class getColumnClass(int col){
47    return columnClasses[col];
48   }
49   //retorna o tamanho das linhas
50   public int getRowCount(){
51     return downloadLista.size();
52   }
53   //retorna valor para uma linha especifica e
54   // a combinação da coluna
55   public Object getValueAt(int row, int col){
56     Download download = (Download) downloadLista.get(row);
57     switch(col){
58       case 0:
59         return download.trataUrl.getNameFile(download.getUrl());
60       case 1:
61         int size = download.getSize();
62         return (size==-1)?"":Integer.toString(size);
63       case 2:
64         return new Float (download.getProgress());
65       case 3:
66         return Download.STATUS[download.getStatus()];
67     }
68     return "";
69   }
70   //Update e chamado quando um download notifica
71   //o observers dele de alguma alteracao
72   public void update(Observable o, Object arg) {
73     int index = downloadLista.indexOf(o);
74     fireTableRowsUpdated(index, index);
75   }
76 }
```

APÊNDICE K – Class TratamentoDeUrl

```
1  package ourdown;
2  import java.net.URL;
3  import java.util.Iterator;
4
5  //Classe para tratamento de url's
6  public class TratamentoDeUrl {
7    public boolean ComparaUrl(OurDown od, String url){
8      boolean found=false;
9      String nomeArquivo = getNameFile(url);
10     Iterator <String> index = od.FilaDownload.iterator();
11     while(index.hasNext()&&found==false){
12       String url_ = index.next();
13       String nomeArquivo_ = getNameFile(url_);
14       if(url.equals(url_) || nomeArquivo.equals(nomeArquivo_)){
15         found = true;
16       }
17     }
18     return found;
19   }
20   public String getNameFile(String s){
21     URL url = null;
22     try {
23       url = new URL(s);
24     } catch (Exception e) {}
25     String fileName = url.getFile();
26     return fileName.substring(fileName.lastIndexOf("/")+1);
27   }
28 }
```