UNIVERSIDADE ESTADUAL PAULISTA
"Júlio de Mesquita Filho"
Pós-Graduação em Ciência da Computação

Luis Gustavo Crepaldi

# *Middleware* de Comunicação entre Objetos Distribuídos para Gerenciamento de Computadores baseado em Redes Sem Fio (WSE-OS)

UNESP
2011

Luis Gustavo Crepaldi

# *Middleware* de Comunicação entre Objetos Distribuídos para Gerenciamento de Computadores baseado em Redes Sem Fio (WSE-OS)

Orientador: Prof. Dr. Marcos Antônio Cavenaghi

Dissertação de Mestrado elaborada junto ao Programa de Pós-Graduação em Ciência da Computação – Área de Concentração em Sistemas de Computação, como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação

UNESP
2011

**LUIS GUSTAVO CREPALDI**

*Middleware* de Comunicação entre Objetos Distribuídos para Gerenciamento de Computadores baseado em Redes Sem Fio (WSE-OS)

Dissertação apresentada para obtenção do título de Mestre em Ciência da Computação, área de Sistemas de Computação junto ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Julio de Mesquita Filho", Campus de São José do Rio Preto.

BANCA EXAMINADORA

Prof. Dr. Marcos Antônio Cavenaghi
Professor Adjunto
UNESP-Bauru
Orientador

Prof. Dr. João Paulo Papa
Professor Assistente Doutor
UNESP-Bauru

Profª. Drª. Regina Helena Carlucci Santana
Professor Adjunto
USP/ICMC – São Carlos

São José do Rio Preto, 31 de Março de 2011

# AGRADECIMENTOS

Agradeço a Deus por ter permitido a realização de um sonho.

Meus sinceros agradecimentos a dois amigos com quem tive a grande honra de trabalhar: ao meu orientador Prof. Dr. Marcos Antônio Cavenaghi e a minha co-orientadora Profª. Drª. Roberta Spolon. Este trabalho não poderia ter sido realizado sem o generoso auxílio das suas interseções e pronta disposição para compartilhamento de seus conhecimentos.

Aos meus queridos pais, Oswaldo e Conceição, e avó Olinda. Mãos amigas presentes em todas épocas da minha vida. A vocês que me deram a vida e me ensinaram a vivê-la com dignidade, não bastaria um muito obrigado. Meus mentores, meu orgulho, meu sangue, toda minha razão.

Ao meu irmão e melhor amigo Paulo, companheiro fiel e querido. Impossível imaginar tudo isso sem você. Por opção e amor, meu ponto inquestionável de referência.

A minha querida namorada Bruna, responsável por um capítulo à parte nessa minha história.

Ao meu amigo Adriano Ricardo Digiere. Meus sinceros agradecimentos pelo oportuno e essencial auxílio no compartilhamento de pesquisas durante estes últimos anos.

À CAPES e Pró-Reitoria de Pós-Graduação da UNESP pelo apoio financeiro.

À UNESP e ao Departamento de Computação da Faculdade de Ciências de Bauru, que proporcionaram recursos para realização de experimentos deste projeto.

Ao Conselho do Programa de Pós-Graduação em Ciência da Computação da UNESP, em especial ao Coordenador Prof. Dr. Aparecido Nilceu Marana por sua atenção e disposição em sempre me atender.

Aos demais amigos e colegas agregados ao longo desses seis anos de UNESP-Bauru e a todos que ajudaram direta ou indiretamente na realização desta dissertação: Profª. Drª. Renata Spolon Lobato, Prof. Dr. João Paulo Papa, Profª. Drª. Andréa Carla Gonçalves Vianna, Elizabeth Bonsaglia Barboza, João Paulo Pascucci Campagna, entre tantos outros. A todos vocês, manifesto meu profundo apreço.

*"Viver no mundo sem tomar consciência do significado do mundo é como vagar por uma imensa biblioteca sem tocar os livros."*

Manly P. Hall (1901 – 1990)

# SUMÁRIO

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE ABREVIATURA E SIGLAS

3DES: *Triple Data Encryption Standard*
API: *Application Programming Interface*
AES: *Advanced Encryption Standard*
ASCII: *American Standard Code for Information Interchange*
BPAC: *Base de Ponto de Acesso Central*
CBC: *Cipher Block Chaining*
CDR: *Common Data Representation*
CORBA: *Common Object Request Broker Architecture*
DCOM: *Distributed Component Object Model*
DDR: *Double Data Rating*
DDR2: *Double Data Rating 2*
DDR3: *Double Data Rating 3*
DHCP: *Dynamic Host Configuration Protocol*
DL: *Digest Lenght*
FC13: *Fedora Core 13*
GB: *Gigabyte*
GNOME: *GNU Network Object Model Enviroment*
GPS: *Global Positioning System*
GUI: *Graphical User Interface*
HMAC: *Hash-based Message Authentication*
HTML: *Hyper Text Markup Language*
HTTP: *Hypertext Transfer Protocol*
IDEA: *International Data Encryption Algorithm*
IDL: *Interface Description Language*
IEEE: *Institute of Electrical and Electronics Engineers*
IETF: *Internet Engineering Task Force*
IP: *Internet Protocol*
ISP: *Internet Service Provider*
ISU: *Imagem de Sistema Única*
JPEG: *Joint Photographic Experts Group*
KL: *Key Length*
LAN: *Local Area Network*
LTSP: *Linux Terminal Server Project*
MAC: *Message Authentication Code*
MD5: *Message-Digest Algorithm 5*
MB: *Megabyte*
MPEG: *Moving Picture Experts Group*
MV: *Máquina Virtual*

OMG: *Object Management Group*
PDA: *Personal Digital Assistant*
PNG: *Portable Network Graphics*
PSK: *Pre-Shared Key*
PXE: *Preboot eXecution Environment*
RAM: *Random-Access Memory*
RDP: *Remote Desktop Protocol*
RFC: *Request for Comments*
RMI: *Remote Method Invocation*
RPC: *Remote Procedure Call*
SCP: *Secure Channel Protocol*
SFTP: *SSH File Transfer Protocol*
SHA1: *Secure Hash Algorithm 1*
SO: *Sistema Operacional*
SOC: *Sistema Operacional Convidado*
SOH: *Sistema Operacional Hospedeiro*
SSH: *Secure Shell*
SSHFS: *Secure Shell File System*
SSI: *Single System Image*
SSID: *Service Set IDentifier*
STD: *Sistema de Transmissão Diferencial*
TCP: *Transmission Control Protocol*
TCSC: *Thin Client Server Computing*
UCP: *Unidade Central de Processamento*
UDP: *User Datagram Protocol*
USB: *Universal Serial Bus*
VDI: *Virtual Desktop Infrastructure*
VNC: *Virtual Netowork Computing*
WAV: *Waveform Audio File Format*
WLAN: *Wireless Local Area Network*
WPA2: *Wi-Fi Protected Access 2*
WSE-OS: *Wireless Sharing Enviroment – Operating Systems*
ZLIB: *Z Library*

# RESUMO

Para simplificar o gerenciamento de computadores, vários sistemas de administração estruturados por conexões físicas adotam técnicas avançadas para gestão de configuração de *software*. No entanto, a forte ligação entre *hardware* e o *software* faz com que haja uma individualização desta gerência, além da penalização da mobilidade e ubiqüidade do poder computacional. Neste cenário, cada computador torna-se uma entidade individual a ser gerenciada, exigindo operações manuais de configuração da imagem de sistema. Tecnologias que oferecem gestão centralizada baseadas em conexões físicas cliente-servidor, combinando técnicas de virtualização com a utilização de sistemas de arquivos distribuídos, refletem a degradação em flexibilidade e facilidade de instalação deste sistema gerenciador. Outras arquiteturas para gerenciamento centralizado que estruturam o compartilhamento de dados através de conexões físicas e dependem do protocolo PXE, apresentam os mesmos impasses descritos anteriormente. Diante das limitações dos modelos de gerenciamento centralizado baseado em conexões físicas, o objetivo deste trabalho é o desenvolvimento de um *middleware* de comunicação cliente-servidor como parte integrante e necessária para um ambiente de gerenciamento centralizado em redes de comunicações sem fio. Este ambiente, denominado WSE-OS (*Wireless Sharing Enviroment – Operating Systems*), é um modelo baseado *Virtual Desktop Infrastructure* (VDI) que associa técnicas de virtualização e sistema de acesso remoto seguro para criação de uma arquitetura distribuída como base de um sistema de gestão. WSE-OS é capaz de realizar a replicação de sistemas operacionais em um ambiente de comunicação sem fio além de oferecer abstração de *hardware* aos clientes. O WSE-OS pode substituir o *boot* local com disco rígido por um *boot* de uma Imagem de Sistema Única virtualizada em servidor através do *middleware* de comunicação, aumentando a flexibilidade e permitindo que diversos sistemas operacionais sejam inicializados sem a necessidade de um disco rígido nos clientes, refletindo em redução de custo de *hardware* e complexidade de manutenção de *software* mesmo em um ambiente com configuração heterogênea de *hardware*. WSE-OS é uma solução de gerenciamento que agrega os benefícios descritos e surge como alternativa inédita aos modelos existentes de gerenciamento centralizado no mercado nos quais tem como base de interligação entre máquinas o enlace com fio. Os experimentos realizados com o *middleware* de comunicação ilustram que o tempo de *boot* de um computador associado com a instanciação de uma imagem de referência sobre a arquitetura WSE-OS e com otimização em memória *Flash* tem degradação menor do que 12.2% em relação a um computador com *boot* através do disco rígido local, e ganho médio de 52.58% para instanciação de aplicativos com otimização em memória RAM se comparados com tempo de inicialização em *hardware* nativo. Através da utilização do Sistema de Transmissão Diferencial de Dados e *cache* em memória RAM é possível afirmar que operações *request-reply* da arquitetura proposta ilustra desempenho 17.42% superior mesmo em ambientes com várias conexões concorrentes. Por outro lado, o uso de *cache* em memória Flash garante também que, após o primeiro boot do sistema WSE-OS, o processo de inicialização do sistema e carregamento de um sistema operacional tenha um rendimento 7.16% superior. Os testes desenvolvidos mostram que os recursos de otimizações do *middleware* auxiliam na desoneração de processamento cliente, servidor e da rede de comunicação sem fio.

**Palavras-Chaves:** *Virtual Desktop Infrastructure*; Gerenciamento de Computadores; Acesso Remoto, *Middleware* para Comunicação Sem Fio

# ABSTRACT

To simplify computer management, various administration systems structured with physical connections adopt advanced techniques to manage software configuration. Nevertheless, the strong link between hardware and software makes for an individualism of that management, besides penalizing computational mobility and ubiquity. In this scenario, each computer becomes an individual entity to be managed, requiring manual operations of the system image configuration. Technologies that offer centralized management based on client-server physical connections, combining virtualization techniques with the use of distributed file systems in clusters with distributed processing on network computers reflect the deterioration in flexibility and ease of installation and maintenance of distributed applications. Other architectures for centralized management that structure the sharing of data through physical connections and depend on the PXE protocol, present the same dilemmas described above. Given the limitations models of centralized management based on physical connections, the objective of this project is the development of a middleware for client-server communication as part necessary of an environment for centralized management in wireless communications networks. This environment, called WSE-OS (Wireless Sharing Environment – Operating Systems), is a model based Virtual Desktop Infrastructure (VDI), which combines virtualization techniques and secure access system for creating a distributed architecture as the basis for a management system. WSE-OS is capable of replicating operating systems in a wireless environment, addition to providing hardware abstraction to clients. The WSE-OS can replace the boot with local hard disk to a boot from SSI (Single System Image) virtualized in server via communication middleware, increasing flexibility and allowing multiple operating systems to be initialized without the need for a hard disk in clients, resulting in reduced hardware cost and complexity of software maintenance even in an environment with heterogeneous hardware configuration. WSE-OS is a management solution that combines the benefits described and appears as a unpublished alternative to existing models of centralized management in the market which is based on the interconnection between that machines wired link. The experiments with the communication middleware show that the boot time of a computer associated with the instantiation of a reference image on the WSE-OS architecture and optimization in Flash memory has less degradation of that 12.2% compared to a computer boot from local hard disk, and 52.58% average gain for instantiation applications in RAM memory compared to the boot time on native hardware. Using Differential Transmission Protocol and RAM memory cache is possible to affirm that request-reply operations of the proposed architecture show superior performance 17.42% even in environments with multiple concurrent connections. Moreover, the use of Flash memory cache ensures that, after the WSE-OS system first boot, the process system boot and operating system loading has a higher yield 7.16%. The tests developed show that the capabilities of the middleware optimizations assist in the relief processing client, server and wireless communication network.

**Keywords** – Virtual Desktop Infrastructure; Computer Management; Virtual Machines; Remote Access; Middleware for Wireless Communication

# 1. INTRODUÇÃO

## 1.1 Contextualização

A agregação de computadores em uma organização pode ser entendida como um reflexo direto no aumento de geração de valor e produtividade da empresa. Com a necessidade da interligação destas máquinas, junto com o crescimento heterogêneo dos tipos e aplicações de redes, a criação de ferramentas de gerenciamento tornou-se indispensáveis para a redução do Custo Total de Propriedade (MASUDA, 2009). A computação baseada em servidor aliado as soluções *thin-clients* são alternativas para que a complexidade de administração individual de computadores seja transladada para um gerenciamento centralizado de redes fundamentadas na arquitetura cliente-servidor (LAI & NIEH, 2006).

O paradigma Cliente-Servidor é amplamente empregado no gerenciamento centralizado por buscar a flexibilidade, interoperabilidade e escalabilidade dos componentes de redes envolvidos (SADOSKI, 2002). O planejamento de gerência baseado nas evoluções das aplicações e necessidades de produção de uma empresa promove um nível dinâmico de maturidade computacional (MORGADO, CRUZ & TWANI, 2008), provendo estabilidade e padronização para os ambientes de execução.

Entretanto, a natureza heterogênea do parque computacional torna a atividade de gerenciamento limitada à individualização de grupos compostos por *hardware*/*software* similares*,* refletindo em complexidade de execução em escala. Neste contexto, a virtualização pode ser utilizada como ferramenta de abstração de *hardware*, otimizando configurações de rede com a concessão de um único conjunto padronizado de periféricos*.*

Como exemplificação, o projeto FLEXlab (CRUZ, 2008) combina técnicas de virtualização com a utilização de sistemas de arquivos distribuídos em *Clusters* de computadores de modo a permitir a criação de uma solução de gerenciamento centralizado mas com processamento distribuídos nos computadores da rede, ao contrário do que sugere soluções *thin client*.

Outras soluções de gerenciamento utilizam ferramentas de virtualização para a criação de uma infra-estrutura virtual de *desktops* (VMWARE, 2007). Tais estruturações são conhecidas como VDI (*Virtual Desktop Infraestructure*) e têm similaridades com a arquitetura *thin client.* O modelo centraliza a gestão de um ambiente completo cliente executados em máquinas virtuais dentro de *data center* de modo a interagir com clientes com recursos mínimos de *hardware*. Algumas abordagens para o gerenciamento empregam a técnicas de virtualização e mecanismos de *boot* remoto para propor a virtualização apenas dos *hard disks* dos terminais clientes. O Ardence *Desktop* (ARDENCE, 2007) e o Citrix Provisioning Server for *Desktops* (CITRIX, 2008) são soluções que utilizam a montagem pela rede, em tempo de *boot,* de um arquivo binário que representa um disco rígido.

Porém, estas soluções atuais de mercado para gestão centralizada penalizam a mobilidade e a ubiqüidade do poder computacional. Pelo fato destes ambientes de compartilhamento serem estruturados por conexões físicas e a grande maioria dependente do protocolo PXE (*Preboot eXecution Environment*) (INTEL, 2010) para a comunicação remota, existe grande limitação da flexibilidade e facilidade de instalação dos terminais. Como a tecnologia de redes sem fio vem ganhando espaço no mercado e tornando padrão para a transmissão de dados, as estruturas de ambiente de trabalho devem acompanhar esta mudança para tornar o gerenciamento distribuído mais flexível e independente de conexões físicas.

Diante deste cenário, com a combinação das técnicas de virtualização e utilização de sistemas de acesso/ambiente remoto, torna-se possível a criação de uma solução de gerenciamento centralizada sem fio, ao contrário das soluções convencionais cabeadas. Esta abordagem representa uma nova perspectiva em soluções para gerenciamento de ambientes de compartilhamento de sistemas operacionais e propõem uma nova utilização para a tecnologia de acesso remoto.

## 1.2 Motivação

Os sistemas distribuídos estão cada vez mais estruturando seus enlaces de comunicação através de redes sem-fio. A interligação de computadores pessoais, telefones móveis, máquinas industriais e até eletrodomésticos utilizam protocolos baseados na radiofreqüência por permitirem a mobilidade e, conseqüentemente, a ubiqüidade do poder computacional (HENDERSON, 2008). Desta maneira, a

cooperação entre dispositivos e a infraestrutura para a troca de dados é realizada sem a necessidade de interligação física entre tais dispositivos.

Com a necessidade de criação de ambientes de gerenciamento que atenda flexibilidade de conexão e mobilidade espacial de clientes, as redes sem-fio encaixam nestas soluções de forma coesa. O emprego de modelagem distribuída orientada a eventos (COULOURIS, 2007) minimiza um potencial gargalo para transmissões de dados em enlaces sem conexões físicas, adequando as redes sem fio ao trabalho de interface entre cliente e servidor para gestão dos negócios.

A diminuição do custo de periféricos de *hardware*, associado ao surgimento de processadores de múltiplos núcleos, e ao interesse de utilização de máquinas virtuais para a padronização de terminais, contribuem para que estruturas virtuais de *desktops* sejam construídas. Além do melhor aproveitamento do poder computacional ocioso em servidores, estas estruturas reduzem significativamente o custo de implantação de uma nova aplicação (HALETKY, 2009).

Além disto, a modelagem de objetos distribuídos outorga benefícios de manipulação remota e instanciação de objetos para que sistemas de acesso/ambiente remoto sejam implementados para transmissão de dados estruturados sem que o enlace de comunicação seja comprometido. A simplificação de toda a complexidade de relacionamento entre máquinas heterogêneas é refletida em ganho de eficiência computacional, tanto para clientes quanto para servidores.

Diante de características peculiares de permissão de instanciação concorrente de diversos sistemas operacionais sobre mesmo conjunto de *hardware,* as máquinas virtuais associada aos sistemas de acesso/ambiente remoto e as redes sem fio, contribuem como componentes de uma proposta de solução para gerenciamento de computadores. Desta forma, uma única configuração de *software* é capaz de atender os computadores em uma rede sem fio, facilitando a disseminação de informação e gerenciamento dos computadores.

## 1.3 Objetivos Gerais

O objetivo desta monografia é apresentar o projeto de um Ambiente de Compartilhamento de Dados baseados em uma Rede de Comunicação Sem Fio chamado WSE-OS (*Wireless Sharing Enviroment – Operating Systems*): um modelo baseado em infra-estruturas virtuais de *desktops* (VDI) que associa técnicas de virtualização e sistema de acesso remoto seguro para criação de uma arquitetura distribuída como base de um sistema de gestão.

O modelo de arquitetura distribuída proposto possui duas entidades independentemente estruturadas dispostas remotamente entre clientes e servidor WSE-OS que, quando conjugadas, são capazes de configurar um sistema de compartilhamento para instanciação e execução de sistemas operacionais em uma rede de comunicação sem fio. Desta forma, o WSE-OS também serviu de base para outro projeto de pesquisa concluído que estudou e avaliou a aplicação de uma Camada de Gerenciamento no servidor WSE-OS como módulo integrante do projeto em discussão.

Diante desta estruturação bilateral, esta dissertação tem por objetivos específicos:

- Desenvolvimento de um *middleware* para execução em terminais clientes funcionando como uma camada de inicialização, de operações remotas e de gerenciamento de recursos entre o *hardware* real e a Imagem de Sistema Única virtualizada sobre servidor WSE-OS;
- Desenvolvimento de capacidade de execução do *middleware* em diversas plataformas de *hardware* baseada na arquitetura x86 e com diversas configurações de periféricos, principalmente aqueles associados a placas de conexão em redes sem fio;
- Desenvolvimento de um mecanismo de *boot* para o *middleware* através de portas USB (*Universal Serial Bus)* por meio do armazenamento deste *middleware* em *Flash Driver USB* (Pen-Drive), indicando independência de qualquer protocolo de *boot* que exija conexões físicas com o servidor;
- Incorporação ao *middleware* de um sistema de acesso seguro remoto fornecendo integridade e confidencialidade para transmissão de dados associado com tecnologias concebidas para o aceleramento de transações de conexões remotas baseadas em *X Window System 11* (COOPERSMITH, 2010) (BOOK, 2006);

- Desenvolvimento de uma interface gráfica com o usuário do sistema para realização de *login/logout,* configurações de rede, configurações de memória *cache* e configuração de compactação de imagem;
- Realização de experimentos para analisar o desempenho da arquitetura do *middleware* em diferentes *hardware* clientes, incluindo medição de desempenho para o uso ou não de memória *cache*;

## 1.4 Estrutura da Monografia

Este trabalho está organizado em sete capítulos e os assuntos abordados nesta monografia estão divididos da seguinte forma:

**Capítulo 1:** Contém a contextualização do escopo de pesquisa, ilustrando as principais motivações para o desenvolvimento do trabalho proposto e os objetivos pretendidos.

**Capítulo 2:** Descreve o conceito de sistemas distribuídos, *middleware* e o detalhamento da arquitetura cliente-servidor empregada neste trabalho. Além disto, o capítulo ilustra os princípios de comunicação entre processos distribuídos para que, posteriormente, os conceitos de comunicação entre objetos distribuídos sejam compreendidos.

**Capítulo 3:** Pormenoriza a modelagem distribuída orientada a objetos, especificando a arquitetura RMI (*Remote Method Invocation*) baseada em requisição e resposta com extensões para eventos/notificações. Estes conceitos guiam a implementação de módulos responsáveis pela comunicação entre clientes e servidor no ambiente proposto.

**Capítulo 4:** Apresenta a proposta WSE-OS associada com características técnicas e fluxo de execução do modelo arquitetural, além dos componentes fundamentais e objetivos específicos no qual esta dissertação está focada.

**Capítulo 5:** Descreve os módulos que compõem a arquitetura do *middleware* WSE-OS bem como características técnicas e funcionamento para que possa ser aplicado como parte integrante do projeto em questão.

**Capítulo 6:** Apresenta os resultados dos experimentos e realiza uma análise dos resultados obtidos.

**Capítulo 7:** Conclui a dissertação apresentando as considerações do trabalho e indicando futuras contribuições e continuidade do projeto com base nos estudos desenvolvidos a partir do WSE-OS.

# 2. SISTEMAS DISTRIBUÍDOS

## 2.1 Considerações Iniciais

As redes de comunicações de dados estão presentes em todos os lugares. Estes conjuntos de computadores independentes e interligados compartilham características básicas que tornam assunto relevante para pesquisa, formalizando a área de sistemas distribuídos. Descrito como um sistema único e consistente (TANENBAUM, 2007), os sistemas distribuídos possibilitam o compartilhamento de recursos e são estruturados por componentes na maioria heterogêneos que coordenam suas ações apenas pela comunicação distribuída de troca de mensagens (COULOURIS, 2007) refletindo na otimização de tarefas em relação ao custo/desempenho. Este capítulo apresenta os conceitos de sistemas distribuídos abordando modelos de arquitetura, comunicação e invocação entre processos remotos.

## 2.2 Caracterização de Sistemas Distribuídos

Um sistema distribuído pode ser definido como sendo aquele onde componentes de *software* e/ou *hardware*, localizados em computadores interligados em rede, se comunicam e coordenam suas ações apenas enviando mensagens entre si (COULOURIS, 2007). A construção de sistemas distribuídos tem por objetivo o compartilhamento de componentes de *hardware* e entidades de *software*, como arquivos e objetos de dados. Diante desta definição, existem três conseqüências relevantes que devem ser levadas em consideração (TANENBAUM e VAN STEEN, 2007) (COULOURIS, 2007):

Concorrência: a concorrência existente em redes de computadores é proveniente de sistemas que possuem entidades que, ao mesmo tempo, operam sobre um grupo de recurso compartilhado. Desta forma, a coordenação das diversas operações deve ser levada em consideração, ou a interferência pode causar inconsistência destes recursos.

*Inexistência de relógio global*: como os processos são executados concorrentemente nos sistemas distribuídos, a necessidade de cooperação entre eles é realizada pela troca de mensagens. Diferentemente dos sistemas isolados, a base global única de tempo para a coordenação dos processos não existe nestes programas concorrentes de modo que a única comunicação se dá por meio de envio de mensagens.

*Falhas independentes:* A falha de um dos componentes do sistema distribuído não implica na falha do sistema como um todo. Falhas de rede resultam apenas no isolamento de computadores que estão conectadas a ela, mas não necessariamente que eles parem de operar. Alguns componentes podem sofrer colapso sem que afetem outros, deixando-os ainda em funcionamento.

### 2.2.1 Exemplos de Sistemas Distribuídos

Como descrito na seção 2.2, o conceito da distribuição de sistemas envolve máquinas autônomas sem memória física compartilhada, mas que estão interligadas para um objetivo comum. Essa tecnologia de distribuição de serviços data aos anos de 70 e 80 com pesquisas de banco de dados distribuídos (NAKANISHI, 1981). Com a modernização e evolução das UCPs (Unidade Central de Processamento), o interesse em distribuir funcionalidades entre várias máquinas conectadas e com canal de comunicação ganhou forma, descartando o uso intensivo das aplicações apenas em *mainframes.*

A conectividade entre máquinas homogêneas fortemente acopladas e geograficamente próximas foi se generalizando em outra realidade à medida que a distribuição de serviços entre computadores heterogêneos e remotos se tornava realidade. A tecnologia Web/Internet nascida no ano 90 causou uma revolução na globalização do acesso a informações, sendo considerada desde então uma grande plataforma de conectividade (DICK, 1997).

A Internet é um grande sistema distribuído onde programas que estão em execução nos terminais conectados a ela interagem através de diversos tipos de canais de comunicação pela troca de mensagens permitindo o uso de serviços como a *World Wide Web*, transferência de arquivos e correspondências eletrônicas. A Figura 1 ilustra uma parte típica da Internet.

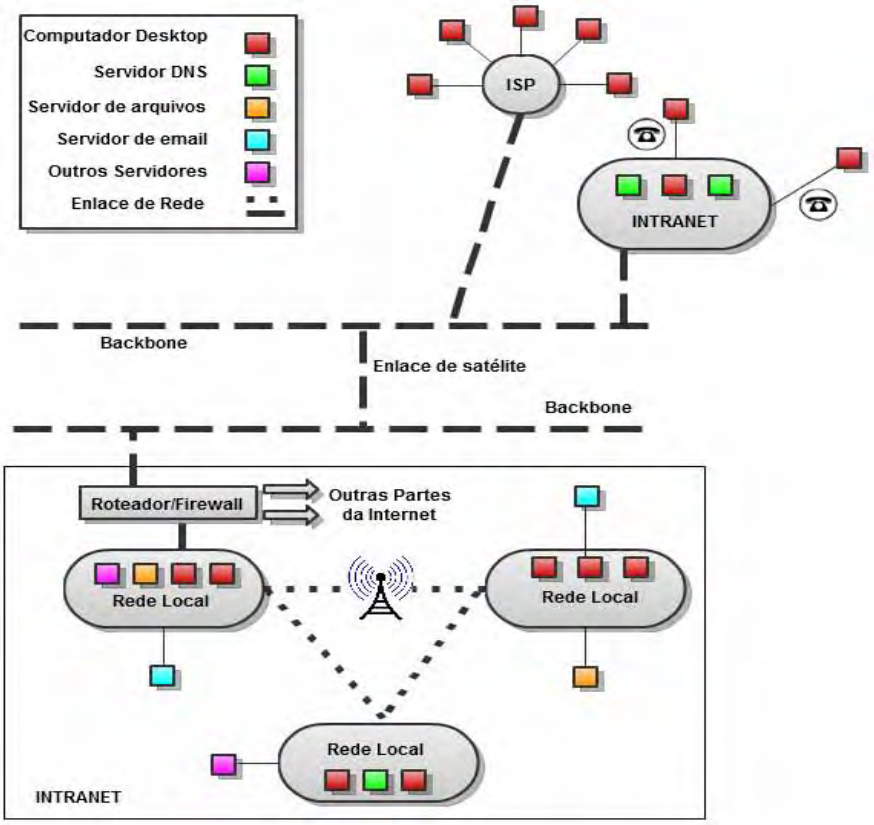


**Figura 1 - Exemplo de Sistema Distribuído. Adaptado de (COULOURIS, 2007)**

A Internet possui um conjunto de serviço aberto, podendo ser ampliada pela adição de servidores e outros tipos de serviços. Conforte ilustrado na Figura 1, os ISP (*Internet Service Providers*) são empresas que fornecem para organizações ou usuários domésticos acesso a Internet para usufruírem de todos os serviços disponíveis na rede.

A Intranet, como mostra a Figura 1, é uma parte da Internet gerenciada separadamente, cujo limite pode ser configurado para impor planos de segurança locais (COULOURIS, 2007). Ela pode ser composta por várias redes locais (LANs – *Local Area Networks*) interligadas por conexões de *backbone*. Um *backbone* é enlace de rede com elevado desempenho para a transmissão de dados (ROSE, 2007), podendo ser empregadas conexões por cabos de fibras ópticas, conexão via satélite ou outros meios que possuam grande largura de banda.

A Intranet pode utilizar serviços da Internet através de um roteador. Este também pode permitir que usuários de outras Intranets acessem serviços e documentos dos terminais. No entanto, algumas organizações preferem não ter acesso a Internet, constituindo redes isoladas. Outro modelo de sistemas distribuídos é exemplificado pelos avanços tecnológicos na miniaturização e interligação em redes sem fio, onde equipamentos portáteis e pessoais estão baseados no conceito de computação móvel e ubíqua (HAN, 2008) (ROY, 2009).

A portabilidade de computadores *notebook*, telefones móveis, câmera de vídeos, entre outros, com a capacidade de conexão com redes de comunicação em diferentes lugares, tornam a *computação móvel* possível (KLEINROCK, 1997). Na computação móvel, os usuários que estão distantes das Intranets de origem podem acessar informações por intermédios dos dispositivos que carregam.

A computação ubíqua (HAN, 2008) (WEISER, 1993) sugere a utilização de equipamentos pequenos conectados entre si onde o comportamento computacional é realizado de forma transparente. No entanto, a computação ubíqua e móvel se contextualizam no mesmo ideal, mas de modo específicos são distintas. A computação ubíqua é onipresente podendo ajudar as pessoas enquanto elas não estão no seu ambiente regular de trabalho, por outro lado, a computação móvel oferece a vantagem da eliminação de conexões físicas entre dispositivos. A Figura 2 mostra diferentes tipos de conexões que um usuário munido de dois dispositivos distintos pode realizar dentro de um ambiente de visita.

A Figura 2 sugere três formas diferentes de conexão sem fio que o usuário pode realizar neste ambiente anfitrião. O computador pessoal pode conectar com a Intranet através de uma rede local sem fio que possui cobertura de centenas de metros. Esta Intranet fornece todos os serviços de Internet já que se mantém interligada com a mesma.

Através de um dispositivo celular, o usuário pode realizar uma solicitação para o *software* embarcado traçar a rotas de itinerários utilizando uma rede de telecomunicação interligada a Internet. Ao mesmo tempo, o usuário pode executar o pedido de impressão a uma impressora remota localizada a poucos metros. Enfim, a Figura 2 ilustra os diversos sistemas distribuídos baseados no conceito de computação móvel e pervasiva.

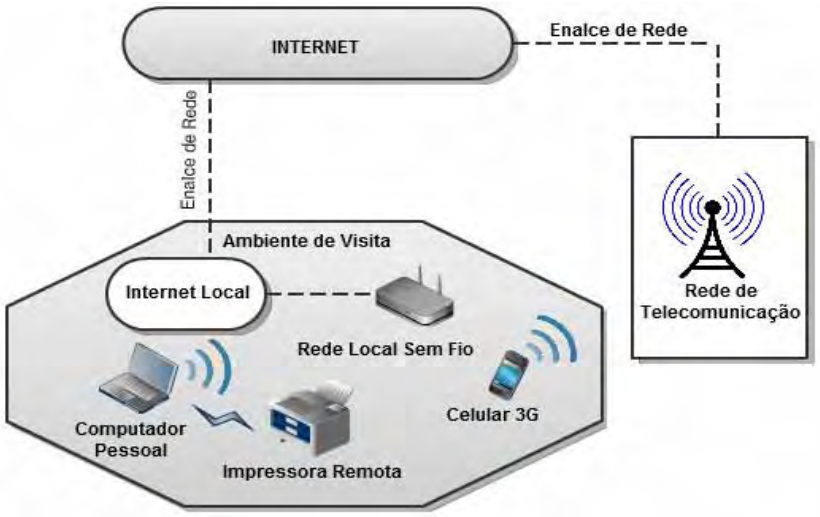


**Figura 2 - Equipamentos Móveis em um Sistema Distribuído. Adaptado de (COULOURIS, 2007)**

## 2.3 Modelos de Arquitetura

A arquitetura de um sistema consiste na sua estrutura de componentes (*hardware* e *software*) especificando suas propriedades e relacionamentos. Existem diversos tipos de arquiteturas para sistemas distribuídos, mas todas elas têm o objetivo de tornar o sistema confiável, gerenciável, adaptável e rentável (COULOURIS, 2007).

O modelo computacional centralizado (legado) baseado em *mainframes* dominou o mundo da computação da década de 60 até meados da década de 80. Uma arquitetura que centraliza todas as operações em um computador central, na qual informações são captadas através de um terminal. Extremamente limitada, a arquitetura não suporta interfaces gráficas com o usuário (*Graphic User Interface* – GUI) e acesso a múltiplos repositórios de dados geograficamente dispersos (SADOSKI, 2001).

Com o surgimento dos computadores pessoais, nasceu um novo paradigma de comunicação através de arquivos compartilhados e os primeiros passos para a arquitetura cliente-servidor. A arquitetura de compartilhamento de arquivos é conceituada na entrega do arquivo em uma localização compartilhada para o ambiente da estação de trabalho. Esta arquitetura apresenta bom desempenho apenas se o

número de compartilhamentos de arquivos e o volume de dados transferidos forem pequenos (SADOSKI, 2001).

Para simplificar a contextualização deste projeto e apresentar a arquitetura que soluciona estas limitações, podemos classificar os componentes de arquitetura como sendo processos *servidores* e processos *clientes*. A abstração destes componentes facilita o entendimento das definições de padrões para a distribuição de dados, além dos papéis funcionais e comunicação entre eles. A seção 2.3.1 descreverá o modelo amplamente usado para sistemas distribuídos e que também servirá de base teórica para o projeto desta monografia.

### 2.3.1 Arquitetura Cliente-Servidor

A divisão de tarefas entre os processos e sua atribuição nos diversos computadores de uma rede de comunicação tem implicações diretas na segurança, desempenho e confiabilidade (TANENBAUM e VAN STEEN, 2007). Um modelo que serve de base para esta divisão de tarefas é a baseada em *processos clientes e servidores*.

A arquitetura cliente-servidor é amplamente empregada por diminuir o tráfego na rede fornecendo respostas às solicitações ao invés de transferir todo o arquivo como ocorre em arquiteturas de compartilhamento de arquivos. Neste contexto, um cliente pode ser definido como um invocador de serviços, e o servidor como um provedor destes serviços.

Como ilustrado na Figura 3, é evidente a estruturação simplista na qual os processos clientes interagem com os processos servidores para acesso de dados compartilhados, ocasionando a divisão do processamento das aplicações. A invocação por parte do processo cliente pode ser distribuída a diversos servidores e, por outro lado, servidores podem ser clientes de outros servidores. Desta maneira, dependendo da configuração destes processos, um computador na rede pode funcionar como sendo cliente e servidor ao ter que fazer uma solicitação adicional à outra máquina (COFFMAN, 2010).

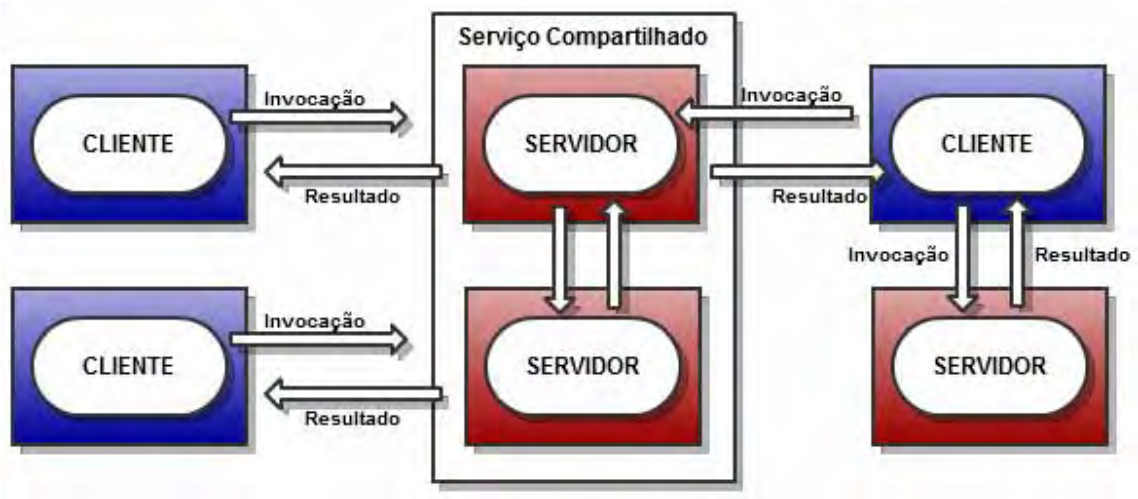


**Figura 3 - Arquitetura Cliente-Servidor. Adaptado de (COULOURIS, 2007)**

A arquitetura cliente-servidor teve sua aceitação no final dos anos 80 com o surgimento da Internet. Baseada em uma estrutura modular, versátil e na comunicação por troca de mensagens, este modelo busca melhorar a flexibilidade, interoperabilidade e escalabilidade dos componentes dos sistemas distribuído (SADOSKI, 2001).

Algumas variações neste modelo podem ser extraídas para que as características citadas sejam válidas. O uso de vários servidores colaborativos e servidores *caches* aumentam o desempenho e resiliência, já o conceito de interoperabilidade espontânea facilita associações de criação e destruição entre dispositivos rotineiramente (COULOURIS, 2007).

A arquitetura cliente-servidor pode ser classificada de acordo com a distribuição de três elementos: interface de aplicação cliente, camada lógica de negócios e armazenamento destes dados. A interface de aplicação é responsável pela apresentação de informações para o usuário final. É nesta interface que ocorre a interpretação dos dados que a camada lógica de negócios oferece ao terminal. Já a camada lógica de negócios tem o objetivo de realizar a comunicação com a base de dados. Esta camada, para atender necessidades de escalabilidade e outros fatores, evolui para uma camada independente apresentada pela seção 2.3.3 como camada intermediadora de aplicação. O último elemento, a camada de dados, tem o objetivo de armazenamento de dados no servidor. Nesta camada ocorrem consultas e respostas de banco de dados para a aplicação cliente.

### 2.3.2 Arquitetura Cliente-Servidor - 2 Camadas

A arquitetura cliente-servidor de duas camadas foi apresentada nos anos 80. Substituindo servidores de arquivos compartilhados por servidores de banco de dados, a distribuição de processos poderia ser feito em duas camadas distintas: cliente e servidor.

Neste modelo, o cliente realiza uma invocação ao servidor. O servidor processa este pedido e retorna apenas os dados resultantes daquela solicitação. Ao contrário do que aconteceria em um servidor compartilhado de arquivos, o modelo cliente servidor reflete um ambiente para múltiplos usuários. Na Figura 4 ilustra-se a arquitetura em 2 camadas.

A distribuição dos três elementos fundamentais desta arquitetura caracteriza subdivisões importantes. Um *Fat Client* é uma atribuição feita para clientes que possuem a interface de aplicação cliente e processamento dos dados. Neste caso, o servidor apenas armazena os dados e faz validações básicas que garantem sua integridade. Por outro lado, clientes que possuem apenas a interface de aplicação cliente baseada em janelas para que este possa executar programas aplicativos em computador remoto, são conhecidos como *Thin Clients*. Nesta situação, o servidor é responsável tanto pelo armazenamento de dados quanto ao processamento.

Apesar deste modelo apresentar vantagens de modularidade do ambiente computacional e menor custo de *software*/*hardware* do que nos ambientes legados que não permitem a distribuição de processos e o suporte multiusuário, o modelo de duas camadas mantém a interface de aplicação cliente juntamente com a camada lógica de negócios. Com a centralização da camada lógica de negócios no cliente, há uma baixa aceitação de aplicações heterogêneas pela limitação do protocolo de transporte/comunicação especificada nestes terminais. Por conseqüência, há um baixo encapsulamento de dados, já que o cliente manipula informações diretamente do banco de dados. A arquitetura em duas camadas não acompanha a evolução de outros modelos da computação como a orientação a objetos e a Internet, de modo a levar à evolução natural para arquiteturas mais flexíveis baseadas em três e N camadas.

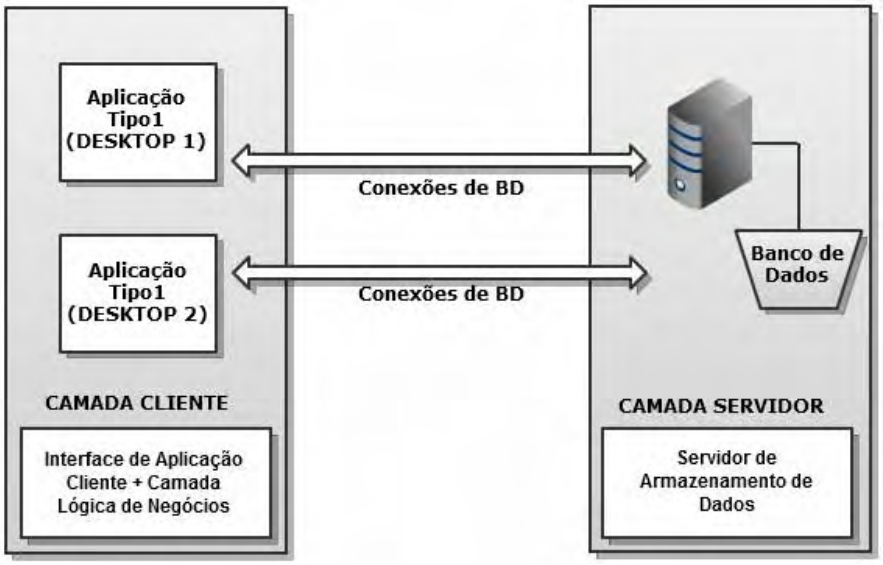


**Figura 4 - Arquitetura Cliente-Servidor de 2 Camadas. Adaptado de (SADOSKI, 2002)**

### 2.3.3 Arquitetura Cliente-Servidor - 3 Camadas / Web-Based

As limitações impostas pela arquitetura de duas camadas foram supridas com o conceito de cliente-servidor em três camadas. Para desviar dos problemas de escalabilidade intrínsecas a conexões de bancos de dados, dificuldades de acesso de fontes heterogêneas e, principalmente, o problema de manutenção, criou-se uma camada média.

Introduzida entre a interface de aplicação cliente (camada de apresentação) e o servidor de armazenamento de dados (camada de dados), esta nova camada (camada de aplicação) provê serviços de gerenciamento de processos compartilhados por múltiplas aplicações diferentes (SADOSKI, 2002). A Figura 5 ilustra uma arquitetura em 3 camadas.

Com a camada intermediadora de aplicação, problemas de manutenções foram reduzidos. Mudanças necessárias na camada lógica de negócios alteravam toda a configuração de software no cliente como visto nas arquiteturas de duas camadas. Conceitualmente diferente, a camada de aplicação, que substitui a camada lógica de negócios vinculada aos clientes, fornece a vantagem de ser independente dos terminais.

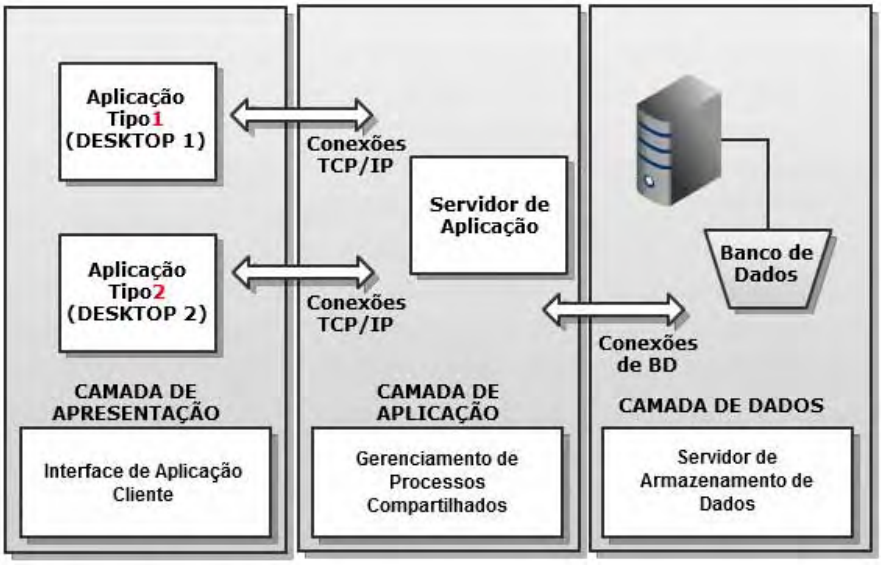


**Figura 5 - Arquitetura Cliente-Servidor de 3 Camadas. Adaptado de (SADOSKI, 2002)**

Através dos servidores de aplicação é possível que diversos aplicativos heterogêneos possam ser executados no mesmo computador cliente utilizando diferentes protocolos de transporte. O transporte de mensagens baseada em datagrama UDP (*User Datagram* Protocol) e fluxo TCP (*Transmission Control Protocol*) associado a identificação de máquinas pelo protocolo IP (*Internet Protocol*), substitui as chamadas e conexões diretas de banco de dados como ocorre no modelo de 2 camadas.

No entanto, esta base original das arquiteturas em 3 camadas tem problemas quanto ao controle administrativo dos desktops clientes. A instalação individual dos programas nos terminais clientes para a comunicação com outras camadas é uma tarefa onerosa. Neste caso, o problema de manutenção persiste quando há mudanças na camada de apresentação.

A solução para a falta de gerenciamento da camada de apresentação é a utilização de programas baseados em clientes universais. A utilização do *Browser* juntamente com o conceito da Intranet, facilitou o gerenciamento dos terminais clientes, onde a instalação prévia de programas foi desconsiderada. A este modelo foi dado o nome de arquitetura *Web-Based* em três camadas. O modelo Web-Based emprega a padronização no transporte com TCP e UDP oferecido pelo primeiro modelo de 3 camadas e adiciona uma nova padronização na comunicação dos aplicativos através do protocolo HTTP (*Hypertext Transfer Protocol* )/HTML (*Hyper Text Markup Language*).

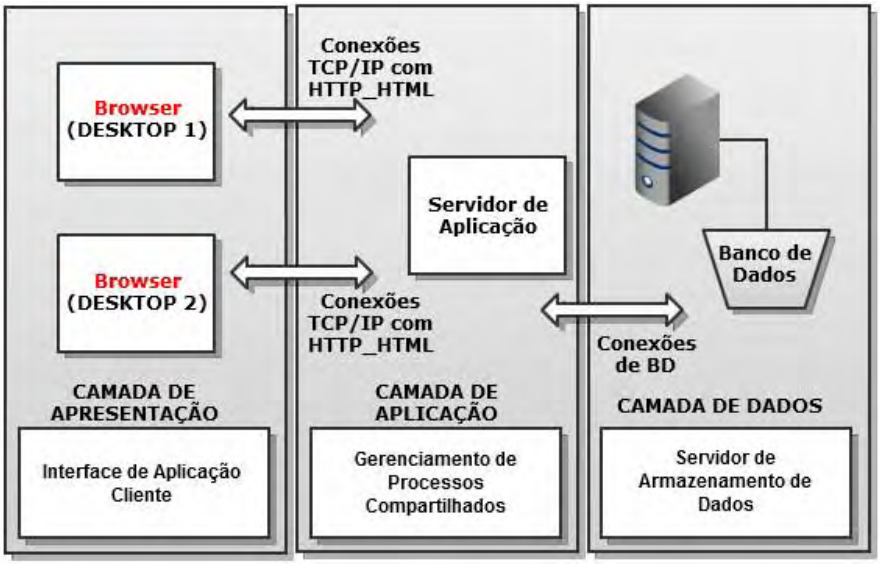


**Figura 6 - Arquitetura Cliente-Servidor Web-Based. Adaptado de (SADOSKI, 2002)**

Desta forma, qualquer mudança na camada de aplicação não reflete na obrigatoriedade de novas instalações e configurações na camada de apresentação. O modelo permite que servidores implementem novos e diferentes serviços e repositórios sem que haja a necessidade de mudança de instalação no aplicativo cliente.

O paradigma cliente-servidor em 3 camadas e *Web-based* adicionam novos atributos ao modelo desta arquitetura tradicional. A representação através da maior abstração de um ambiente de negócio na forma de componentes, ilustrado pelas diversas camadas, resulta em coordenação, padronização e disciplina no desenvolvimento de redes de comunicações.

### 2.3.4 Arquitetura Distribuída – N Camadas

A camada intermediária apresentada pela arquitetura em 3 camadas não é adequada a todo tipo de aplicação e equipamento. O mundo do desenvolvimento de *software* se tornou mais complexo nos últimos anos, onde a necessidade de novas funcionalidades segue tendências de melhor adaptabilidade, manutenabilidade, reusabilidade, interoperabilidade e escalabilidade (VMWARE 2007).

Diante deste cenário, a utilização de programação orientada a objetos, modularização, componentização, fez com que as arquiteturas convencionais dessem suporte a estes novos requisitos. Aplicações distribuídas baseadas em arquiteturas com n-camadas foi sugerida para que diversos equipamentos heterogêneos (sistemas operacionais, *hardware*, linguagem de programação) pudessem ser interligados em rede. No entanto, o gerenciamento distribuídos e heterogêneo deve implementar uma série de especificações voltadas para a padronização da complexidade de comunicação.

Aqui entra o conceito de *middleware*. Diante da heterogeneidade dos diversos equipamentos para a comunicação em uma rede, o *middleware* surge como um *software* de conectividade. Este permite a interação de múltiplos processos executando em uma ou mais máquinas, de modo a implementar vários tipos de camadas médias com suas funcionalidades específicas para o terminal em questão. A Figura 7 ilustra um exemplo de arquitetura em n-camadas de diferentes equipamentos para a conexão com servidores.

*Middlewares* são específicos para cada aplicação, onde a quantidade e funcionalidade dos módulos dependem diretamente da especificação de *hardware*, sistema operacional e linguagem de programação do paradigma cliente e servidor. Como ilustra a Figura 7, alguns módulos podem ser comuns para aplicações distintas.

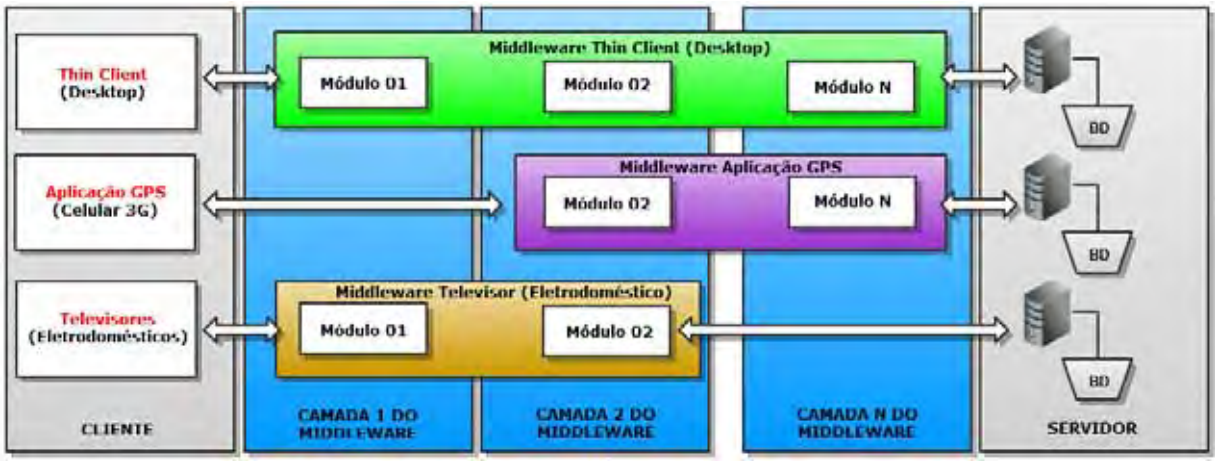


**Figura 7 - Arquitetura Distribuída em N Camadas**

No caso de um *middleware* de comunicação com um servidor para *Thin Client,* aplicação GPS e televisores, o módulo 02, representado de forma generalista, poderia ser responsável pelo canal de comunicação e negociação para troca de mensagens cliente/servidor. O *middleware* é essencial para migrar aplicações de *mainframe* para aplicações cliente/servidor provendo comunicação através de plataformas heterogêneas, assim como o mascaramento do *hardware*, sistemas operacionais e linguagens de programação (BRAY, 1998) (COULOURIS, 2007).

Além de ocultar a heterogeneidade, componentes de *middleware* fornecem um modelo uniforme para ser utilizado por programadores de aplicativos distribuídos. A invocação remota de objetos, notificação de eventos, acesso a banco de dados e processamento de transações são exemplos destes modelos. O *middleware* é capaz de oferecer serviço de propósito geral, podendo estar localizado entre a aplicação servidora e a plataforma de hardware cliente.

## 2.4 Comunicação entre Processos Distribuídos

A utilização de redes de comunicações norteadas por aplicações distribuídas desencadeou o desenvolvimento de vários modelos de comunicação entre processos. Com o objetivo de abstrair a codificação de transmissão em redes, os modelos de comunicação de processos aumentam a flexibilidade, interoperabilidade e portabilidade das aplicações baseadas em n camadas que precisam ser executadas em plataformas heterogêneas.

Dois modelos de comunicação para sistemas distribuídos são utilizados freqüentemente: Invocação a Métodos Remotos (RMI – *Remote Method Invocation*) e Chamada de Procedimentos Remotos (RPC – *Remote Procedure Call*). Os modelos são estruturados para arquitetura cliente-servidor (pedido e resposta) e solicitam abertura de canais de comunicação baseado tanto em UDP quanto em TCP. De maneira geral, pode-se considerar que a chamada de procedimento está para a invocação de objetos assim como o modelo RPC está para o modelo RMI.

Esta seção apresenta a base de comunicação entre processos conectados por datagramas e fluxos de dados, além do ponto padrão destino de processos (*soquetes*). Na seção 3.4, será discutido o modelo de

comunicação entre objetos distribuídos feita por meio da invocação a método remoto que servirá de conceito base para o projeto desta monografia.

### 2.4.1 Comunicação Síncrona e Assíncrona

Os canais de mensagens estruturam uma abstração de suporte para a coordenação entre aplicações que estão distribuídas, refletindo na não obrigatoriedade de uma associação direta e explícita entre os terminais de comunicação. Desta forma, a passagem de mensagens entre processos é fundamentada por duas operações de comunicação: *send* e *receive*.

A comunicação ocorre pelo envio de uma seqüência de bytes do processo produtor para o processo consumidor. Essa atividade de comunicação remetente-destinatário pode implicar na sincronização dos processos comunicantes.

Uma fila é associada a cada terminal de destino. Os processos remetentes enviam mensagens para serem adicionadas nesta fila remota, de modo que os processos destinatários possam consumir as mensagens da sua fila local.

Na forma de comunicação síncrona, ocorre uma sincronização entre os processos comunicantes. As operações *send* e *receive* causam bloqueio de mensagens subseqüentes. Quando um envio é realizado (s*end*), o processo remetente é bloqueado até que a recepção (*receive*) desta mensagem seja feita.

Já na forma de comunicação assíncrona, mensagens para destinatários são armazenadas em um *buffer* local do remetente, paralelizando a produção e a continuidade de execução do processo remetente. O uso da operação *send* é *não bloqueante*, ao contrário da operação *receive*, que pode ter variantes com e sem bloqueio. Na operação *receive não bloqueante,* o processo destinatário prossegue sua execução após realizar a operação *receive*. Neste caso, o processo fornece um *buffer* para ser preenchido em *background* e será notificado que possuem dados a serem lidos. Isso pode ser feito baseado em *polling* ou em interrupção (COULOURIS, 2007).

A comunicação não bloqueante para o destinatário aparenta ter melhor eficiência, mas apresenta uma complexidade adicional da necessidade de leitura fora do fluxo normal de execução do processo. Além disto, favorece os contras desta técnica não bloqueante, o fato de que a operação *receive não bloqueante* não tem vantagens em ambientes que suportam múltiplas *threads* (Sistema Java) em um único processo. A operação pode ser executada em uma *thread,* enquanto as restantes permanecem ativas.

### 2.4.2 Soquetes, Datagramas UDP e Fluxo TCP

Como visto na seção anterior, os canais de mensagens estruturam um suporte para a comunicação entre os aplicativos distribuídos. Estes canais de comunicação são baseados nos protocolos TCP (*Transmission Control Protocol*) e UDP (*User Datagram* Protocol), que fornecem recursos para a comunicação via Internet ou redes locais.

Enquanto o IP (*Internet Protocol*) identifica computadores para a comunicação, os protocolos UDP e TCP fornecem a comunicação de processos para processos. No entanto, apenas a utilização destes protocolos não identifica o correto processo destinatário, necessitando assim de mais um referencial. Isto é solucionado com o uso de portas.

Portas são representações de pontos de entradas (inteiro de 16bits) em um computador que permitem a comunicação direta com programas específicos. Cada porta pode ser usada por um programa ou serviço diferente. Desta forma, com apenas um endereço IP pode-se ter milhares de serviços mapeados por portas correspondentes para a comunicação remota desse computador.

O datagrama UDP tem um cabeçalho curto que inclui os números de porta de origem e destino, um campo de comprimento e uma soma de verificação. O UDP, por ser um protocolo não orientado a fluxo, não possui nenhum mecanismo de confiabilidade de entrega. Seu uso está restrito aos serviços que não exigem integridade da entrega das mensagens. O UDP fornece meios de transmissão de até 64Kbytes entre os pares de processos distribuídos (MORIMOTO, 2008).

Em contra partida, o protocolo TCP fornece um serviço confiável para a entrega de mensagens longas por meio do fluxo de dados. O protocolo é orientado a conexão. Desta forma, antes da transferência dos dados há a necessidade da criação de um canal de comunicação bidirecional. No entanto, os nós intermediários de infra-estrutura de rede não têm conhecimento desta conexão, e ,por este motivo, nem todos os datagramas seguem a mesma rota em uma transmissão TCP (KSHEMKALYANI, 2008).

Os dois protocolos utilizam a abstração de *soquete*. Soquete é um ponto de destino para a comunicação entre processos e são originários do UNIX BSD. Atualmente o conceito é encontrado na maioria das versões Linux, MacOS e Windows (MORIMOTO, 2008). A Figura 8 ilustra a estruturação de toda a comunicação entre dois terminais.

A mensagem é transmitida do soquete de um processo para o soquete de outro processo. Diante disto, para que o processo receba mensagens, seu soquete deve estar relacionado a uma porta local e o endereço IP do computador. Os processos podem utilizar o mesmo soquete para receber e enviar mensagens.

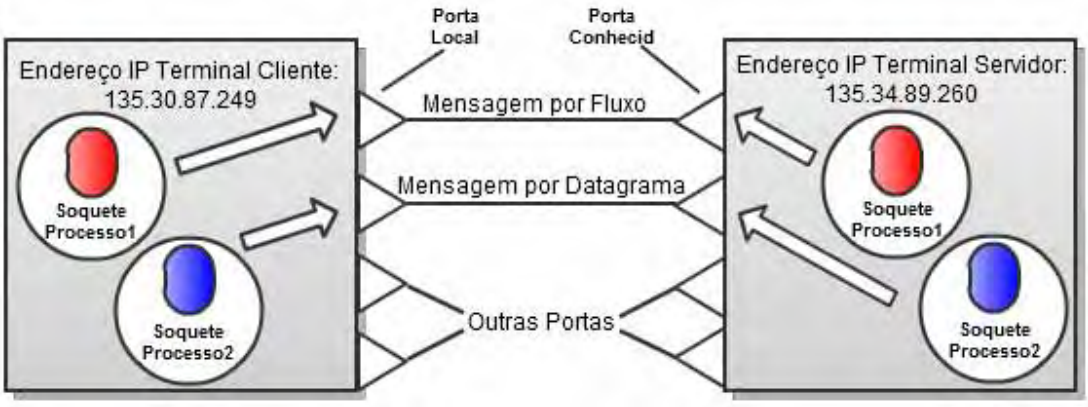


**Figura 8 - Comunicação entre Terminais – Soquetes. Adaptado de (COULOURIS, 2007)**

Qualquer processo pode fazer uso de diferentes portas para receber mensagens, mas um processo não pode compartilhar portas com outros processos no mesmo computador (COULOURIS, 2007). Os processos que usam *multicast* IP são uma exceção, pois eles compartilham portas. Cada soquete é associado a um protocolo em particular (UDP ou TCP).

### 2.4.3 Estruturação da Comunicação em *Kernel* UNIX

Os núcleos dos sistemas operacionais são arquitetados com as operações de comunicação entre processos distribuídos. Utilizando mecanismos básicos de agrupamento de funcionalidades, os núcleos implementam chamadas de sistemas para a comunicação entre os processos. Estas chamadas de sistemas são utilizadas pelas linguagens de programação, oferecendo ao usuário programador abstração da comunicação entre os processos.

No *kernel* UNIX BSD, estas primitivas de comunicação são oferecidas como chamadas de sistemas implementadas como uma camada sobre os protocolos UDP/TCP (TANENBAUM e VAN STEEN, 2007). As mensagens são direcionadas para o endereço de soquete (composição do endereço IP e o número de porta local).

A criação de um soquete em *kernel* UNIX é iniciada com a chamada da função *socket* e seus argumentos que especificam o domínio da comunicação, o tipo de transmissão (datagrama ou fluxo), e opcionalmente um protocolo específico. Este retorna um descritor que estará ativo até o momento que o soquete é fechado ou quando os processos que estarão usando este descritor terminem. A comunicação bidirecional entre dois processos (situados no mesmo computador ou não) é realizada por meio do descritor de soquete. No entanto, este deve ser associado a um endereço de soquete.

A chamada *bind* do núcleo UNIX faz a vinculação entre o descritor instanciado e a estrutura referente ao endereço de soquete. Ao contrário do que a API (*Application Programming Interface* ) Java implementa, o UNIX realiza duas chamadas de sistema para a criação da estrutura de comunicação: uma para criação do soquete e outra para a vinculação.

No caso do UNIX, soquetes são endereços públicos. Estes endereços podem ser utilizados como destino por diferentes processos. Logo que um processo faz a vinculação do soquete instanciado com a estrutura de endereço de soquete, outros processos poderão contatá-lo através do endereço de soquete. Desta forma, todo processo que deseja receber mensagens através de um soquete, deve primeiro realizar a vinculação a uma estrutura de endereço e torná-lo conhecido para outros processos (KSHEMKALYANI, 2008).

A comunicação por datagrama exige a instanciação de dois soquetes a cada mensagem enviada. Tanto o processo cliente quanto o processo servidor, utilizam chamadas de sistemas para a criação do soquete. A Figura 9 ilustra os passos de execução no cliente e servidor.

A chamada pela função *socket* especifica a comunicação como sendo a Internet, a utilização de datagramas, e o ultimo argumento faz com que o núcleo utilize o protocolo UDP. Cliente e Servidor realizam a vinculação do descritor e do endereço de soquete a partir da chamada da função *bind*. No caso do servidor, o endereço de soquete deve especificar o endereço que contém a porta do serviço mais o IP e torná-la conhecida para o remetente. Já para o cliente, o número de porta local poderá ser qualquer disponível.

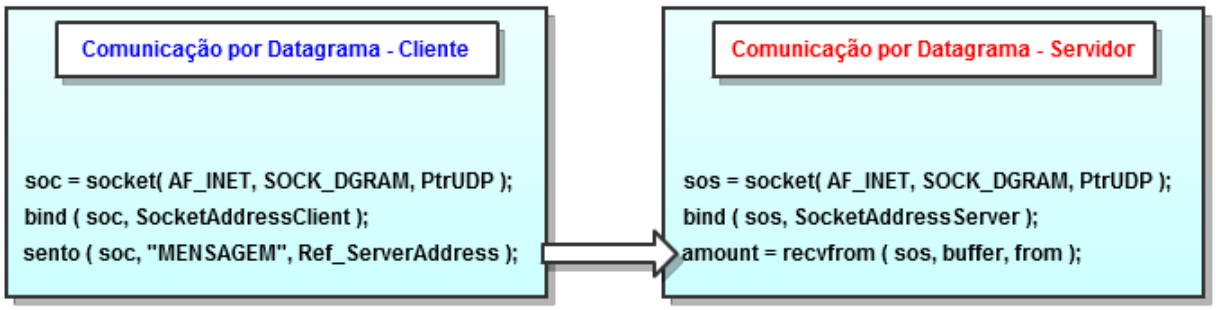


**Figura 9 - Comunicação por Datagrama Cliente-Servidor. Adaptado de (TANENBAUM, 2007)**

O processo cliente utiliza a chamada *sendto* para o envio da mensagem. Os argumentos descrevem o soquete que deverá ser usado para transmissão, a mensagem a ser recebida no servidor e a referência de soquete destino. Esta chamada de envio retorna o número de bytes transmitidos, no caso de emprego por comunicação TCP.

O processo servidor utiliza a chamada *recvfrom* para receber a primeira mensagem da fila ou, esperar até que ela chegue. O argumento desta função explicita o soquete local para o recebimento e área de memória para o armazenamento, além de uma referência para estrutura de soquete do remetente. A comunicação por fluxo segue as mesmas idéias básicas da comunicação por datagrama. A utilização do protocolo TCP exige, antes da troca de mensagens, a criação de uma conexão entre os processos. O

estabelecimento do canal é assimétrico, um soquete desempenha papel cliente (solicitação de conexão) e outro papel servidor (esperando requisição).

Assim que a conexão é criada, os soquetes estarão associados podendo ser usados para a troca de dados. Os dados são lidos imediatamente na mesma ordem em que foram armazenados no *buffer* receptor. Este *buffer* receptor bloqueia o processo caso a fila esteja vazia. No caso do *buffer* estar cheio, o processo emissor é bloqueado

A Figura 10 ilustra a comunicação por fluxo no *kernel* UNIX. O servidor espera pela solicitação do cliente e aceita o pedido para a conexão e cria outro processo para a comunicação com o cliente. Isto permite que o servidor possa continuar a esperar por pedidos de novas conexões.

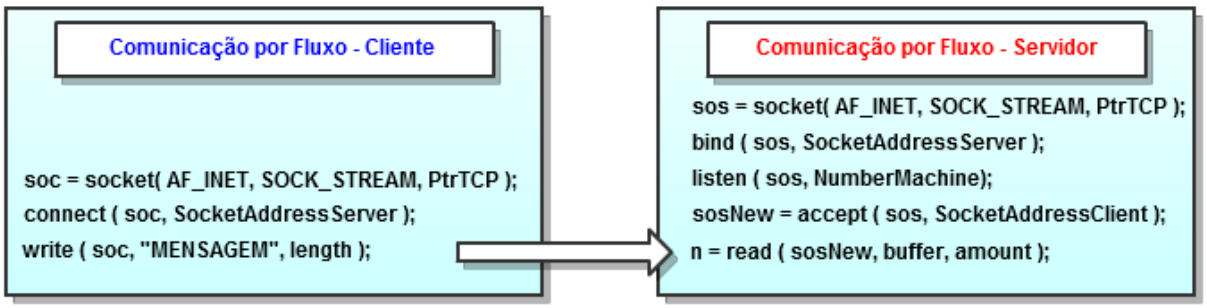


**Figura 10 - Comunicação por Fluxo Cliente-Servidor. Adaptado de (TANENBAUM, 2007)**

O servidor espera por conexões no soquete instanciado através da operação *listen.* A chamada indica o número máximo de clientes que podem ser respondidos e aceitos para o soquete. Para aceitar conexões e atribuir a instanciação a outra variável de soquete, o UNIX utiliza a chamada *accept*. Neste caso, a variável pelo soquete original instanciado fica livre para aguardar novos pedidos (COULOURIS, 2007).

No caso do processo cliente, este utiliza a chamada *connect* para solicitação de conexão com o soquete servidor. A vinculação necessária em comunicação por datagrama através da chamada *bind*, é dispensável quando se utiliza a operação de conexão por fluxo. A chamada *connect* automatiza esta tarefa, não havendo necessidade da vinculação explícita.

As operações de leitura e escrita (*read* e *write*) são usadas para o recebimento e envio de mensagens logo após a criação do canal. Ao fechar um soquete, o processo envia todos os dados do seu *buffer* de saída para o processo receptor com uma indicação de finalização da conexão. Desta maneira, quando um processo termina ou falha, todos os seus soquetes são finalizados, e qualquer tentativa de comunicação de outros processos será notificada como conexão desfeita.

## 2.5 Considerações Finais

O principal fator de motivação para estruturação de sistemas distribuídos é o compartilhamento de recursos como, por exemplo, imagens de sistemas operacionais, registros de banco de dados, páginas web, nos quais são gerenciados por servidores apropriados para cada recurso em questão. Diante desta construção, estes recursos são acessados por clientes específicos (COULOURIS, 2007). Como visto na seção 2.3.4, a interação entre cliente e recursos alocados no servidor pode depender de *software* de conectividades nos quais implementam vários tipos de camadas médias com funcionalidades específicas para o terminal em questão.

Portanto, a grande parte dos sistemas distribuídos é construída e organizada com referencia em um, entre vários, modelo de arquitetura de sistemas (TANENBAUM e VAN STEEN, 2007). Estes sistemas são constituídos por componentes na maioria heterogêneos que coordenam suas ações apenas pela comunicação distribuída de troca de mensagens o que reflete na otimização de tarefas, prontificando instalação e manutenção de aplicativos.

# 3. MODELAGEM ORIENTADA A OBJETOS DISTRIBUÍDOS

## 3.1 Considerações Iniciais

A comunicação entre clientes e servidores pode ser estruturada por diferentes formas, mas todas elas são evoluções dos conceitos vistos no capítulo anterior. Alguns desses modelos de comunicação refletem em uma maior flexibilidade, interoperabilidade e portabilidade das aplicações distribuídas, e conseqüentemente fornecem transparência para as invocações de chamadas remotas. Este Capítulo apresenta a arquitetura de comunicação entre objetos distribuídos realizada por meio da invocação a método remoto. Esta modelagem de comunicação entre objetos distribuídos é a base estabelecida no enlace de comunicação sem fio cliente-servidor do projeto WSE-OS.

## 3.2 Contextualização da Comunicação Distribuída

Tanto a Chamada de Procedimento Remoto (RPC – *Remote Procedure Call*) quanto a Invocação a Métodos Remotos (RMI – *Remote Method Invocation*) são infraestruturas de programação para aplicativos distribuídos. Enquanto a RPC é uma extensão das chamadas convencionais de procedimentos (linguagens procedurais) para chamadas de procedimentos remotos que estão em processos e computadores diferentes, a RMI é uma ampliação do modelo orientado a objetos responsável por estender a invocação de métodos locais para a invocação de objetos que estão em processos e computadores diferentes (COULOURIS, 2007).

Embora comum e eficiente, a conexão entre processos distribuídos por soquetes sem o auxílio de infraestruturas não fornece possibilidade de acesso (invocação) aos métodos de um objeto. A conexão pode ser considerada de baixo nível, pois a troca de dados baseado em fluxo/datagrama não é estruturado (LOBO, 2005). Neste caso, a aplicação cliente e servidor devem impor uma estrutura local para a interpretação destes bytes. Ao contrário disto, a modelagem distribuída de aplicações orientada a objetos permitem a invocação de métodos remotos com especificação de objetos como argumentos de

chamada. Toda esta especificação pode ser realizada através de interfaces remotas, que será analisado nas próximas seções.

Existem mecanismos de RMI que estão integrados a linguagens de programação. O Java RMI (JRMI, 2010) é um exemplo na qual toda a infraestrutura de invocação a objetos foi adicionado à linguagem por permitir uma notação adequada de definição de interfaces. Desta forma, parâmetros de entrada e saída dos métodos remotos são mapeados no uso normal de parâmetros da linguagem (COULOURIS, 2007).

A técnica de integração de infraestrutura com a linguagem é conveniente quando os diferentes aplicativos podem ser codificados em uma mesma linguagem. Diante disso, a invocação de métodos locais ou remotos é escrita conforme uma mesma representação.

No entanto, diversos aplicativos são codificados em diferentes linguagens guiados por necessidades inerentes a elas. As *IDLs* (*Interface Description Language*) permitem que programas escritos em diferentes linguagens realizem comunicação entre seus objetos. O objetivo da Linguagem de Definição de Interface é fornecer mecanismos de notação para definição de interfaces. Ou seja, especificar parâmetros de entrada e saída de um método de forma a abstrair diferenças nas implementações dos aplicativos.

Exemplo destes mecanismos não integrados a linguagens são as IDLs CORBA e IDLs DCOM. O CORBA (*Common Object Request Broker Architecture*) é um projeto da OMG (*Object Management Group*) que permite as aplicações distribuídas, implementadas em diferentes linguagens, comunicarem entre si. As aplicações são arquitetadas por meio de objetos CORBA, os quais implementam a linguagem de definição de interfaces.

Como exemplo, um cliente escrito em C++ pode localizar e invocar um objeto de uma aplicação servidor escrita em Delphi. Seguindo a mesma idéia, o DCOM (*Distributed Component Object Model*) permite que o modelo de objetos possa ser utilizado para a distribuição entre clientes e servidores, provendo a interoperabilidade de software (MOLINARI, 2002). Ao contrário do CORBA, O DCOM é uma tecnologia proprietária da Microsoft.

De forma resumida, a implementação de aplicações distribuídas é realizada por linguagens de programação de uso geral, no entanto com as facilidades oferecidas por estas infraestruturas que fornecem funcionalidade de comunicação, localização, migração, e detecção (PONNEKANTI e FOX, 2004).

Por esta monografia tratar de um projeto que visa o desenvolvimento de um *middleware* baseado em comunicação de objetos distribuídos, as próximas seções pormenorizarão os módulos cliente e servidor da arquitetura RMI de modo a ilustrar o estado da arte.

## 3.3 Representação Externa e *Marshalling/Unmarshalling*

As informações transmitidas entre dois computadores, independentemente da forma de transmissão, são seqüências simples de dados. Embora estejam encapsuladas em diversos protocolos de redes, as mensagens não possuem estrutura de dados como as informações armazenadas nos aplicativos em execução local. Quando a mensagem é enviada, ocorre uma simplificação da estrutura do dado em um desmembramento seqüencial de bytes.

Diversas arquiteturas de *hardware* e *software* possuem discrepâncias de interpretação destas seqüências. Representação de tipos primitivos possui variantes que podem causar falha de comunicação por falta de protocolo comum, caso não sejam indicadas. Tipo de dados inteiros é um exemplo deste cenário. A ordenação *big-endian* e *little-endian* devem ser devidamente especificadas na programação de protocolos de comunicação. Outro exemplo é a representação de caracteres. A transmissão normalmente segue a codificação ASCII (*American Standard Code for Information Interchange*) (um byte por caracter) ou o padrão Unicode (dois bytes por caracter).

Diante disto, invocações a métodos remotos devem possuir mecanismos de simplificação comum de objetos no cliente e servidor. Esta representação externa de dados é conduzida por operações de empacotamento (*marshalling*) e desempacotamento (*unmarshalling*). O empacotamento abstrai toda a estrutura e item de dado de um objeto em uma seqüência conveniente para a transmissão, enquanto que o desempacotamento realiza a desmontagem da mensagem para a produção do conjunto de itens e estruturas originais (TANENBAUM e VAN STEEN, 2007).

Tanto a representação de dados comum (CDR – *Common Data Representation*) do CORBA quanto a serialização de objetos da linguagem Java, estão relacionados a um representação externa de tipos estruturados e primitivos. Nestes casos, atividades de *marshalling* e *unmarshalling* são executadas por uma camada de *middleware*, sem nenhum envolvimento por parte do programador de aplicativo (COULOURIS, 2007). Embora seja eficiente e necessária para o suporte RMI nas aplicações, a representação externa de dados tem uso genérico em documentos estruturados, objetos e estrutura de dados para o armazenamento em arquivos.

## 3.4 Referências a Objetos Remotos

O princípio da comunicação entre processos distribuídos é baseado no protocolo IP e a porta de serviço correspondente ao aplicativo. A invocação de um método em um objeto remoto exige a identificação do processo e do objeto vinculado ao método solicitado. Desta forma, a estrutura RMI necessita de *Referências a Objetos Remotos* como um identificador exclusivo de objetos distribuídos no tempo e espaço.

A referência de objeto remoto é transmitida na mensagem de invocação cliente para encaminhar a correta ativação do método para o objeto alvo. Por existir diversos processos contendo objetos remotos, a reutilização de referencias não é realizada. Alguns processos invocadores podem manter referências obsoletas e qualquer tentativa de ativar um objeto inexistente, produzirá um erro (TANENBAUM e VAN STEEN, 2007).

A exclusividade de identificação dos objetos em diferentes processos remotos é construída, nos casos mais simples, pela junção do endereço IP do computador na rede, número da porta na qual o processo reside, hora de criação e um número de controle de instanciações para objetos no processo correspondente. O número de controle de instanciações é concatenado a toda nova referência e sempre incrementado a cada criação de objeto. Esses quatro identificadores mais um campo reservado para o detalhamento da interface do objeto (métodos oferecidos) constituem a referência exclusiva de um objeto remoto. A Figura 11 ilustra uma representação da referência.

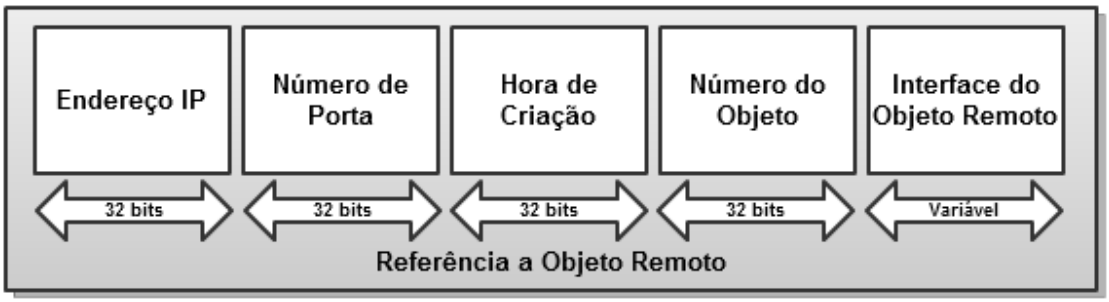


**Figura 11 - Representação Referência de Objeto Remota. Adaptado de (COULOURIS, 2007)**

O objeto remoto normalmente reside no processo pai de criação e mantém sua execução até o momento em que o processo é suspenso ou bloqueado. Nos casos em que o objeto é migrado para um processo diferente, a referência de objeto remoto possui algumas alterações nos campos para o correto roteamento. Referências de objeto remoto CORBA, adicionam um campo na estrutura de referência de modo que a mensagem seja redirecionada para um repositório de interfaces. Isto permite que a definição de IDL (*Interface Description Language*) seja recuperada em tempo de execução (COULOURIS, 2007).

## 3.5 Modelo de Objeto Distribuído

Sistemas de objetos distribuídos podem adotar diferentes arquiteturas de comunicação. No modelo cliente-servidor, objetos são gerenciados por processos servidores e os clientes invocam os métodos através da infraestrutura RMI. Nesta arquitetura é utilizado o protocolo requisição-resposta baseado em três primitivas de comunicação.

As operações *doOperation, getRequest* e *sendReply* garantem a associação das respostas a respectivas requisições. O método *doOperation* é utilizado para a invocação de métodos remotos. Os argumentos especificam o objeto destino através de uma referência de objetos, e duas outras informações que identificam o método e seus argumentos. Como ilustra a Figura 12, o resultado desta invocação é uma resposta RMI.

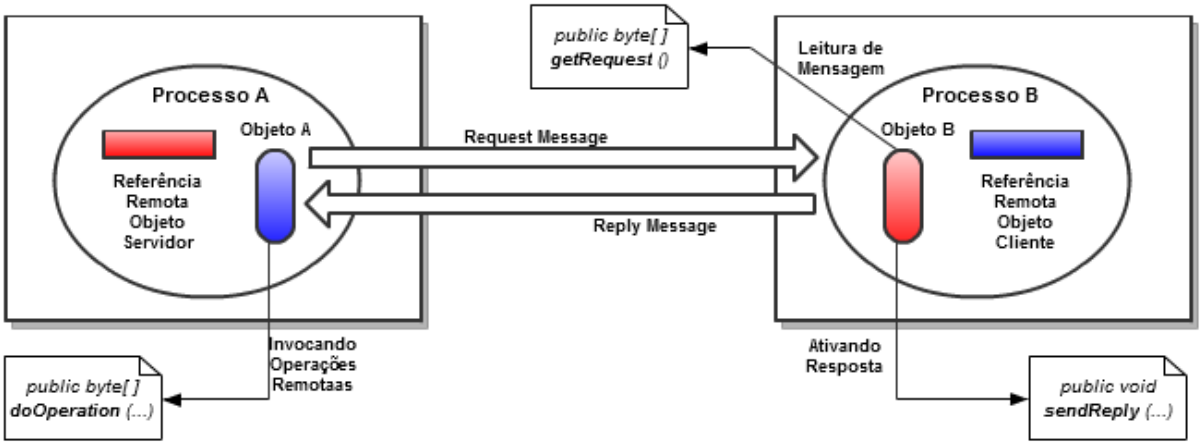


**Figura 12 - Comunicação Requisição-Resposta**

Logo após a invocação, o cliente espera pela resposta para absorver o resultado e repassar para o objeto invocador. Nesta situação, o invocador é bloqueado até que o servidor execute a solicitação e repasse os dados processados. Já no processo servidor ocorre a execução da operação *getRequest* e *sendReply*.

A primeira operação (*getRequest*) é utilizada para que o processo destino consiga ler as requisições de serviço, ou seja, localizar o objeto desejado e executá-lo. A segunda operação (*sendReply*) é utilizada apenas para enviar a mensagem de resposta para o cliente. As mensagens trocadas pelos clientes e servidor seguem um pacote definido de cinco campos, como ilustra a Figura 13.

O primeiro campo identifica se a mensagem é de requisição ou resposta. O segundo campo *requestId* armazena um identificador de mensagem. É através deste campo que os clientes sabem que a mensagem é de resposta da requisição corrente e não de uma chamada anterior atrasada (COULOURIS, 2007).

O *requestId* é gerado pelo cliente e enviado na mensagem. O processo servidor copia este valor e retransmite para o processo invocador. O terceiro campo é uma referência do objeto remoto. O quarto e quinto campos são especificações do método e os seus argumentos respectivamente.

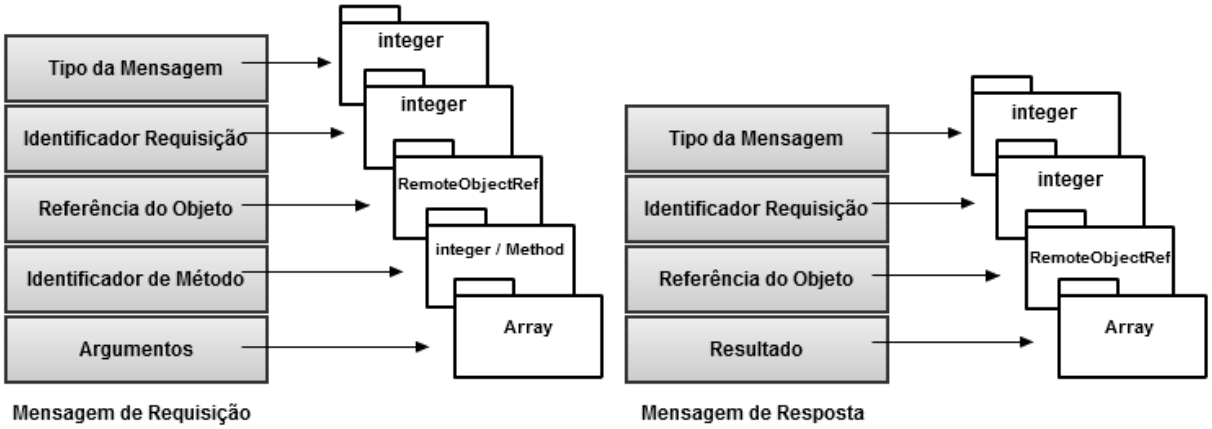


**Figura 13 - Estrutura Mensagem Requisição-Resposta**

### 3.5.1 Invocações Locais e Remotas

Na modelagem orientada a objetos, os processos possuem diversos objetos nos quais trocam mensagens com outros objetos residentes no mesmo cliente/servidor ou não. Invocações a métodos entre objeto localizados no mesmo processo são invocações a métodos locais. Ao contrário desta situação, mas seguindo os princípios de orientação a objetos, a comunicação entre objetos remotos residentes em computadores distintos necessita de dois conceitos primordiais: Referência de Objeto Remoto e Interface Remota.

Para realizar a invocação de métodos, os objetos clientes possuem a referência de objeto remoto destinatário. Este conceito da programação orientada a objetos é estendido para permitir que objetos em aplicativos distribuídos possam implementar RMI. Como visto na seção 3.2, a referência de objetos remotos é um identificador que pode ser utilizado por todo o sistema para referenciar um objeto único no tempo e espaço.

Além disto, os objetos clientes possuem uma interface remota do objeto alvo servidor. Esta interface especifica os métodos do objeto servidor que podem ser invocados de forma remota. Como ilustrado na Figura 14, objetos de processos separados lógica e fisicamente apenas tem permissão de acessar métodos pertencentes à interface remota.

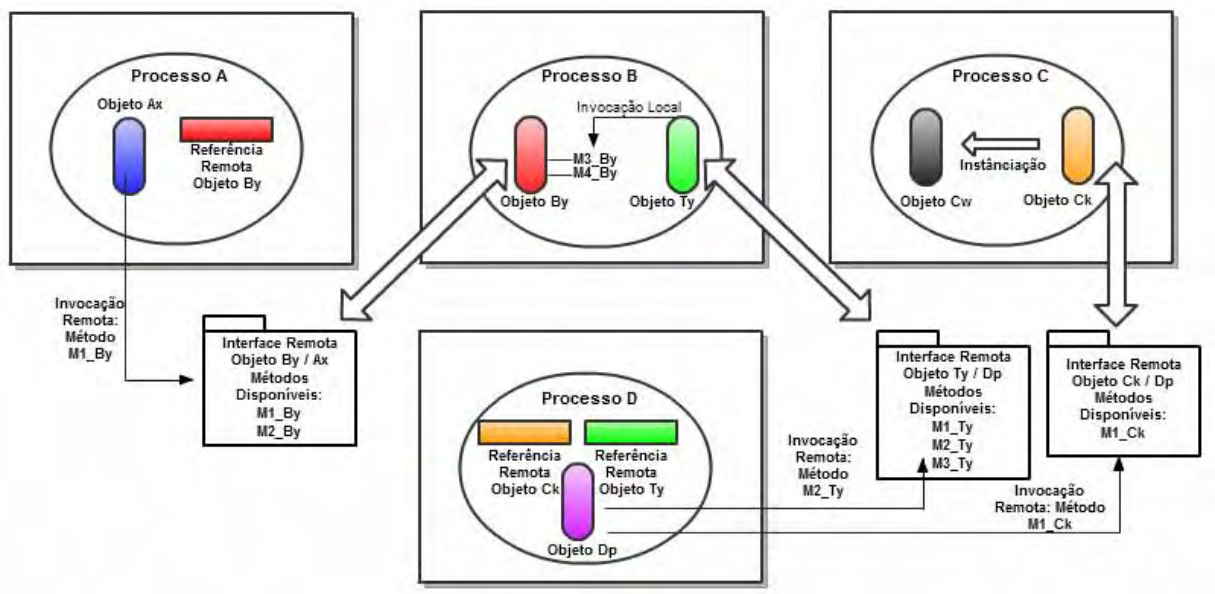


**Figura 14 - Invocação Local e Remota**

Embora o objeto Ax do processo A faz invocação ao método M1_By do objeto By processo B, ele não tem acesso aos métodos M3_By e M4_By. A definição da interface remota para o processo A apenas tornou pública os três primeiros métodos ao objeto Ax. Além disto, o objeto Ax não possui uma referência remota de objeto Ty. Por outro lado, o objeto Dp do processo D possui referências remotas tanto para o objeto Ck processo C quanto para objeto Ty processo B.

Quando uma invocação levar a instanciação de um novo objeto, este normalmente ficará dentro do processo onde o construtor foi ativado. É o caso da instanciação do objeto Cw, ativado pelo objeto Dp através da interface remota do objeto Ck. Nesta situação, caso o objeto instanciado contenha uma interface remota para o acesso RMI, ele será um objeto remoto com uma referencia de objeto remoto.

Como descrito anteriormente, o sistema CORBA fornece linguagem de definição de interfaces para a interoperabilidade de comunicação de aplicativos escritos em diversas linguagens na qual tenha disponível um compilador de IDL. Já no Java RMI, as interfaces são definidas como interfaces públicas Java da própria linguagem. A capacidade de se tornarem remotas vem apenas pela extensão na declaração como sendo *Remote*. A herança múltipla de interfaces é ponto comum tanto da IDL CORBA quanto na linguagem Java RMI (MENDONÇA, 2008).

## 3.6 Estrutura Arquitetural RMI Request-Reply

A estruturação da arquitetura RMI é composta por vários módulos separados, mas dependentes entre si para a realização da invocação a métodos remotos. A Figura 15 ilustra estes módulos fundamentais localizados no cliente e servidor. A arquitetura RMI é baseada no módulo de comunicação, módulo de referência remota e o software RMI (*Proxy* + *Despachante* + *Esqueleto*).

Os módulos de comunicação cliente e servidor são baseados no protocolo requisição-resposta vista na seção 3.3, mas apta a extensão *publish-subscribe* baseada em eventos (COULOURIS, 2007) que será analisada na próxima seção. Estes módulos são responsáveis por fornecer uma semântica de invocação utilizando os três primeiros campos da mensagem de requisição, ilustrado na Figura 13. A identificação de métodos e o empacotamento dos seus argumentos são realizados pelo software RMI que apenas repassa para o módulo de comunicação completar a mensagem. No servidor, o módulo de comunicação encaminha a mensagem com a referência local do método a ser invocado para o *despachante* da classe do respectivo objeto.

O módulo de referência remota administra uma *tabela de objetos remotos* que mantém correspondências de referências locais e remotas. Quando ativado, o módulo cria a referência de objeto remoto (Figura 11) solicitado, na situação dele ainda não existir. No caso do módulo de referência remota servidor, este tem a responsabilidade de estabelecer a correspondência de referencia remota para a referência local direcionando as ações ao objeto de acesso.

Já o módulo de referência remota cliente tem função similar ao servidor, no entanto a correspondência remoto-local é para o *proxy* do objeto invocado mapeado no cliente. Os dois módulos de referência remota são chamados pelo software RMI quando seus componentes estão realizando ações de empacotamento e desempacotamento. O correto encaminhamento ao objeto local ou remoto depende da consulta de correspondência armazenado nos módulos de referência remota (KSHEMKALYANI, 2008).

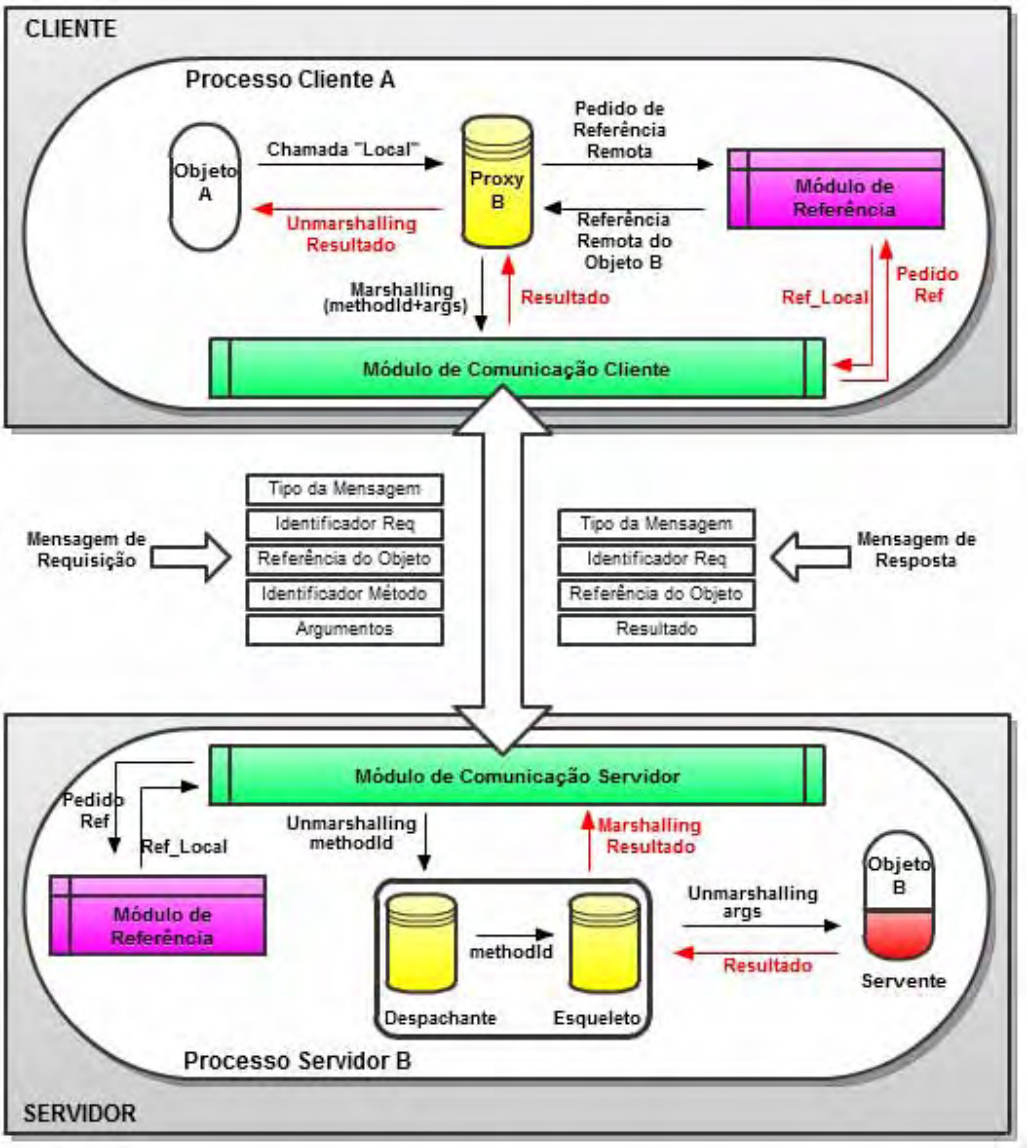


**Figura 15 - Arquitetura RMI *Request-Reply***

Nos sistemas distribuídos a função de administração das tabelas dos módulos de referência remota é realizada por um *Vinculador. O* serviço *RMIregistry* e *Naming Service* são vinculadores do Java RMI e CORBA, respectivamente. Eles são utilizados pelos aplicativos servidores para registrar seus objetos remotos. Como mostra a Figura 15, o objeto servidor B terá sua referência registrada na tabela do módulo de referência remota servidor, e o *proxy* para B será armazenado como entrada na tabela no módulo de referência remota cliente.

O software RMI é uma camada intermediadora entre objetos dos processos e os dois módulos de comunicação cliente/servidor. Toda mensagem de requisição ou resposta é encaminhada a esta camada pelos módulos de comunicação e de referência remota. O software RMI é composto por três elementos básicos: *Proxy*, *Despachante* e *Esqueleto*.

O *Proxy* é um componente representante de objeto servidor e está localizado em todo cliente RMI que possui referência de objeto remoto. O objetivo é tornar transparente ao cliente os processos de empacotamento/desempacotamento, referência a objetos remotos, e o próprio envio de mensagens (TANENBAUM e VAN STEEN, 2007). Desta forma, a invocação a métodos remotos é abstraída como uma simples chamada local.

O *Proxy* possui classes que implementam os métodos da interface remota do objeto remoto que ele representa, garantindo que as invocações sejam corretas e apropriadas para o objeto representado servidor. Cada método do *Proxy* empacota a identificação do método e os argumentos necessários, além de fazer uma consulta sobre a referência remota do objeto ao módulo de referência remota. É no *Proxy* que ocorre atividades de *Marshalling* e *Unmarshalling* cliente descritas na seção 3.1 deste capítulo. Logo após a atividade de *Marshalling*, a mensagem é enviada para o computador remoto. O *Proxy* aguarda pela resposta, e ao receber a mensagem de resposta realiza a atividade de *Unmarshalling* de forma a retornar os resultados para o invocador.

Os elementos *Despachante* e *Esqueleto* cooperam para executar, respectivamente, as ações de direcionamento de método *Esqueleto* e encaminhamento para o *Servente* do Objeto Alvo. O *Servente* é uma instancia de uma classe do objeto remoto. Ele é responsável por executar as solicitações remotas repassadas pelo *Esqueleto*. Sua criação é ativada na instanciação do objeto remoto e permanecem em uso até o fim do processo.

Para cada classe do objeto remoto servidor existe um *Despachante* e *Esqueleto* que fazem o direcionamento apropriado da mensagem de requisição. O *Despachante* utiliza o identificador de métodos (*methodId*) na mensagem de requisição para selecionar o método correspondente no *Esqueleto*. Tanto o *Proxy* quanto o *Despachante* utilizam o mesmo *methodId* para representação dos métodos de interface remota.

Toda classe de objeto remoto tem um *Esqueleto* que implementa os métodos da interface remota, mas que não personificam o objeto alvo. Na realidade, a função do *Esqueleto* é desempacotar os argumentos da mensagem de requisição e invocar o método correspondente no *Servente*. Ao terminar a invocação, o resultado é empacotado pelo *Esqueleto* e enviado para o *Proxy* que realizou a invocação.

A geração de classes para os componentes do software RMI é automática. *Proxy*, *Despachante* e *Esqueleto* dependem do compilador de interface RMI para o seu funcionamento. O compilador JavaRMI define os métodos de um objeto como uma interface Java implementada dentro da classe do objeto remoto gerando as classes de *Proxy, Despachante* e *Esqueleto* a partir da classe do objeto remoto (COULOURIS, 2007).

## 3.7 Eventos e Notificações

A estrutura arquitetural RMI baseada no protocolo requisição-resposta pode ser adaptada a suportar o paradigma *publish-subscriber*. Esta extensão baseada em eventos permite que objetos distribuídos sejam notificados sobre ações que ocorrem em outros objetos remotos (BACON, 2008). O modelo publica tipos de eventos para a monitoração, onde interessados reagem a uma alteração notificada pelo objeto.

Ações de pressionar botões ou movimentar o mouse podem ser considerados eventos em aplicativos interativos que mantém uma monitoração sobre o estado de periféricos de hardware. Objetos interessados na mudança de estado são notificados e podem invocar algum método ou mesmo finalizar o processo em execução. As notificações podem ser usadas em diferentes situações para diversas finalidades.

O paradigma de publicação e registro trabalha de forma assíncrona, onde a fonte de monitoração pode gerar eventos de diferentes tipos. Por este motivo, cada evento possui atributos que identificam o nome do objeto, operação, parâmetros e momento da ação. Os tipos de eventos são necessários para a inscrição nos objetos de interesse. Desta forma, quando o evento do tipo especificado ocorre no objeto monitorado, as partes interessadas serão notificadas. O CORBA possui um serviço de orientação a eventos chamada CORBA *Event Service* responsável por tratar um sistema baseado no paradigma *publish-subscribe* (ULRIKE et. al., 2008).

Na Figura 16 ilustram-se os componentes que participam dos sistemas distribuídos baseados em eventos. O principal componente desta arquitetura é o serviço de eventos que mantém um banco de dados das ações publicadas e os interesses dos assinantes. Eventos de um objeto são publicados neste serviço. Objetos remotos informam a este banco de dados sobre o tipo de eventos que estão interessados. Quando ocorre um evento no objeto monitorado, uma notificação é entregue ao objeto assinante.

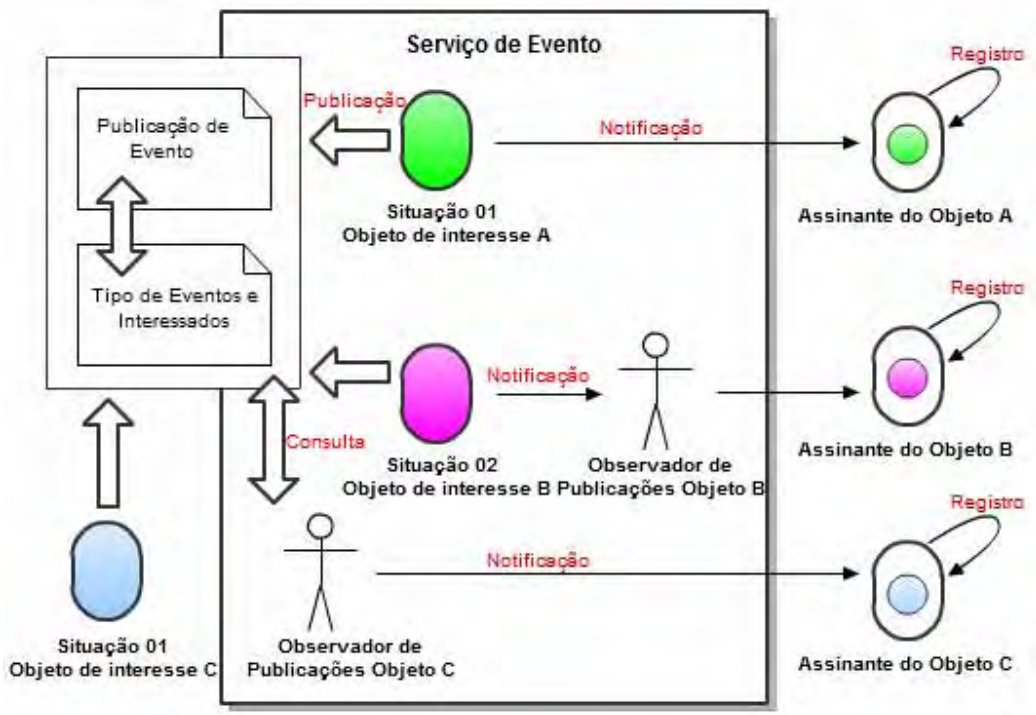


**Figura 16 - Arquitetura Evento e Notificação. Adaptado de (COULOURIS, 2007)**

Os objetos de interesse são objetos que modificam seus estados por invocarem métodos de suas classes. Esta modificação pode ter interesse em outros objetos remoto como, por exemplo, a digitação em um teclado cliente pode ativar um aviso na tela de visualização do servidor remoto.

Caso exista um observador responsável pela monitoração, o objeto de interesse pode ser independente do serviço de evento. Na situação do próprio objeto enviar mensagens de notificação para os objetos remotos assinantes, ele é considerado parte do serviço de eventos.

Os objetos observadores têm a função de desvincular o objeto de interesse do seu assinante. Esta situação é conveniente quando um objeto de interesse possui diversos assinantes. Diversos objetos assinantes podem ter diferentes interesses sobre o objeto monitorado. A complexidade de executar toda a lógica para direcionar as notificações entre as necessidades dos assinantes faz com que o desempenho da arquitetura diminua. Portanto, objetos observadores comportam-se como roteadores de notificações para os diferentes objetos assinantes.

Três situações são descritas na Figura 16. No primeiro caso, o próprio objeto de interesse repassa as notificações ao assinante daquele evento. Não há a necessidade de um observador e o objeto faz parte do serviço de evento. No segundo caso, o objeto de interesse tem auxílio de um observador. O observador apenas recebe as notificações do objeto de interesse e repassa aos assinantes interessados. No ultimo cenário, o objeto de interesse não tem envolvimento com o serviço de evento. Por este motivo, um observador faz a monitoração através de consultas ao serviço de evento para notificar assinantes quando um evento ocorrer.

Embora as notificações possam ser encaminhadas diretamente do objeto de interesse ao assinante, a complexidade do processamento de notificações pode ser dividida entre os processos observadores (COULOURIS, 2007). Os observadores podem desempenhar funções de filtragem de notificações (reduzir o número de notificações recebidas pelos assinantes), padrões de eventos (relacionamento entre vários eventos que o assinante está interessado) e caixas de correio de notificação (notificações podem ser suspensas até que assinante esteja apto a receber).

## 3.8 Considerações Finais

O Capítulo discutiu o paradigma de comunicação distribuída de invocação a métodos remotos e a sua adaptação para o suporte aos sistemas baseados em *publish-subscriber*. Embora os modelos caracterizem objetos como independentes, estes possuem estrutura arquitetural *request-reply* para realizar a invocação de seus métodos remotamente. Esta modelagem permite que seja viável a comunicação entre entidades cliente e servidor refletindo em interoperabilidade, portabilidade e flexibilidade das aplicações distribuídas.

# 4. *WIRELESS SHARING ENVIROMENT – OPERATING SYSTEMS*

## 4.1 Considerações Iniciais

Neste Capítulo será apresentada a proposta de trabalho para o WSE-OS (*Wireless Sharing Environment – Operating Systems*) bem como sua arquitetura e fluxo de execução, além de uma breve contextualização de gerenciamento centralizado de computadores ilustrando características e limitações dos conceitos atuais. Este Capítulo também apresenta os objetivos específicos nos quais este trabalho se propõe a desenvolver, pormenorizando as camadas necessárias para o desenvolvimento do *middleware* de comunicação sem fio para gerenciamento centralizado de computadores em rede.

## 4.2 Gerenciamento Centralizado de Computadores

Grande parte dos sistemas de redes de comunicação adota uma linha de estruturação bastante simples para o gerenciamento destes computadores. Um computador central principal, denominado gerente, é responsável por reunir informações e realizar tarefas de controle do sistema, onde os demais computadores, os agentes, coletam informações e executam as ordens. Desta forma, todas as políticas e as atividades de administração são realizadas em um único ponto para a conseqüente reflexão nos demais nós gerenciados da rede (CHERVENAK, et AL. 2000). Na Figura 17 é ilustrada a atuação de um servidor e diversos clientes conectados a ele por diferentes meios de comunicação.

O gerente servidor de rede deve dispor, integrar e coordenar elementos de hardware e software para monitorar, testar, consultar, configurar, analisar, avaliar e controlar os recursos da rede (SAYDAM, 1996) de modo que satisfaça exigências operacionais de desempenho e qualidade a um custo razoável. De maneira resumida, o gerenciamento centralizado de computadores busca pela automação de configurações repetitivas de software de modo que possa ser replicados para os clientes. Segundo este modelo centralizado, pode-se considerar duas vertentes de soluções: (1) Configuração Centralizada e (2) Execução Centralizada.

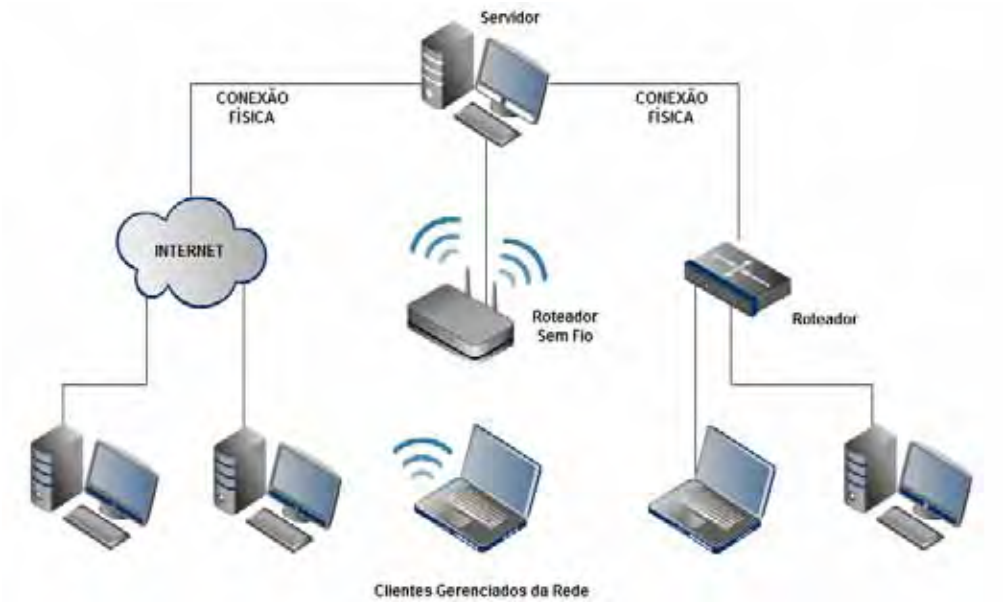


**Figure 17 - Arquitetura de rede de computadores com um gerente servidor e clientes gerenciados**

A Configuração Centralizada é baseada no conceito de código móvel, onde o gerenciamento pode ser focado tanto para aplicativos específicos quanto para sistemas operacionais. No contexto de aplicativo pode-se citar o recurso de *applet* na linguagem Javascript (COULOURIS, 2007). Este *applet* é carregado pelo cliente, cujo código está armazenado no servidor, e posteriormente executado. Outro exemplo é a utilização de Imagem de Sistema Única (ISU), onde um arquivo binário representando um sistema operacional com configurações pré-estabelecidas deve ser replicado nos clientes da rede (ZHOU, 2006).

Uma vantagem da configuração centralizada baseada no conceito de código móvel é a boa resposta interativa, pois a execução local não sofre atrasos, nem variação de banda associada à comunicação na rede (COULOURIS, 2007). Por outro lado, o uso de código móvel, além de ser uma ameaça potencial à segurança do computador destino, este sofre limitações imposta pelo forte acoplamento entre *hardware* e *software* das estações (CRUZ, 2008). Desta forma, o gerenciamento de computadores com diferentes características físicas se torna inviável através da padronização de apenas uma ISU, tornando-se uma tarefa individual e desqualificando o gerenciamento de forma centralizada(JONES, 2007) (GRIMM, 2001).

A vertente de Execução Centralizada é baseada no conceito TCSC (*Thin Client Server Computing).* O termo cliente “leve” se refere a uma camada de software no cliente gerenciado no qual permite

execução de programas em um servidor de computação remoto. Normalmente, o computador servidor possui vários processadores ou é estruturado em *cluster* executando uma versão para multiprocessadores de um sistema operacional, como o UNIX ou o Microsoft Windows (COULOURIS, 2007). A utilização de modelos leves (hardware e software) proporciona maior facilidade e centralização do gerenciamento, diminuindo o custo de administração, suporte e manutenção. Ao mesmo tempo, o *hardware* servidor, responsável por hospedar todo o processamento dos clientes, pode ser o principal gargalo operacional em relação ao número de clientes conectados.

Algumas soluções de gerenciamento com execução centralizada utilizam uma infra-estrutura com mecanismos de virtualização e acesso remoto, conhecidas como *Virtual Desktop Infrastructure* (VDI). Como por exemplo, o *VMware Virtual Desktop Infrastructure* (VMWARE, 2007) que cria uma infra-estrutura de *desktops* virtualizados onde a execução e gerenciamento de dados ocorre em um único servidor. Os protocolos de conexão para transmissão das mensagens podem ser baseados, por exemplo, em RDP (*Remote Desktop Protocol*) ou VNC (*Virtual Networking Computing*). Desta forma, terminais acessam *desktops* virtuais através dos servidores de autenticação e alocação de máquinas virtuais nos quais estão sendo gerenciados e configurados remotamente.

## 4.3 Proposta WSE-OS

O WSE-OS é um ambiente de compartilhamento sem fio para Sistemas Operacionais (SOs) que tem por objetivo centralizar o gerenciamento de distribuição de dados dos clientes em um servidor multiprocessado, oferecendo a coordenação de instanciações de SOs através de invocações remotas por um canal sem fio entre cliente e servidor.

O sistema integra e sincroniza componentes de *hardware* e *software* de clientes de modo a controlar os recursos de rede. Baseado no modelo arquitetural cliente-servidor, o WSE-OS é estruturado no conceito de sistemas TCSC (*Thin Client Server Computing)* e utiliza a comunicação entre objetos distribuídos para oferecer um ambiente de execução completo para os usuários do sistema. A Figura 18 ilustra a arquitetura inovadora do sistema WSE-OS sem precedentes publicados para esta forma de gerenciamento (CREPALDI, 2011).

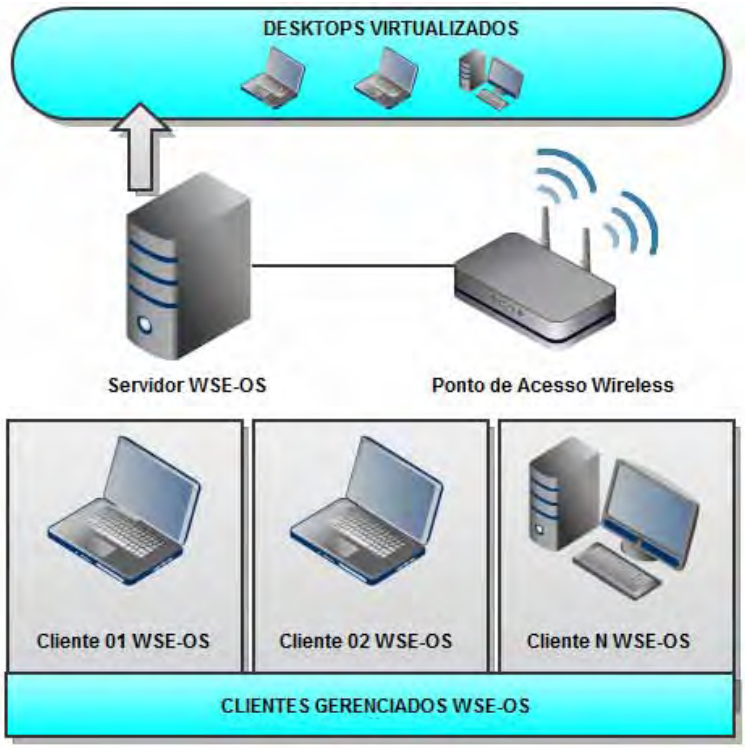


**Figura 18 - Arquitetura WSE-OS**

O sistema é baseado na construção de uma infra-estrutura virtual de *desktops* (VDI), na qual os clientes acessam remotamente máquinas virtuais que mantém instâncias de execução de sistemas operacionais distintas e independentes para cada cliente gerenciado.

O projeto de arquitetura tem foco na flexibilidade e facilitação de conexão na qual associações entre os dispositivos cliente e servidor são rotineiramente criadas e destruídas. Ao contrário do que oferece as soluções atuais de gerenciamento centralizado de imagens, como o *Ardence* Desktop (ARDENCE, 2007), *Citrix Provisioning Server for Desktops* (CITRIX, 2008) e FlexLab (CRUZ, 2008), o WSE-OS utiliza transmissão de dados através de rede sem fio, propondo uma alternativa sem conexões físicas entre o servidor WSE-OS e os computadores clientes para o compartilhamento de imagens de sistemas operacionais.

Seguindo a mesma idéia de *boot* do projeto LTSP (*Linux Terminal Server Project*), onde uma distribuição Linux minimalista destinada a ser carregada pelos terminais que não possuem poder de processamento ou memória RAM suficiente para executar distribuições de SOs mais atuais com bom desempenho

(MORIMOTO, 2008), o WSE-OS herda características do modelo *thin client* onde há a necessidade de recursos mínimos de hardware no cliente para a representação de uma imagem de sistema operacional onde os processos locais e periféricos clientes são administrados por este núcleo recompilado.

Como tira proveito de toda execução centralizada provida pelo sistema de acesso remoto, processadores de alta freqüência, memórias de grande capacidade e periféricos tais como discos de armazenamento local, são abstraídos totalmente pelo *middleware* de comunicação refletindo diretamente na diminuição do consumo de energia, no baixo custo de hardware e da administração dos clientes WSE-OS.

Os clientes podem ser inicializados pela instanciação de um sistema operacional no servidor multi-processado que mantém instâncias de execução distinta e independente dos demais terminais monitorados através de uma camada de gerenciamento. O pedido de instanciação ocorre através do *middleware* WSE-OS de comunicação remota que deve estar em mídias de armazenamento secundário (*Flash Driver USB*) para que o *boot* ocorra por meio da porta USB (*Universal Serial Bus*) do computador cliente. Diante desta situação, o cliente torna-se independente de qualquer protocolo de *boot* remoto que exija conexões físicas com o servidor.

O pré-estabelecimento deste *middleware* nos terminais garante que toda a comunicação posterior será realizada por meio sem fio. Isto assegura que o processo de *boot* seja independente de protocolo como o PXE, que é um padrão aberto adotado pela indústria para permitir que clientes de uma rede corporativa possam carregar automaticamente, através da rede cabeada, imagens de *software* e configurações (HENRY, 2000).

Embora o projeto WSE-OS utilize o modelo cliente-servidor baseado em execução centralizada, a arquitetura possui alterações visando um melhor desempenho. Como exemplo destas alterações, o *middleware* WSE-OS possui em seu Módulo de Comunicação Cliente um Sistema de Transmissão Diferencial de Dados (STD) com configuração de *cache* em memória RAM e memória *Flash* (*Flash Driver USB*), além de um sistema de compressão otimizado para cada tipo de dado recebido (imagem, texto, vídeo e som).

O volume de dados armazenados em memória *cache* é reaproveitado em conexões futuras, refletindo na diminuição do tráfego de dados a partir da segunda conexão remota ou instanciação de aplicativos, aumentando o desempenho de inicialização.

O funcionamento do WSE-OS é orientado a comunicação por fluxo, uma vez que o SO disponível para o cliente, através do sistema de acesso remoto, se torna inviável sem conexão TCP. Embora arquivos de texto, componentes gráficos, arquivos de áudio e vídeo estejam em *cache* em memória RAM ou *Flash,* estes dados apenas poderão ser processados após resposta do Servidor WSE-OS pelo fato da arquitetura do sistema ser baseada em *request-reply* com suporte a eventos e notificações. A Figura 19 ilustra a arquitetura em camada do cliente e do servidor WSE-OS.

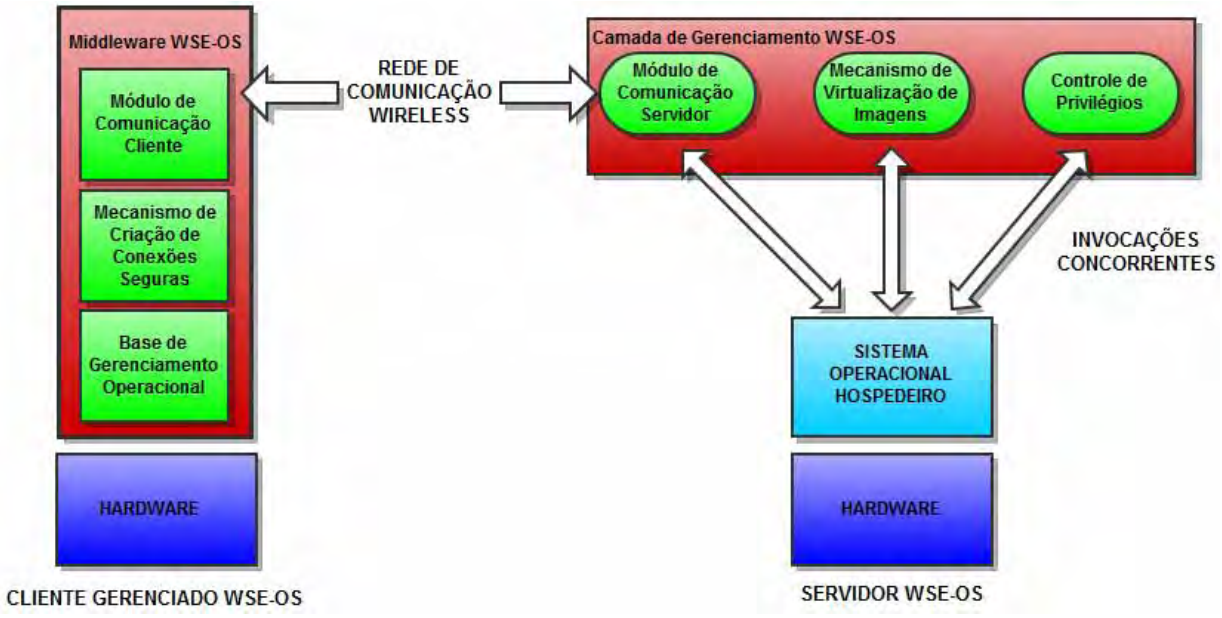


**Figura 19 - Arquitetura em Camada WSE-OS**

O ambiente WSE-OS apresenta simplificação de adicionar e remover dispositivos de maneira conveniente por ser estruturado com conexões sem fio e comunicação entre objetos distribuídos, além de não possuir acoplamento entre *hardware* e *software* de um computador como acontece nos casos de gerenciamento de configuração centralizada que não utilizam mecanismos de virtualização de imagens. A Figura 20 apresenta o fluxo de execução do sistema WSE-OS.

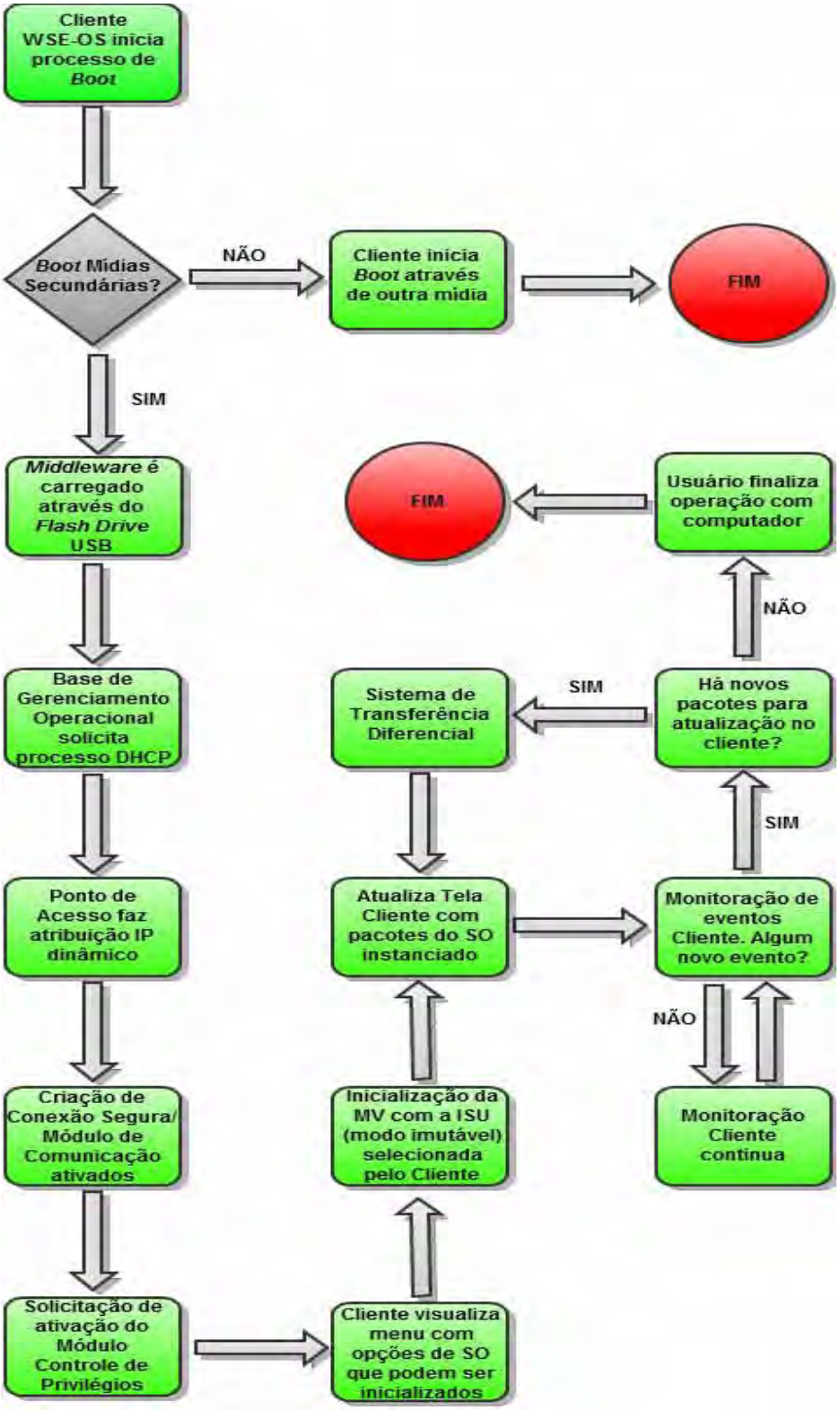


**Figura 20 - Fluxograma WSE-OS**

O processo de inicialização do sistema WSE-OS começa com o carregamento do *middleware* através de mídias secundárias (*Flash Driver* USB). Em seguida, o Módulo Base de Gerenciamento Operacional solicita processo de atribuição dinâmica de endereços IP à base de ponto de acesso central do ambiente. Neste momento, os Módulos de Mecanismo de Criação de Conexão Seguras e Comunicação Cliente são ativados para o estabelecimento do canal de comunicação com o servidor. Estabelecida essa comunicação cliente-servidor, é habilitada ao usuário do ambiente a escolha de sistemas operacionais disponíveis para execução de acordo com o seu nível de privilégios. A solicitação pela execução de algum desses SOs faz com que máquinas virtuais sejam inicializadas para, assim, fornecer um *desktop* virtual completo para o usuário remoto. Toda monitoração de eventos clientes é realizado com auxílio do Sistema de Transmissão Diferencial de Dados que, quando necessário, utiliza arquivos de *cache* em memória RAM e *Flash*.

## 4.4 Elementos Fundamentais

Esta seção caracteriza os elementos fundamentais que compõem o projeto de ambiente de compartilhamento sem fio para sistemas operacionais. O WSE-OS é composto por dois componentes principais: *Middleware* de Comunicação Remota Sem fio WSE-OS e Camada de Gerenciamento WSE-OS, além de uma rede de comunicação sem fios.

### *Middleware* de Comunicação Remota Sem Fio WSE-OS

O *middleware* de comunicação sem fio WSE-OS é um sistema localizado entre o hardware do cliente gerenciado e a Camada de Gerenciamento WSE-OS. Ele oferece a chamada de instanciação de um sistema operacional, além de todo gerenciamento operacional dos processos internos e comunicação do terminal cliente com o servidor. Este mediador é composto por três módulos dependentes e hierárquicos em sua construção para a transmissão e recebimento de dados.

Como ilustra a Figura 19, o Módulo Base Operacional é responsável pelo gerenciamento de recursos deste cliente, controlando processos, memória, sistema de arquivo e periféricos de forma a dar suporte à rede de comunicação sem fio, além de realizar solicitação ao ponto de acesso para inicialização do processo de atribuição dinâmica de endereços IP (*Internet Protocol*).

Seguindo a hierarquia ascendente do *middleware*, o Módulo Mecanismo de Criação de Conexões Seguras tem o objetivo de criar uma conexão segura com o servidor para que a execução de comandos na unidade remota não tenha interceptações e roubo da informação confidencial. Desta maneira, a troca de dados é realizada através de um canal seguro garantindo confidencialidade e integridade dos dados no ambiente WSE-OS.

Em relação ao Módulo de Comunicação Cliente, este tem por objetivo fornecer um ambiente de trabalho gráfico remoto para o terminal cliente. Para que isto seja possível, este módulo cliente trabalha associado ao Módulo de Comunicação do Servidor na Camada de Gerenciamento WSE-OS.

Através da sincronicidade entre os módulos distribuídos, o processamento pode ser centralizado no servidor. Diante desta estruturação, o Módulo de Comunicação Cliente repassa eventos/dados para que o processamento seja realizado pelo servidor e, por conseqüência, o Módulo de Comunicação Servidor transmite apenas as instruções e comandos correspondentes ao evento disparado para a atualização da interface gráfica cliente.

**Camada de Gerenciamento WSE-OS**

A Camada de Gerenciamento do WSE-OS consiste em um sistema localizado no servidor WSE-OS que reúne ferramentas de virtualização, acesso remoto e de ambiente de trabalho remoto, visando oferecer uma solução para que um cliente remoto seja capaz de interagir com uma instanciação de um SO.Esta camada é composta por três módulos (Módulo de Comunicação Servidor, Módulo de Mecanismo de Virtualização de Imagens e Módulo de Controle de Privilégios) independentes funcionalmente entre si, mas que quando executados simultaneamente são capazes de configurar o servidor para que este seja capaz de fornecer acesso a um SO virtualizado.

Como ilustra a Figura 19, o Módulo de Comunicação Servidor tem o objetivo de aceitar conexões seguras por fluxo de dados com os diversos clientes do ambiente e transmitir dados referentes aos eventos solicitados para atualização de interface dos terminais. O Módulo de Mecanismo de Virtualização de

Imagens é o responsável por realizar a virtualização de imagens. Este módulo mantém instância de um sistema operacional (SO Convidado) para cada usuário do ambiente. A máquina virtual é executada sobre o Sistema Operacional Hospedeiro (SOH) do servidor.

Por fim, o Módulo de Controle de Privilégios consiste em um *script* de controle que entra em execução imediatamente no momento em que o usuário realiza o *login* no SOH do servidor. É a partir das instruções programadas neste *script* de controle que há um gerenciamento de quais imagens de SO o servidor possui para cada cliente registrado do ambiente.

**Rede de Comunicação Sem Fio**

O ambiente de compartilhamento de dados entre servidor e clientes é baseado no tipo WLAN (*Wireless Local Area Network*) modalidade infra-estrutura, onde os dispositivos sem fio clientes comunicam-se diretamente com a base de ponto de acesso central (BPAC) . Ao contrário do enlace estabelecido com os clientes do ambiente, o servidor está conectado através de conexões físicas com a BPAC.

O padrão estabelecido para a rede de comunicação sem fio é o IEEE 802.11n (PARDALOS, 2009) com taxa máxima de transferência de 300Mbps (faixa de freqüência 2,4Ghz). Este padrão de transmissão, se comparado aos padrões anteriores IEEE (*Institute of Electrical and Electronics Engineers*) para transmissão sem fio (802.11b, 802.11g), tem um alcance de sinal aproximadamente duas vezes maior e um nível maior de confiabilidade. Para contornar qualquer problema de segurança, o padrão estabelecido utiliza protocolo de segurança WPA2-PSK (NARAYAN, 2010) para autenticação da interface da placa cliente e inicialização do processo DHCP [RFC 2132] (*Dynamic Host Configuration Protocol* ) para a concessão de endereços IP para clientes da rede.

Com esta configuração de rede, o padrão 802.11n associado com mecanismos de autenticação WPA2-PSK é capaz de oferecer um meio de comunicação com elevadas taxas de transmissão, grande alcance de sinal (aproximadamente 70 metros para ambientes internos e 250 metros para ambientes externos) e confidencialidade assegurada.

## 4.5 *Middleware* WSE-OS

Como já descrito no Capítulo 1, seção 1.3, o projeto WSE-OS possui duas linhas de pesquisa e implementação. A primeira está focada na estruturação do *middleware* de comunicação remota, um mediador que deve estar presente em todos os clientes WSE-OS para a troca de dados com o servidor de *desktops.* A segunda linha tem por objetivo a elaboração da Camada de Gerenciamento WSE-OS. Esta camada estará situada no servidor do sistema e será responsável pela coordenação de instanciações de sistemas operacionais para os clientes.

O *middleware* que este trabalho se propõe a desenvolver permite que invocações de instanciação de *desktops* virtualizados ao servidor WSE-OS sejam válidas. O *middleware* WSE-OS é baseado na comunicação entre objetos distribuídos realizada por meio de invocações a métodos remotos (RMI – *Remote Method Invocation*), além de possuir mecanismos de criação de canal de comunicação seguro entre processos cliente e servidor. No entanto, esta estruturação apenas se torna possível com o Módulo Base de Gerenciamento Operacional. A Figura 21 ilustra as camadas necessárias do *middleware* WSE-OS.

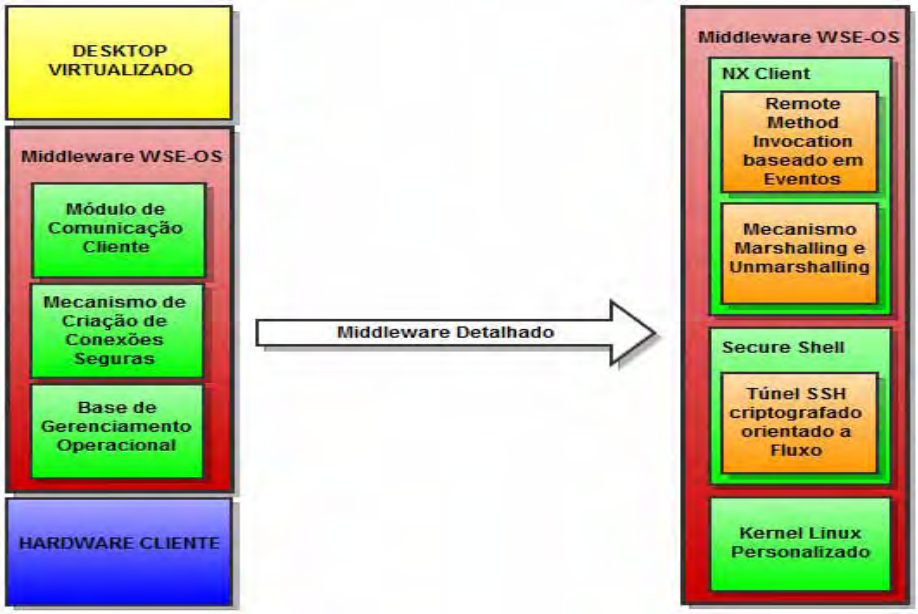


**Figura 21 - Arquitetura detalhada do Middleware WSE-OS**

Como apresentado na Figura 21, o Módulo de Gerenciamento Operacional está codificado com base em um *kernel* Linux personalizado. Ou seja, o gerenciamento de recursos do cliente é realizado por um núcleo Linux recompilado de modo que ofereça controle de processos e periféricos para o suporte à rede de comunicação sem fio. Esta base de gerenciamento, como será visto na seção 5.2.1, está configurada de maneira que garanta desempenho em seu carregamento e aceite clientes com diferentes plataformas de *hardware* para o reconhecimento de *drivers*.

O Módulo de Mecanismo de Criação de Conexões Seguras utiliza o protocolo de acesso remoto *OpenSSH* (Open *Secure Shell* versão 1.2.12)(YLONEN, 2006a) para a criação de conexões seguras. O SSH permite criar conexões com outros computadores garantindo confidencialidade e integridade dos dados através de técnicas de criptografia e túnel orientado a fluxo TCP. O SSH atende todas as necessidades para garantir confidencialidade e integridade nas transmissões vulneráveis a interceptação, sendo considerado um protocolo padrão de transmissão segura com publicação pela IETF (*Internet Engeneering Task Force*). Pelo fato do projeto WSE-OS estar estruturado por uma rede sujeita a interceptação e perda de dados, a utilização deste sistema de comunicação garante o encapsulamento das mensagens de comunicação cliente e servidor fornecendo transmissão segura.

O Módulo de Comunicação Cliente possui mecanismo para a realização da comunicação RMI baseado em eventos entre os objetos distribuídos cliente-servidor. Neste contexto, o módulo utiliza um cliente de interface gráfica (NX*Client* versão 3.4.0)(REGIS, 2007) que possui estrutura para o empacotamento e desempacotamento de modo a dar suporte para orientação a objetos distribuídos (ver seções 3.1/3.3.1). O Módulo de Comunicação Cliente será pormenorizado no Capítulo 5.

A proposta de modularização do *middleware* torna viável a possibilidade de, em um trabalho futuro, efetuar modificações em qualquer um dos módulos que o compõe. Isto faz com que a adaptação às novas necessidades tecnológicas ou diferentes ambientes de execução seja realizada de modo prático sem afetar o sistema mediador por inteiro. As seções do Capítulo 5 pormenorizarão a estruturação e o fluxo de execução de cada um dos módulos do *middleware* WSE-OS.

## 4.6 Considerações Finais

A evolução das redes locais sem fio e dos mecanismos de virtualização de imagens de sistemas operacionais, associado aos ambientes de trabalho gráfico remoto, criam novas perspectivas para que o gerenciamento centralizado de computadores seja estruturado por conexões sem fios e sem a necessidade do uso de sistemas de arquivos distribuídos.

Ao contrário dos atuais trabalhos envolvendo máquinas virtuais, onde o propósito de criar um ambiente utilitário de computação distribuída com configuração centralizada é a maior motivação do uso da virtualização, a utilização de uma infraestrutura virtual de *desktops*, como no projeto *VMware Virtual Desktop Infrastructure,* que apesar de ser uma solução para gerenciamento com execução TCSC não utiliza uma rede de comunicação sem fio, mostra que tecnologias baseadas em VDI estão maduras para este tipo de solução para o gerenciamento de computadores.

# 5. ARQUITETURA *MIDDLEWARE* WSE-OS

## 5.1 Considerações Iniciais

Neste Capítulo serão apresentadas detalhadamente as camadas da arquitetura do *middleware* WSE-OS e sua dinâmica de funcionamento em plataformas clientes. Como ilustrado pelo capítulo anterior, a descrição das camadas seguirá pela ordem hierárquica da modularização. Por fim, o capítulo apresenta as considerações finais levando em consideração características de execução deste mediador.

## 5.2 Camadas Mediadoras

O *middleware* WSE-OS é um sistema localizado nos clientes (*Flash Driver* USB) do ambiente gerenciado. O *middleware* WSE-OS oferece, além da chamada de instanciação de um sistema operacional, todo o gerenciamento operacional dos processos internos. Este *middleware* de comunicação sem fio está dividido em três grandes módulos hierárquicos e dependentes entre si (Módulo Base de Gerenciamento Operacional, Módulo Mecanismo de Criação de Conexões Seguras e Módulo de Comunicação Cliente). Diante desta estruturação, a metodologia utilizada para o seu desenvolvimento segue três linhas de pesquisas e testes associados a cada módulo. As seções a seguir mostrarão, de modo pormenorizado, toda arquitetura individual dos módulos.

### 5.2.1 Base de Gerenciamento Operacional

A Base de Gerenciamento Operacional é o módulo responsável por controlar os processos, sistema de arquivo, memória e periféricos dos clientes WSE-OS. Esta base operacional fornece o suporte para os módulos superiores (Módulo Mecanismo de Criação de Conexões Seguras e Módulo de Comunicação Cliente) estabelecerem a comunicação na rede sem fio. A engenharia necessária para a construção deste módulo é norteada pela eficiência do carregamento de um núcleo Linux minimalista associado a uma

versão recompilada de um Sistema Operacional Hospedeiro e aos arquivos de configuração da interface cliente WSE-OS. A Figura 22 ilustra o módulo detalhadamente.

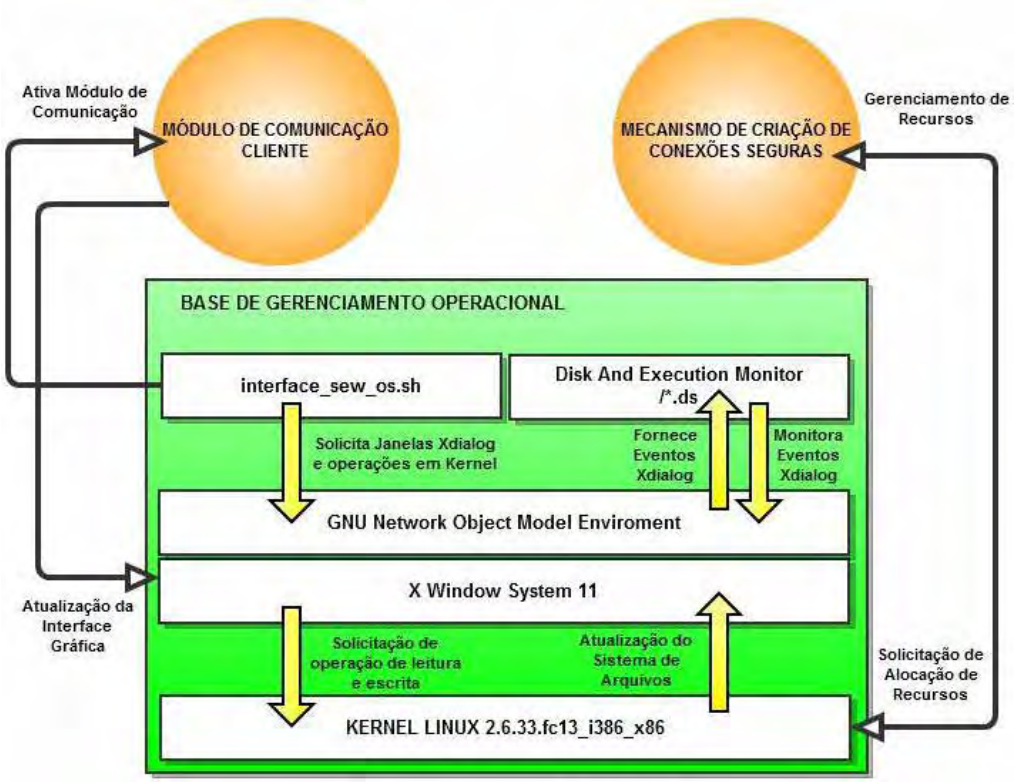


**Figura 22 - Base de Gerenciamento Operacional**

O Módulo Base de Gerenciamento Operacional é composto por um núcleo Linux versão 2.6.33 modelado para o sistema operacional hospedeiro (SOH), constituído pelo Fedora Core 13 (FC13) para arquiteturas x86 (32 bits). Neste caso, o sistema operacional hospedeiro FC13 está recompilado de modo a oferecer apenas os recursos e serviços essenciais para o funcionamento dos outros módulos (seção 5.2.2/5.2.3).

Embora o *kernel* Linux versão 2.6.33 e o SOH FC13 estejam em versões minimalistas para atender aos requisitos de eficiência no carregamento do sistema WSE-OS, ambos estão aptos a atender requisitos de diferentes plataformas de hardware clientes em que são executados. O desempenho de execução em processadores clientes de alta freqüência demonstra pequena diferença em relação aos processadores de baixa freqüência. Estes dados serão discutidos com maiores detalhes na seção 6.3.

O SOH FC13 é reduzido para que o conjunto de administração de janelas (X *Window System* 11 associado com o GNU *Network Object Model Enviroment*) possa ser executado pelo cliente WSE-OS sem que seja copiado por completo para a memória RAM, resguardando este recurso apenas para a configuração de *cache* na transmissão/recebimento dos dados com o servidor como será mostrado na seção 5.3.3.

Para que a comunicação baseada em TCSC (*Thin Client Server Computing*) possa ser inicializada, o módulo de gerenciamento possui um *script* (interface_sew_os.sh) (Anexo B) que implementa uma interface (Figura 23) para que o cliente faça configurações das duas outras camadas mediadoras. Com isto, o cliente WSE-OS pode editar configurações de rede, imagem e alocação em memória *cache* RAM (memória primária) e em *Flash Driver* USB (memória secundária). Esta interface de usuário é codificada em *Shell Script* com chamadas *XDialog* para construção de janelas baseadas no X *Window System 11 GNU Network Object Model Enviroment* do SOH FC13.

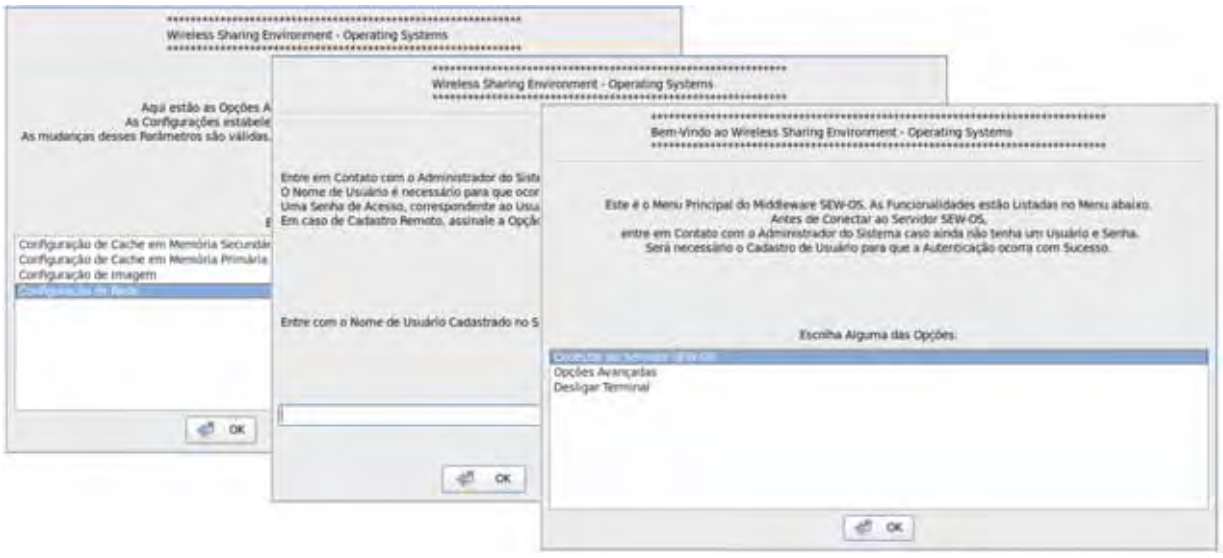


**Figura 23 - Interface WSE-OS (interface_sew_os.sh)**

Associado a este *script*, existem processos *daemons* (*.ds) que monitoram ações do usuários e ,quando necessário, impede que estes acessem partes do *middleware* não liberado. Esta é uma forma de assegurar que o cliente não tenha acesso direto às camadas mediadoras a não ser pela interface

principal do cliente nesta base de gerenciamento. De forma sucinta, o Módulo Base de Gerenciamento Operacional do *middleware* WSE-OS fornece gerenciamento de recursos e o suporte à rede de comunicação sem fio, além de ser arquitetado de modo a garantir interoperabilidade de execução em diversas plataformas de hardware e variações de periféricos, principalmente aqueles associados à placa de conexão em redes sem fio. Uma lista detalhada de placas de rede suportadas pelo *middleware* está no Anexo A desta dissertação.

### 5.2.2 Mecanismo de Criação de Conexões Seguras

O Mecanismo de Criação de Conexões Seguras é o módulo do *middleware* WSE-OS que sustenta a comunicação com o servidor remoto. Ou seja, toda transmissão e/ou recebimento de dados com o servidor é realizado por este módulo. A Figura 24 ilustra sua estrutura de funcionamento.

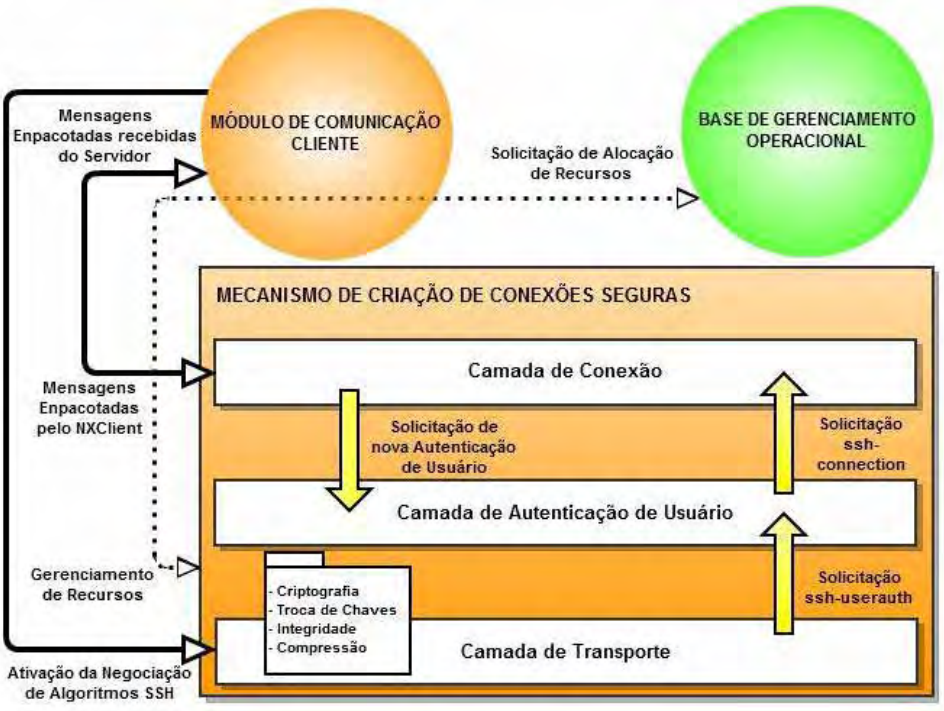


**Figura 24 - Mecanismo de Criação de Conexões Seguras**

O Módulo de Mecanismo de Criação de Conexões Seguras é baseado no sistema de acesso remoto *OpenSSH* (*Open Secure Shell* versão 1.2.12)(YLONEN,2006a). O *OpenSSH* trabalha com camadas distintas de execução utilizando criptografia de chave pública para autenticação do servidor (YLONEN 2006a).

A camada base do Módulo de Mecanismo de Criação de Conexões Seguras (camada de transporte) fornece serviço de autenticação de servidor, negociações de algoritmos de confidencialidade, integridade e compressão de dados entre cliente e servidor WSE-OS. Detalhes das escolhas destes algoritmos serão vistos nas seções 5.3.2/6.2.1. Por padrão, esta camada está configurada para rodar sobre conexões TCP/IP, mas não há restrição quanto ao uso de outros fluxos de dados confiáveis.

A autenticação de usuário é habilitada logo após toda negociação inicial dos algoritmos da camada de transporte (YLONEN 2006b). O método de autenticação de clientes utilizado no *middleware* é baseado em nome de usuário e senha. Desta forma, cabe ao servidor WSE-OS interpretar esta correspondência e validar o usuário em seu banco de dados.

Após toda negociação dos algoritmos e a autenticação de usuário e servidor WSE-OS, o módulo ativa a sua última camada de trabalho. A Camada de Conexão utiliza o túnel criptografado estabelecido pelas camadas inferiores para criar um canal de programa gráfico remoto. Desta forma, toda transmissão e/ou recebimento dos eventos/dados é realizado pela camada de conexão através do serviço *Secure Channel Protocol* (SCP) (YLONEN, 2006d) do *OpenSSH* versão 1.2.12.

Para que o ambiente WSE-OS tenha o suporte às mídias removíveis dos terminais clientes remotos é necessário o estabelecimento de um segundo canal dentro do mesmo túnel criptografado. Este segundo canal apenas é utilizado quando uma mídia removível é conectada no cliente. Como os clientes possuem seus SOs em execução em um servidor remoto, os eventos disparados pelo *middleware* são processados neste servidor multiprocessado. Portanto, é a partir de um evento de sinalização cliente que ocorre a montagem dos dispositivos clientes no sistema de arquivo do SO virtualizado.

Para que este evento de sinalização e solicitação de montagem de dispositivos seja interpretado corretamente pelo servidor WSE-OS, o Módulo de Comunicação Servidor possui uma camada específica baseada no SSHFS (*Secure Shell FileSystem*) (SZEREDI, 2010). O SSHFS faz acesso às mídias removíveis dos clientes e a montagem no SO virtualizado correspondente. Desta forma, os terminais clientes têm acesso ao conteúdo desta mídia local através da montagem no sistema de arquivo virtualizado do SO em execução.

Através do serviço SFTP (*Secure File Transfer Protocol*) (YLONEN, 2006d) do Módulo de Mecanismo de Criação de Conexões Seguras do *middleware* WSE-OS, os dados relativos ao sistema de arquivo das mídias removíveis clientes são transmitidas neste segundo canal de forma segura ao terminal destino. O serviço SFTP criptografado garante proteção contra potenciais *sniffers* de rede, evitando possíveis extrações de informações dos pacotes transmitidos na rede sem fio WSE-OS.

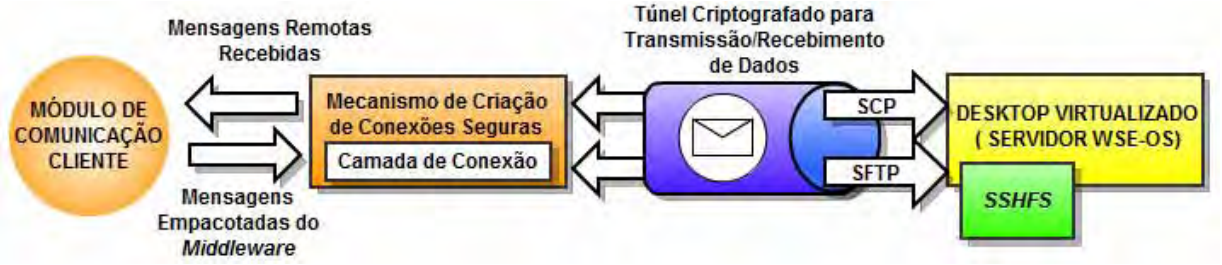


**Figura 25 - Esquema Simplificado de Conexão Cliente-Servidor**

Neste caso, a Camada de Conexão recebe e encaminha mensagens do túnel criptografado SSH ao Módulo de Comunicação Cliente do *middleware* WSE-OS. Os eventos disparados pelo terminal cliente seguem a mesma idéia em fluxo contrário até serem processadas na Camada de Gerenciamento WSE-OS no servidor. O Módulo de Mecanismo de Criação de Conexão Segura é a ponte de ligação propriamente dita entre o *desktop* virtualizado no servidor e o cliente WSE-OS. Não diferente dos outros módulos do *middleware*, este mecanismo de criação de conexões garante a segurança de transmissão nesta rede aberta sem fio ao mesmo tempo em que assegura a eficiência na troca de dados.

### 5.2.3 Módulo de Comunicação Cliente

O módulo de Comunicação Cliente é a camada do *middleware* WSE-OS que contém os mecanismos necessários para a comunicação entre objetos distribuídos realizados por meio da invocação a métodos remotos baseado em eventos. Por seguir esta idéia, o módulo está estruturado por um sistema gráfico remoto (NXClient versão 3.4.0)(REGIS, 2007) que possui camada responsável pelo empacotamento e desempacotamento de objetos distribuídos (ver Capítulo 3). A Figura 26 detalha esta arquitetura.

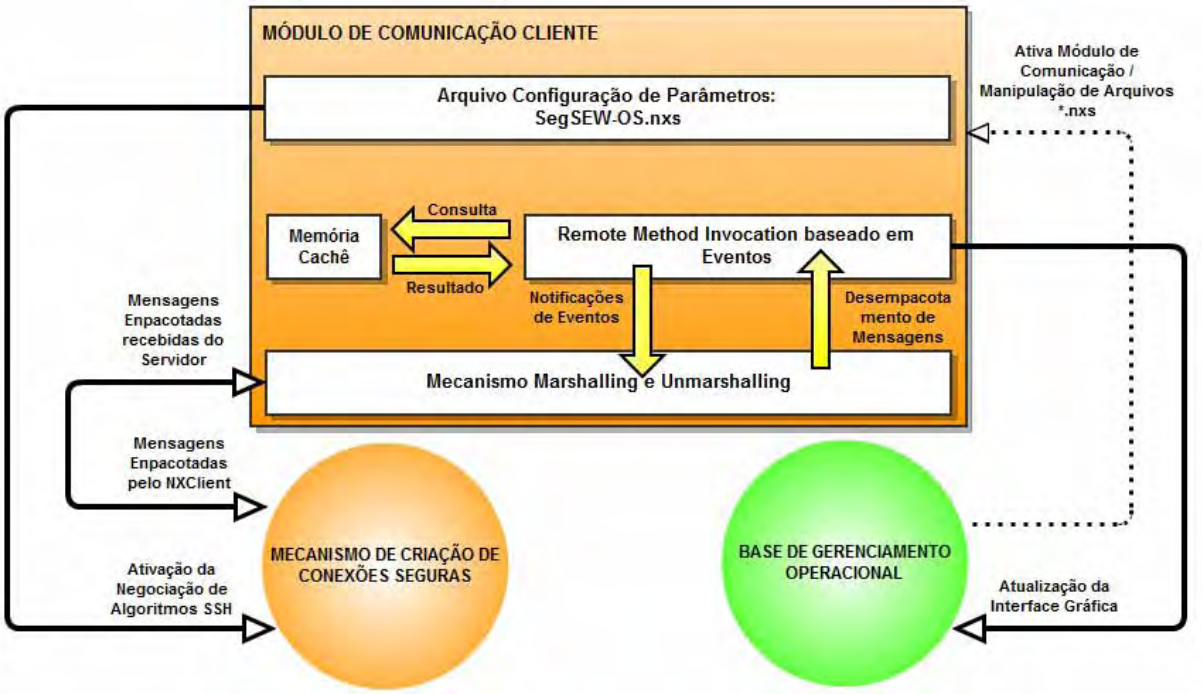


**Figura 26 - Módulo de Comunicação Cliente**

Basicamente, este sistema gráfico remoto permite que clientes acessem remotamente servidores Linux/Unix. No caso do projeto WSE-OS, este módulo tem um objetivo melhor definido. O NXClient está modelado para instanciar uma imagem de um sistema operacional no servidor WSE-OS. Isto faz com que o cliente seja denominado *thin-client*, ou seja, a partir da execução deste módulo cria-se a comunicação baseada em TCSC em modo gráfico.

Todas essas configurações e outras como, por exemplo, de rede, imagem e *cache*, estão centralizadas em um arquivo (SegSEW-OS.nxs) (Anexo C) que aguarda a entrada dos parâmetros da interface cliente do Módulo Base de Gerenciamento Operacional. Além disto, o arquivo deste módulo dispara a ativação de toda a negociação dos algoritmos do Módulo de Mecanismo de Criação de Conexão Segura.

Como a Figura 26 ilustra, toda a ação do cliente é monitorada pela camada RMI baseado em *publish-subscriber* na qual repassa estes eventos para serem empacotados e, depois, direcionados para a camada de conexão do Módulo de Mecanismo de Criação de Conexões Seguras. O mecanismo de empacotamento e desempacotamento deste módulo possui um Sistema de Transmissão Diferencial de Dados (STD), fazendo com que outras soluções para a comunicação gráfica remota sejam desconsideradas para utilização no projeto por não possuírem mecanismos de otimização de transmissão e recebimento de dados como, por exemplo, o RealVNC e TightVNC (DIGIERE, 2011).

O NXClient é um sistema gratuito, open-source e possui compatibilidade com o protocolo *OpenSSH* para a encriptação de dados. Por isso, segue o mesmo princípio das escolhas das tecnologias anteriores, tais como *OpenSSH,* SOH FC13, núcleo Linux e *scripts* codificados em *Xdialog*. Detalhes do funcionamento do *cache* e do Sistema de Transmissão Diferencial de Dados serão vistos nas seções 5.3.3 e 6.4.3.

## 5.3 Funcionamento do *Middleware* WSE-OS

De acordo com os módulos detalhados pelas seções 5.2.1, 5.2.2 e 5.2.3, as seções 5.3.2, 5.3.3 e 5.3.4 mostrarão características de configuração e dinâmica de execução do *middleware* WSE-OS. Todos os experimentos realizados com o sistema mediador serão descritos no Capítulo 6.

### 5.3.1 Caracterização Geral

A proposta de trabalho do WSE-OS define uma série de requisitos que o *middleware* deve oferecer para que possa ser aplicado como um mediador no gerenciamento de distribuição de dados em um ambiente de compartilhamento sem fio para sistemas operacionais. Assim, cabe ao *middleware*:

- Fornecer interoperabilidade de execução em diversas plataformas de hardware e variações de periféricos, principalmente aqueles associados a placas de conexão em redes sem fio (Anexo A);
- Gerenciar recursos (controle de processos, memória, sistema de arquivo e periféricos) de computadores com processador baseado na arquitetura x86;
- Fornecer solicitação de autenticação de interface de rede cliente e processo DHCP à rede de comunicação sem fio do ambiente WSE-OS;
- Fornecer uma interface com o usuário do sistema para realização de *login*/*logout* no ambiente;
- Fornecer um sistema gráfico remoto para a troca de informações com o servidor WSE-OS;
- Fornecer integridade e confidencialidade na comunicação remota;
- Fornecer um Sistema de Transmissão Diferencial de Dados para comunicação TCSC;
- Permitir configuração de rede (número IP servidor e porta de acesso);
- Permitir configuração de *caches* em memória RAM e *Flash Driver* USB;
- Permitir configuração de compactação de imagens;
- Permitir a utilização de Sistemas Operacionais sem a necessidade de protocolos de *boot* remoto;
- Permitir execução do sistema a partir de mídia secundária com *boot* pela porta USB.

A Figura 27 ilustra o funcionamento de execução simplificado do *middleware* WSE-OS.

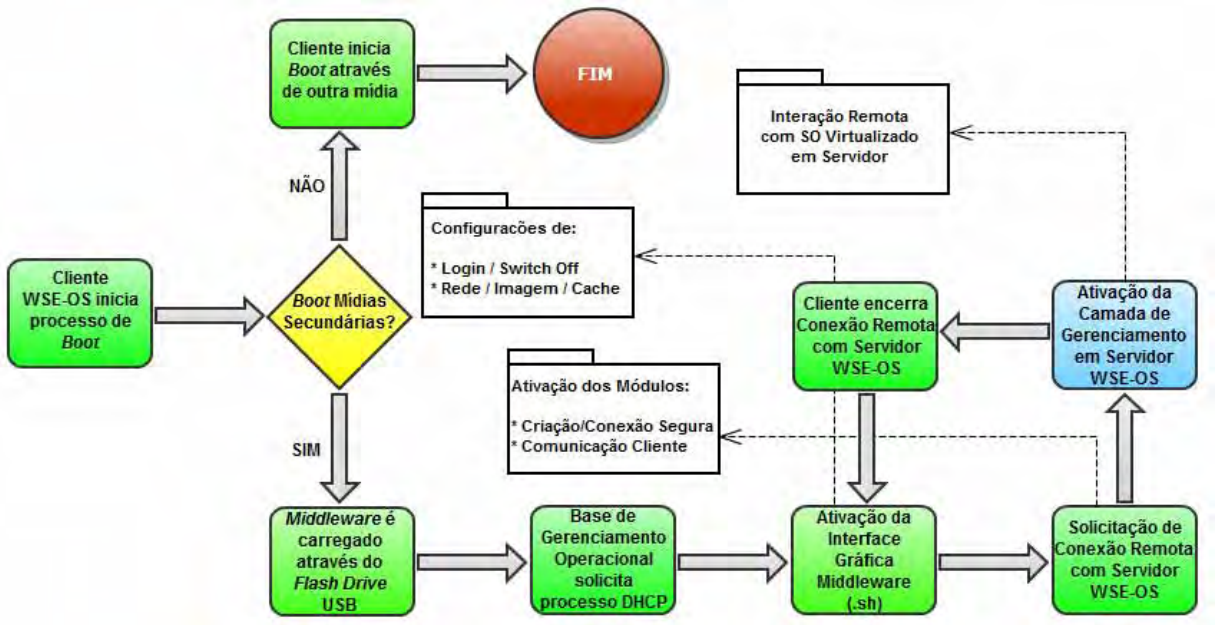


**Figure 27 - Fluxo Simplificado de Execução do Middleware WSE-OS**

Assim que o usuário inicia o processo de *boot* através de mídias secundárias, o *middleware* WSE-OS é carregado através do *Flash Driver* USB. Em seguida, o Módulo Base de Gerenciamento Operacional solicita processo DHCP à BPAC simultaneamente com a ativação da interface gráfica do *middleware.* Nesse momento, o usuário pode realizar configurações de rede, imagem, *cache*, além de desligar o terminal ou realizar o *login* no ambiente. Caso seja efetuada esta última operação, ocorre a solicitação de conexão remota com o Servidor WSE-OS e ativação dos Módulos de Criação da Conexão e Comunicação Cliente do *middleware.*

Estabelecida a conexão remota cliente-servidor, a Camada de Gerenciamento WSE-OS é ativada no servidor fornecendo recursos necessários (DIGIERE, 2011) para a interação remota com o SO virtualizado. Assim que o usuário encerra a sessão com o SO virtualizado e desfaz a conexão por fluxo com o servidor, a interface gráfica do *middleware* é novamente ativada. Nesta situação, o usuário pode realizar um novo processo de *login* ou desligar o terminal em questão.

### 5.3.2 Negociação de Algoritmos e Conexão Remota

O Módulo de Mecanismo de Criação de Conexões Seguras do *middleware* WSE-OS mantém um arquivo de configuração de algoritmos de compressão, integridade e criptografia de dados, além de utilizar um algoritmo para a troca inicial de chaves. Como descrito anteriormente, este módulo é baseado no protocolo *OpenSSH* versão 1.2.12. Basicamente, o protocolo *OpenSSH* estabelece conexão com um *daemon* SSH que está na Camada de Gerenciamento no servidor WSE-OS (DIGIERE, 2011) para negociação dos algoritmos.

Toda a negociação inicial entre cliente e servidor ocorre na camada de transporte do Módulo Mecanismo de Criação de Conexões Seguras. Dois métodos de troca de chaves estão definidos no *middleware* para que toda a negociação posterior dos algoritmos ocorra de forma segura. O método *diffie-hellman-group1-sha1* [RFC2409] e *diffie-hellman-group14-sha14* [RFC3526] são utilizados para que cliente e servidor compartilhem a mesma PSK (*Pre-Shared Key*). Portanto, é através de PSK que todas as negociações dos algoritmos de segurança, integridade e compressão são realizadas. Logo na primeira conexão, após a negociação que estabelece uma PSK, o servidor WSE-OS envia sua chave pública ao

cliente junto com uma assinatura. Isto garante uma autenticação explícita do servidor WSE-OS, de modo que todas as mensagens posteriores criptografadas pelo cliente poderão apenas ser compreendidas pelo servidor WSE-OS. Esta técnica de verificação de servidores previne um tipo de ataque muito comum conhecida como *man in the middle* (MEYER, 2009), em que há uma substituição do servidor legítimo por outra máquina, utilizando o mesmo IP (YLONEN, 2006c). A figura 28 ilustra estas negociações de algoritmos do ambiente WSE-OS.

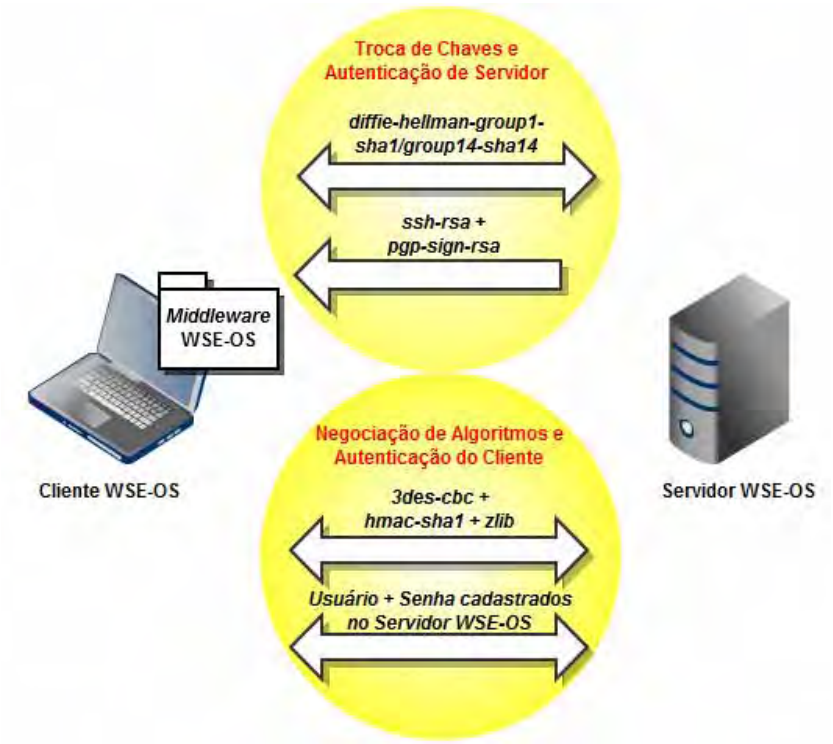


**Figura 28 - Negociação de Algoritmos e Autenticações**

Diante disto, neste primeiro cenário de negociação, o *middleware* utiliza as seguintes técnicas:

- Algoritmos *diffie-hellman-group1-sha1/group14-sha1* para o compartilhamento do PSK;
- Chaves *ssh-rsa/ssh-dss* de 512 bits para negociação dos próximos algoritmos;
- Assinaturas *pgp-sign-rsa/pgp-sign-dss* para a autenticação explícita do servidor WSE-OS.

Pelo fato do projeto WSE-OS estar estruturado por uma rede sujeita a interceptação e perda de dados, como de fato é uma rede sem fio, o *middleware* de comunicação está preparado para estes impasses aplicando criptografia e compressão, além de verificar a integridade dos dados na transmissão e recebimento de mensagens remotas. Como será detalhado no Capítulo 6, o módulo tem suporte a diversos algoritmos, mas as escolhas seguem o critério de eficiência na transmissão e recebimento destes dados.

Após a autenticação do servidor e por padrão estabelecido, o Módulo de Mecanismo de Criação de Conexões Seguras ativa o uso dos seguintes algoritmos:

- Algoritmo 3DES (*Triple Data Encryption Standard*) em modo CBC (*Cipher Block Chaining*) para a criptografia dos dados enviados;
- Algoritmo MAC *hmac-sha1* para computação de integridade dos dados;
- Algoritmo de compressão de dados ZLIB versão 1.2.1.

Com a negociação de todos os algoritmos estabelecida, a camada de transporte do Módulo de Mecanismo de Criação de Conexões Seguras faz solicitação (*ssh-userauth*) de serviço para a camada de autenticação de usuário. Associada a Camada de Gerenciamento WSE-OS no servidor, este serviço torna autêntico a conexão remota baseada na validação de usuário e senha cadastrados no servidor WSE-OS.

Diante desta validação, a conexão remota pode ser estabelecida e o sistema gráfico remoto envia e recebe dados dentro do canal SCP (*ssh-connection*) e mídias removíveis cliente dentro do canal SFTP (*ssh-connection*). Com todos os algoritmos em operação, a troca de mensagens é conduzida garantindo integridade e segurança nesta rede de conexão sem fio do ambiente WSE-OS.

### 5.3.3 Opções Avançadas de Configuração

O *middleware* WSE-OS de comunicação remota possui em sua interface (interface_sew_os.sh) com o usuário algumas opções de configurações avançadas. Todas as opções de configurações podem ser editadas conforme a necessidade do ambiente de trabalho no qual o *middleware* está inserido.

Uma dessas opções está diretamente relacionada ao módulo responsável pela criação de conexão remota. A edição de rede envolve, caso necessário, a configuração de IP e *Porta* do Servidor. Por padrão, o endereço IP do servidor para o acesso remoto cliente segue atribuição de acordo com o roteador sem fio do ambiente de testes. O número de porta 22 é o pré estabelecido para a conexão SSH entre cliente e servidor WSE-OS, visto que o protocolo SSH *Server* ouve pedidos de conexão nesta porta.

Outros parâmetros válidos para configuração de opções envolve o Módulo de Comunicação Cliente. Pelo fato deste módulo ser estruturado por um sistema gráfico remoto baseado na comunicação RMI *publish-subscriber* utilizando o *toolkit* X *Window System* 11 (Módulo Base de Gerenciamento Operacional), isto estabelece alguns limitantes na eficiência de transmissão e recebimento de dados.

Operações remotas que utilizam o X *Window System* apresentam o problema dos *roundtrips* (DIGIERE, 2011). Estes pacotes de respostas (*roundtrips*) são enviados a cada instrução recebida. De modo a garantir a integridade, o servidor X apenas transmite a próxima instrução caso receba a resposta da última enviada. Isto reflete negativamente na dinâmica da comunicação cliente-servidor.

Exatamente para contornar este problema, tanto o Módulo de Comunicação Cliente quanto o Módulo de Comunicação Servidor possuem um Sistema de Transmissão Diferencial de Dados (STD). Como ilustra a Figura 26, o módulo possui uma memória *cache* que trabalha sincronizada com outra memória *cache* idêntica no Módulo de Comunicação Servidor. Estes são responsáveis por armazenar dados já transferidos, reutilizando-os sempre que possível. Os benefícios do STD fazem com que o problema dos *roundtrips* seja contornado com sucesso e que ao invés da transmissão de pacotes onerosos à rede, sejam transmitidos pequenos pacotes com instruções de poucos *bytes* apenas com indicação para que o cliente use os dados em memória *cache*. Para que se tenha ótimo desempenho em redes com limitação de banda, o STD implementa um sistema de compressão otimizado para cada tipo de dado. Assim, o mecanismo de *marshalling* e *unmarshalling* trabalha seletivamente os dados de som, vídeo, imagem e texto utilizando os algoritmos *wav*(TSAI, 2001)*, mpeg*(LULIAN, 2009)*, jpeg*(RANI, 2009)*/png*(TSAI, 2010)*, zlib*(DEUTSH, 1996) respectivamente.

De acordo com os eventos disparados pelo *middleware*, o servidor WSE-OS envia instruções, textos e componentes gráficos para que sejam usados na montagem de tela do cliente WSE-OS. De forma sucinta, as opções avançadas de configurações do *middleware* permitem:

- Edição do número IP e *Porta* do Servidor WSE-OS;
- Edição do *Cache* em Memória RAM, estabelecendo escala de seleção entre 4, 8, 16, 32, 64 e 128 Mbytes da plataforma cliente;
- Edição do *Cache* em Memória *Flash Driver* USB, estabelecendo escala de seleção entre 0, 32, 64, 128, 512 Mbytes em Disco;
- Edição do tipo de compressão de imagens do SDT entre JPEG, PNG e sem compressão.

O volume de dados armazenados nos *caches* em memória RAM e *Flash Driver* USB no SDT pode ser reaproveitado em conexões futuras. Isto se reflete na diminuição de tráfego de dados a partir da segunda conexão. Desde que não ocorra um novo processo de *boot* do *middleware*, o *cache* em memória RAM será armazenado mesmo com a desconexão com o servidor WSE-OS. Dados mais específicos serão apresentados no Capítulo 6.

### 5.3.4 Fluxo de Execução

Todas as mensagens trocadas entre o Módulo de Mecanismo de Criação de Conexões Seguras e o servidor WSE-OS caracterizam o fluxo de execução externo onde ocorre a negociação dos algoritmos de criptografia, integridade, compressão, autenticações de usuário e servidor, além das mensagens originadas pelos eventos de usuário. A Figura 28 ilustra este conjunto de negociações e autenticações.

Por outro lado, o *middleware* sustenta um funcionamento interno de comunicação para que a conexão com o servidor esteja em perfeito sincronismo com o sistema gráfico remoto do Módulo de Comunicação Cliente. Esta comunicação inter-módulos tem uma seqüência definida dos eventos a partir do *boot* do sistema até o encerramento da conexão remota. A Figura 29 ilustra toda a comunicação entre os módulos do *middleware* WSE-OS.

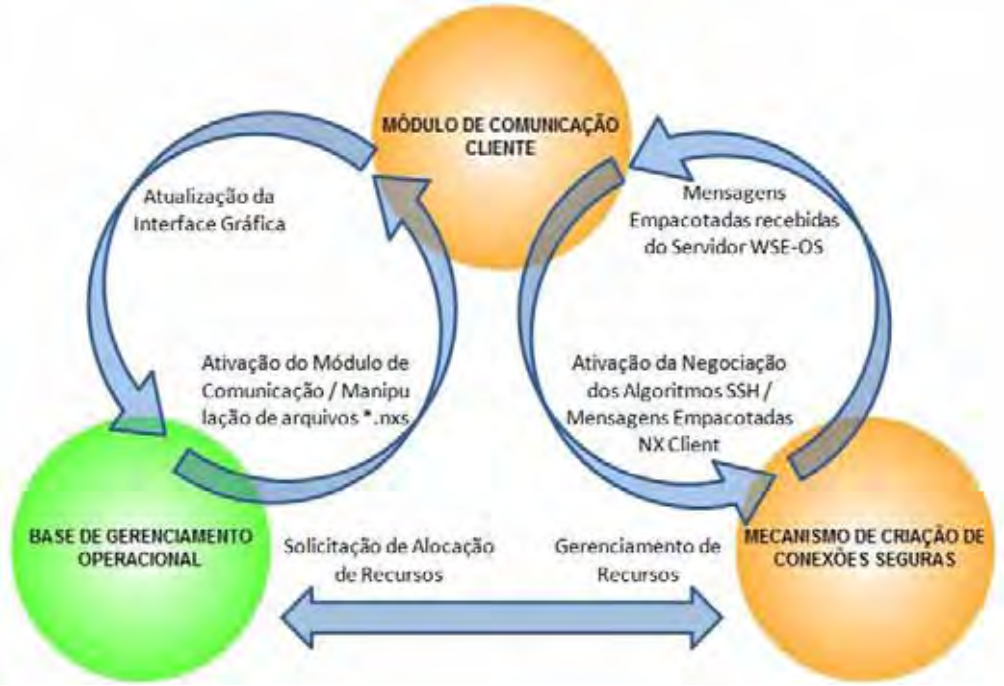


**Figura 29 - Fluxo de Execução do *Middleware* multicamada WSE-OS**

O fluxo de execução mais intenso é entre o Módulo de Comunicação Cliente e o Módulo de Mecanismo de Criação de Conexões Seguras. São nestes módulos de comunicação e conexão que ocorre a leitura e interpretação das instruções recebidas do servidor WSE-OS. Quando o módulo de conexão recebe a mensagem de encerramento de seção remota, imediatamente todo o fluxo entre estes módulos é desfeito.

Nesta situação, novamente é ativado o *script* de interface (interface_sew_os.sh) com o usuário. O Módulo de Comunicação Cliente envia uma mensagem ao Módulo Base de Gerenciamento Operacional solicitando a execução imediata do arquivo. Isto garante que, mesmo com a conexão desfeita, o usuário WSE-OS possa realizar um novo processo de autenticação para a escolha de instanciação de outro sistema operacional. É importante ressaltar que todo o controle de imagens para cada cliente do ambiente WSE-OS se encontra no Módulo de Controle de Privilégios na Camada de Gerenciamento do Servidor. A Figura 30 ilustra este mecanismo de solicitação de conexão.

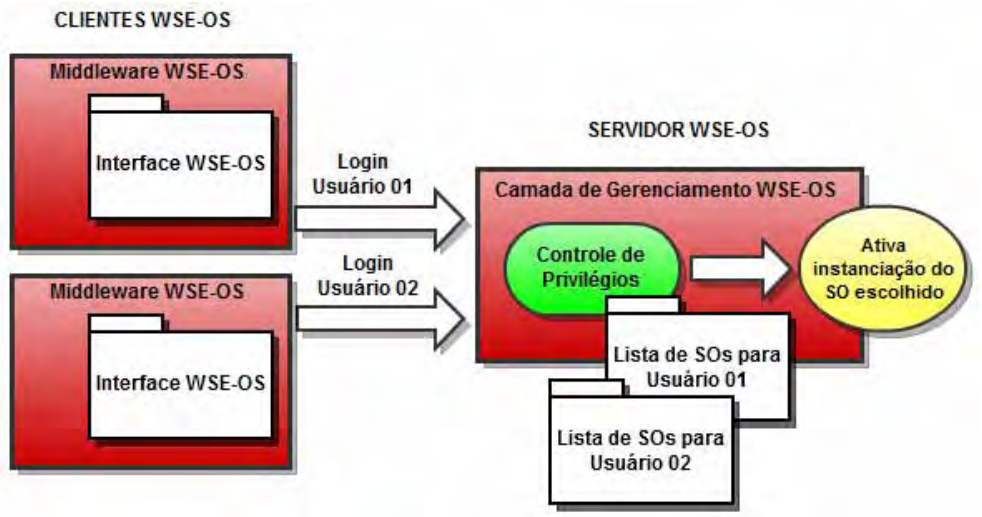


**Figura 30 - Controle de Listas de Sistemas Operacionais por Usuário**

A partir do *login* no servidor, o cliente WSE-OS consegue selecionar um sistema operacional em uma lista pré-definida pelo administrador do sistema. Como dito anteriormente, este controle de listas e usuários estão sob responsabilidade do Módulo Controle de Privilégios da Camada de Gerenciamento WSE-OS. Após a escolha do usuário, o Módulo Controle de Privilégios ativa a instanciação do SO e estabelece parâmetros de hardware (memória RAM, disco rígido, resolução de tela) no servidor para o cliente em questão (DIGIERE, 2011).

## 5.4 Considerações Finais

Com a estruturação proposta, o *middleware* é capaz de tornar parte integrante para uma solução de gerenciamento centralizado sem fio. Esta abordagem apenas é válida diante das seguintes características oferecidas por este mediador:

- Utilização de acesso remoto seguro e sistema gráfico baseado em conexão TCSC para a sincronização de componentes de *hardware* e *software* do sistema;
- Caracterização modular orientada para soluções VDI, estabelecendo uma infra-estrutura básica para a chamada de instanciações de sistemas operacionais;
- Não obrigatoriedade de utilização do protocolo PXE para o processo de *boot*, refletindo em flexibilidade e facilidade de instalações em terminais utilizando mídias secundárias;

- Abstração de *hardware* oferecida pelo Módulo Base de Gerenciamento Operacional atendendo requisitos de diferentes plataformas clientes em que são executados;
- Implementação do STD no Módulo de Comunicação Cliente, reduzindo tráfego de dados na comunicação orientada a fluxo;
- *Boot* do *middleware* WSE-OS com execução local a partir da mídia secundária, resguardando memória RAM e parte da memória *Flash Driver* USB para configuração de *cache* de comunicação.

# 6. EXPERIMENTOS

## 6.1 Considerações Iniciais

Este Capítulo apresenta os resultados dos experimentos realizados sobre o *Middleware* WSE-OS para a indicação de desempenho em diferentes plataformas clientes. Embora o projeto WSE-OS seja constituído por duas partes principais (*Middleware* WSE-OS e Camada de Gerenciamento Servidor), as avaliações deste capítulo de experimentos, em maior parcela das seções, estão voltadas para o *Middleware* WSE-OS. Análise de desempenho e considerações sobre a Camada de Gerenciamento WSE-OS em relação ao mecanismo de virtualização e configurações são vistas em (DIGIERE, 2011).

A base de dados dos experimentos realizados neste capítulo reúne informações sobre o tempo de inicialização do *middleware* WSE-OS e instanciação do sistema operacional de referência, além de seu desempenho em função da configuração do Sistema de Transmissão Diferencial de Dados (STD). Com os resultados é possível compreender a viabilidade de uso deste mediador como parte integrante de um ambiente de compartilhamento sem fio para sistemas operacionais.

## 6.2 Parâmetros de Configuração

O *middleware* WSE-OS possui dois grandes parâmetros de configuração para que a conexão remota seja realizada com excelência de desempenho. Os algoritmos do Módulo de Mecanismo de Criação de Conexões Seguras junto com o Sistema de Transmissão Diferencial de Dados (STD) devem ser editados de modo que garantam eficiência na troca de mensagens e segurança destes dados. Embora apenas um destes dois parâmetros possa ser editado pelo usuário do sistema, é importante ilustrar a escolha dos algoritmos de negociação SSH que será padrão e sem possibilidade de edição para qualquer nova conexão com o servidor WSE-OS.

Por outro lado, tanto o STD quanto os algoritmos SSH estão inicialmente parametrizados para a execução em clientes de referência, ou seja, clientes nos quais assumem requisitos mínimos para integrar ao sistema WSE-OS (Tabela 3; seção 6.3). Os testes de avaliação de inicialização do *middleware*, configurações SDT e outros parâmetros de desempenho foram conduzidos sob monitoramento da suíte *benchmark* SANDRA (SISOFTWARE, 2010) e *InterMapper Flows* (DARTWARE, 2011).

O aplicativo *InterMapper Flows* é uma ferramenta de monitoramento e análise para redes sem fio 802.11 a/b/g/n capaz de capturar pacotes de dados transmitidos entre cliente e servidor ilustrando informações úteis para que os testes deste capítulo fossem realizados com sucesso. Com diversos protocolos suportados, o analisador permite a análise detalhada de transmissão e recebimento de pacotes clientes oferecendo recursos para medição do tempo de inicialização de aplicativos e sistemas operacionais replicados no *middleware* WSE-OS através de comunicação remota TCSC.

O aplicativo SANDRA (*System Analyser Diagnostic and Reporting Assistant*) é um sistema que oferece um conjunto de ferramentas para *benchmark* projetadas para auxiliar o diagnóstico de desempenho de diferentes conjuntos de *hardwares* como, por exemplo, processadores, placas gráficas, sistemas de memória, multimídia, rede, *motherboard.* A suíte de monitoramento, no caso deste projeto, tem a função de dispor relatórios sobre medições de tempo de *boot* de aplicativos/sistemas operacionais através de execução nativa em servidor WSE-OS e máquinas clientes.

Através dos dados obtidos a partir destas duas ferramentas de monitoramento é possível correlacionar dados de execução remota e nativa, servindo como ponto de referência para discutir as diferenças de desempenho e a viabilidade de uso do *middleware* associado ao servidor WSE-OS como ferramenta de replicação de sistemas operacionais em redes sem fio. Todos os resultados dos experimentos que serão apresentados em gráficos e tabelas ilustram valores médios calculados sobre dez execuções em ambiente e configurações de *hardware* que serão detalhadas nas próximas seções. Quando necessário, os desvios padrões e intervalos de confiança, baseado em distribuições t de *Student* (FARHAT, 2008) com intervalo de confiança de 90%, serão ilustrados associados aos experimentos representando possíveis dispersões estatísticas.

### 6.2.1 Algoritmos do Mecanismo de Criação de Conexões Seguras

As escolhas dos algoritmos de criptografia, integridade, compressão e troca de chaves são de grande importância para o desempenho do mediador do sistema WSE-OS. Diante disto, o objetivo desta parametrização padrão do *middleware* é garantir a segurança e eficiência nos dados trocados entre cliente e servidor. Por outro lado, devem-se considerar todas as configurações na Camada de Gerenciamento WSE-OS do servidor, pois embora independentes em sua estruturação, estes mantêm a cooperação de funcionalidade para estabelecer conexões remotas para o alto desempenho.

O Módulo de Mecanismo de Criação de Conexões Seguras do *middleware* WSE-OS trabalha baseando-se em três camadas: camada de transporte, camada de autenticação de usuários e a camada de conexão. Como já visto na seção 5.2.2, a camada de transporte deste módulo é responsável pela negociação dos algoritmos de confidencialidade, integridade e compressão de dados.

No primeiro cenário de negociação, onde ocorre o compartilhamento do PSK e autenticação do servidor WSE-OS, o *middleware* emprega chaves de 512bits para criptografia dos dados utilizando o algoritmo RSA (JONSSON, 2003). Isto garante sigilo absoluto no caso de interceptação da informação nesta rede sem fio que compõe o sistema (YLONEN, 2006a). Por outro lado, como previsto por (YLONEN, 2006b), o processamento para codificação utilizando chaves de 512bits é altamente penoso, em alguns casos inviabilizando o fluxo contínuo de dados na rede de comunicação.

Embora o próprio módulo SSH recomende a utilização de chaves de 512bits ou mais para a primeira fase da conexão, este fornece, após a autenticação do servidor, uma lista de algoritmos e chaves funcionalmente mais simplistas. A Tabela 1 ilustra estas opções. Ao mesmo tempo em que assegura confidencialidade em uma fase já não mais crítica (troca de chaves e autenticação de servidor), a utilização destas chaves faz com que o volume do processamento seja reduzido, melhorando o desempenho do *middleware* WSE-OS.

Por padrão e seguindo as recomendações dos *Request for Comments* para SSH, o mediador em sua segunda fase de conexão utiliza algoritmo 3DES (*Triple Data Encryption Standard*) com três chaves de 64bits cada em modo CBC (*Cipher Block Chaining*). Em relação aos algoritmos de integridade e compressão dos dados, como visto na seção 5.3.2, estes também são negociados na segunda fase da conexão remota na camada de transporte do Módulo de Mecanismo de Criação de Conexões Seguras.

**Tabela 1 - Algoritmos de Criptografia OpenSSH versão 1.2.12**

| | | |
|---|---|---|
| **3des-cbc** | **RECOMENDADO** | 3DES em modo CBC |
| **blowfish-cbc** | **OPCIONAL** | Blowfish em modo CBC |
| **twofish256-cbc** | **OPCIONAL** | Twofish em modo CBC com chave de 256 bits |
| **twofish-cbc** | **OPCIONAL** | Twofish padrão em modo CBC |
| **twofish192-cbc** | **OPCIONAL** | Twofish em modo CBC com chave de 192 bits |
| **twofish128-cbc** | **OPCIONAL** | Twofish em modo CBC com chave de 128 bits |
| **aes256-cbc** | **OPCIONAL** | AES em modo CBC com chave de 256 bits |
| **aes192-cbc** | **OPCIONAL** | AES em modo CBC com chave de 192 bits |
| **aes128-cbc** | **OPCIONAL** | AES em modo CBC com cahve de 128 bits |
| **serpent256-cbc** | **OPCIONAL** | Serpent em modo CBC com chave de 256 bits |
| **serpent192-cbc** | **OPCIONAL** | Serpent em modo CBC com chave de 192 bits |
| **serpent128-cbc** | **OPCIONAL** | Serpent em modo CBC com chave de 128 bits |
| **Arcfour** | **OPCIONAL** | ARCFOUR com chave de 128 bits |
| **idea-cbc** | **OPCIONAL** | IDEA em modo CBC |
| **cast128-cbc** | **OPCIONAL** | CAST em modo CBC com chave de 128 bits |
| **None** | **OPCIONAL** | Sem Criptografia; Não Recomendado |

A única opção deste módulo para a compressão dos dados é o algoritmo *Zlib* (DEUTSH, 1996). No entanto, o Módulo de Comunicação Cliente utiliza algoritmos mais eficientes de compressão de acordo com o dado transmitido. Isto será discutido na seção seguinte sobre o SDT do *middleware* WSE-OS. Por outro lado, a escolha do algoritmo de integridade segue a padronização da configuração SSH. A Tabela 2 ilustra as opções possíveis.

**Tabela 2 - Algoritmos de Integridade OpenSSH versão 1.2.12**

| | | |
|---|---|---|
| **hmac-sha1** | **RECOMENDADO** | HMAC-SHA ( digest length = key length = 20 ) |
| **hmac-sha-96** | **OPCIONAL** | Primeiros 96 bits de HMAC-SHA1 ( D.L.=12 K.L.=20 ) |
| **hmac-md5** | **OPCIONAL** | HMAC-MD5 ( D.L.= K.L = 16 ) |
| **hmac-md5-96** | **OPCIONAL** | Primeiros 96 bits de HMAC-MD5 ( D.L.=12 K.L.=16 ) |
| **None** | **OPCIONAL** | Sem MAC; Não Recomendado |

Para a computação de integridade dos dados, o *middleware* WSE-OS utiliza o algoritmo MAC *hmac-sha1* (KRAWCZYK et. al., 1997). Este algoritmo se adéqua ao perfil do sistema WSE-OS. Além de garantir a integridade dos dados, o tamanho da chave de trabalho e a técnica de divisão dos blocos monitorados asseguram um melhor desempenho em redes com limitações de bandas (YLONEN, 2006b).

Ainda que estas configurações sejam padrão e sem possibilidade de edição, ou seja, o usuário do sistema WSE-OS não tem o privilégio necessário para alterá-las, é importante descrever os algoritmos utilizados no *middleware* WSE-OS. Para atender aos requisitos do sistema e se adequar ao *middleware* de comunicação remota, é importante que todos esses processos de integridade, compressão e criptografia de dados estejam em sincronismo com o servidor WSE-OS. Isto é, embora não obrigatórias, as escolhas dos algoritmos para a comunicação em excelência devem ser as mesmas tanto no cliente quanto na Camada de Gerenciamento WSE-OS no servidor.

### 6.2.2 Sistema de Transmissão Diferencial

Como já apresentado anteriormente, o Módulo de Comunicação Cliente do *middleware* WSE-OS possui um sistema de transmissão diferencial de dados (SDT) para a comunicação cliente-servidor TCSC. O SDT é composto por memórias *caches* e sistema de compressão otimizado para diferentes tipos de dados. Estes parâmetros podem ser editados pelo usuário do sistema para aumento de desempenho, ajustando o mediador para necessidades do ambiente de trabalho no qual está inserido.

As memórias *caches* estão divididas em principais (Memória RAM) e secundárias (*Flash Driver* USB). As corretas alocações destes *caches* variam conforme o *hardware* cliente disponível. No caso deste estudo, todos os testes foram realizados considerando uma especificação de referência. Por padrão estabelecido e seguindo recomendações mínimas para alto desempenho do *NXCLient* (REGIS,2007), os testes consideram:

- 64 Mbytes de *cache* em Memória Secundária – Memória *Flash Driver* USB.
- 16 Mbytes de *cache* em Memória Primária – Memória RAM.

Em relação aos algoritmos, o *middleware* oferece a edição de compressão de imagens. A escolha pode ser feita pela utilização de algoritmos de compressão de imagens *JPEG, PNG* ou sem compressão específica. No caso da opção sem compressão, o *middleware* utiliza o algoritmo ZLIB para o empacotamento e desempacotamento dos arquivos referentes a imagens. Isto pode refletir em perda de desempenho, pois comandos de *unmarshalling* para algoritmos genéricos de compressão exigem um maior grau de processamento do *middleware* em mídias secundárias.

O algoritmo *JPEG*, como escrito por (RANI, 2009), tem uso em bandas de redes limitadas. No entanto, é um formato com perdas. Ou seja, há uma alteração sutil da imagem. Não é o tipo de compressão recomendado para arquivos de imagens que necessitem de linhas nítidas e/ou bordas muito definidas.

Por outro lado, o formato PNG não degrada as imagens, ou seja, não possui perdas em sua compressão (TSAI, 2010). Embora seja excelente para imagens melhores definidas, o formato PNG exige pacotes com maior unidade de bytes para sua transmissão, podendo assim refletir negativamente no desempenho da rede de comunicação.

Algoritmos responsáveis pela compactação de áudio, vídeo e textos são pré definidos pelo NXClient versão 3.4.0 utilizado no Módulo de Comunicação Cliente do *middleware* WSE-OS. Desta forma, o *middleware* padrão de testes considera:

- Algoritmo *WAV* para compressão de áudio;
- Algoritmo *MPEG* para compressão de vídeo;
- Algoritmo *ZLIB* para compressão de textos;
- Algoritmo *JPEG* para compressão de imagens.

O SDT implementa em seu mecanismo de *unmarshalling* algoritmos para a descompressão de áudio, vídeo, imagem e texto recebidos do servidor WSE-OS para que a montagem de tela cliente seja realizada. Como a comunicação é bidirecional, todos estes algoritmos devem estar sincronizados tanto no cliente quanto no servidor, ao contrário dos algoritmos de criptografia e integridade do Módulo de Mecanismo de Criação de Conexão Segura que, embora não recomendado, podem ser assíncronos.

## 6.3 *Boot Middleware* WSE-OS

Os testes de *boot* do *middleware* WSE-OS extraíram informações de tempo relativo à inicialização deste sistema mediador em quatro diferentes cenários. Em função da camada de virtualização no servidor WSE-OS e ao módulo de comunicação remota cliente-servidor, o *middleware* pode ser replicado a diferentes conjuntos de *hardware*. Diante disto, os testes consideraram quatro máquinas clientes com distintas configurações para cada cenário onde o *middleware* foi avaliado. A Tabela 3 ilustra estas configurações clientes.

**Tabela 3 - Configuração *Hardware* Cliente para Ambiente de Teste**

| | PROCESSADOR | *MOTHERBOARD* | MEMÓRIA PRINCIPAL | PLACA DE REDE SEM FIO |
|---|---|---|---|---|
| **CLIENTE 01** | Intel Pentium 4 HT @1.0GHz | ASUS P4P800 865pe chipset | 640MB DDR @398MHz | Cartão para Rede Sem Fio Atheros AGN |
| **CLIENTE 02** | Intel Dual Core E4500 @2.2GHz | ThinkCent. IntelQ43 | 1.0GB Dual-Channel DDR @400MHz | Cartão para Rede Sem Fio Atheros AGN |
| **CLIENTE 03** | Intel Core 2 Duo P8600 @2.4GHz | Sony Corp. VAIO GM47 | 4.0GB Dual-Channel DDR2 @398MHz | Intel WiFi Link 5100 AGN |
| **CLIENTE 04** | Intel Core 2 Quad Q9400 @2.66GHz | ThinkCent. IntelQ43 | 4.0GB Triple-Channel DDR3 @532MHz | Cartão para Rede Sem Fio Atheros AGN |

Embora o processo de *boot* não envolva diretamente a comunicação cliente-servidor para a instanciação de um sistema operacional remotamente virtualizado, é neste cenário de inicialização do *middleware* que ocorre autenticação da interface da placa sem fio cliente e processo DHCP para a concessão de endereço IP para clientes da rede e, por isto, calculado no tempo de teste.

Desta forma, o experimento considerou que o processo de inicialização do *middleware* envolve o carregamento do Módulo Base de Gerenciamento Operacional, verificação da interface de rede cliente e atribuições DHCP do BPAC (Base de Ponto de Acesso Central – Roteador WSE-OS). Desta forma, a avaliação de desempenho da conexão remota – Módulo de Mecanismo de Criação de Conexões Seguras - associado ao SDT - Módulo Comunicação Cliente – são avaliadas na próxima seção.

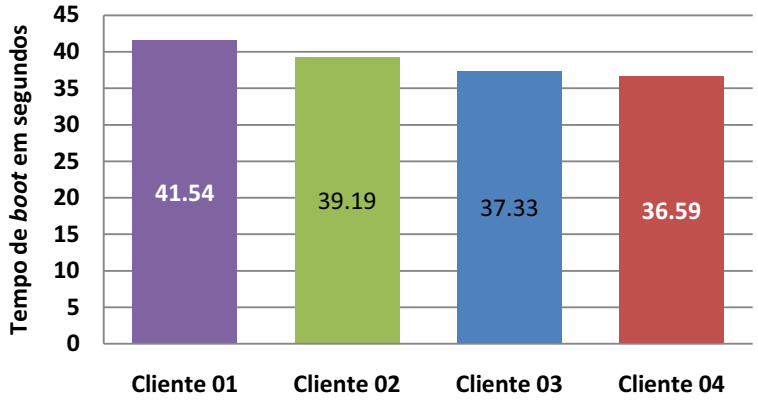


**Figura 31 - Tempo de boot do middleware WSE-OS para diferentes configurações de hardware**

O desempenho de inicialização do *middleware* WSE-OS tem uma variação decrescente de até 12% do tempo em configurações de *hardware* que possuem melhores processadores de dados e maiores memórias principais comparadas às máquinas que obedecem aos requisitos mínimos de *hardware* para o ambiente de projeto. Os resultados dos experimentos para os clientes 01, 02, 03 e 04 apresentam desvios padrões/intervalos de confiança de 0.32/ [41.35, 41.72], 0.25/ [39.04, 39.33], 0.22/ [37.20, 37.45] e 0.18/ [36.48, 36.69] segundos respectivamente.

Ainda que o *middleware* WSE-OS seja executado e inicializado a partir de mídia secundária (*Flash Driver* USB), sem cópia integral para a memória principal do *hardware* cliente, o tempo de *boot* entre os dois extremos de configurações clientes pode ser considerado esperado (menor do que 5 segundos). A esta diferença, além dos já mencionados, se deve a diferentes placas de rede sem fio clientes onde há perda pela verificação da interface de rede pela procura do SSID (*Service Set IDentifier*) do BPAC para desencadear o processo DHCP.

## 6.4 Conexão Remota e Avaliação de Desempenho

As extrações de informações referentes ao tempo de conexão remota e ao comportamento parametrizado do Sistema de Transmissão Diferencial de Dados (STD) fornecem referências para a avaliação de desempenho do *middleware* WSE-OS. Associado a apreciação dos dados coletados da seção

anterior e contextualizando o sistema WSE-OS à ambientes de trabalho corporativos, estes indicadores ilustram a viabilidade de uso do sistema como ferramenta estrutural de um ambiente sem fio compartilhado para instanciações de sistemas operacionais.

Embora o sistema WSE-OS ofereça, de acordo com o nível de privilégios, diversas imagens de sistemas operacionais para a instanciação, os testes da próxima seção foram realizados baseando-se em uma imagem de referência. Desta forma, os resultados serviram como ponto de análise de desempenho. As próximas seções avaliam, simultaneamente, o Módulo de Comunicação Cliente e Módulo de Mecanismo de Criação de Conexões Seguras.

### 6.4.1 Instanciação Remota de Sistema Operacional

Para realização do teste de instanciação remota de sistema operacional foram criados três cenários de inicialização para a imagem de referência *Windows XP Service Pack3.* De forma a validar o experimento, o primeiro cenário ilustra o tempo de *boot* da imagem de referência a partir de uma instalação local no servidor WSE-OS, o segundo com a imagem virtualizada a partir do acesso em disco rígido no servidor WSE-OS, e o terceiro com a imagem remotamente virtualizada com comunicação TCSC sem fio a partir de um modelo cliente 3 descrito na Tabela 3.

Para a padronização dos experimentos, a alocação de memória do servidor WSE-OS reservou 192MB para a execução em máquina virtual com 10GB de disco padrão da máquina virtual (.vdi) para a imagem de referência. Assim, todos os dados dos testes seguintes são extraídos da seguinte configuração de servidor para o ambiente de compartilhamento sem fio de sistemas operacionais:

**Tabela 4 - Configuração de *Hardware* Servidor WSE-OS**

| | **PROCESSADOR** | ***MOTHERBOARD*** | **MÁQUINA VIRTUAL** | **MEMÓRIA PRINCIPAL** | **ROTEADOR WSE-OS** |
|---|---|---|---|---|---|
| **SERVIDOR WSE-OS** | Intel Xeon E5540 @2.53GHz 45nm Technology | Intel Corporation S5520SC | Virtual Box SunMicrosystems V.4.0 | 16GB Triple-Channel DDR3 @533MHz | 300M *Wireless* N Roteador, Modelo TL-WR941ND |

O comparativo da Tabela 5 constata que para o ambiente com apenas um cliente WSE-OS, o tempo de *boot* (imagem de referencia) ativado pela comunicação TCSC é 5.13 segundos mais lento que uma inicialização nativa e 3.85 segundos mais lento que esta imagem virtualizada com acesso em disco rígido no servidor WSE-OS. É importante notar que este tempo não inclui o tempo do *middleware* WSE-OS calculado na seção anterior e os desvios padrões/intervalos de confiança para os dados da Tabela 5 são de 0.33/ [13.85, 14.24], 0.42/ [15.08, 15.57] e 0.25/ [19.04, 19.33] segundos para tempo de *boot* nativo em servidor, virtualizado em servidor e remotamente virtualizado com comunicação TCSC respectivamente.

**Tabela 5 - Tempo de boot da imagem de referência para um computador cliente**

| Sistema Operacional | Modo de Inicialização da Imagem | Memória RAM alocada pelo Servidor | Tempo de *boot* em segundos |
|---|---|---|---|
| **Windows XP Service Pack3** | Nativo com Acesso em Disco Rígido | 16GB | 14.05 |
| **Windows XP Service Pack3** | Virtualizado com Acesso em Disco Rígido | 192MB | 15.33 |
| **Windows XP Service Pack3** | Remotamente Virtualizado com Comunicação TCSC | 192MB | 19.18 |

Esta latência adicional no *boot,* ativado pela conexão remota através da comunicação TCSC, se deve a dois fatores principais: técnicas para transmissão de pacotes e execução em mídia secundária (*Flash Driver* USB). Como dito na seção 5.2.3, o Módulo de Comunicação Cliente trabalha associado ao Módulo de Comunicação Servidor através de algoritmos de criptografia, integridade e compressão pré-negociados. A cada pacote recebido pelo cliente WSE-OS, há a necessidade do processo de desencriptação, descompressão dos dados e verificação da integridade antes do processamento final para atualização de interface gráfica. Por outro lado, o tempo de *boot* pode ser reduzido com o acréscimo de alocação de memória para a máquina virtual no servidor. Embora o *middleware* esteja configurado para utilizar algoritmos e cifras mais simples a partir da negociação inicial (ver seção 6.2.1), este volume de processamento ainda se torna penoso pelo fato do *middleware* ser executado a partir de mídias secundárias (*Flash Driver* USB). A Figura 32 ilustra o tempo total de *boot* do sistema WSE-OS (*middleware* WSE-OS *+* instanciação remota da imagem de referência). Com este experimento é possível notar que o tempo de inicialização do sistema sofre variação de acordo com o *hardware* cliente disponível (Tabela 3).

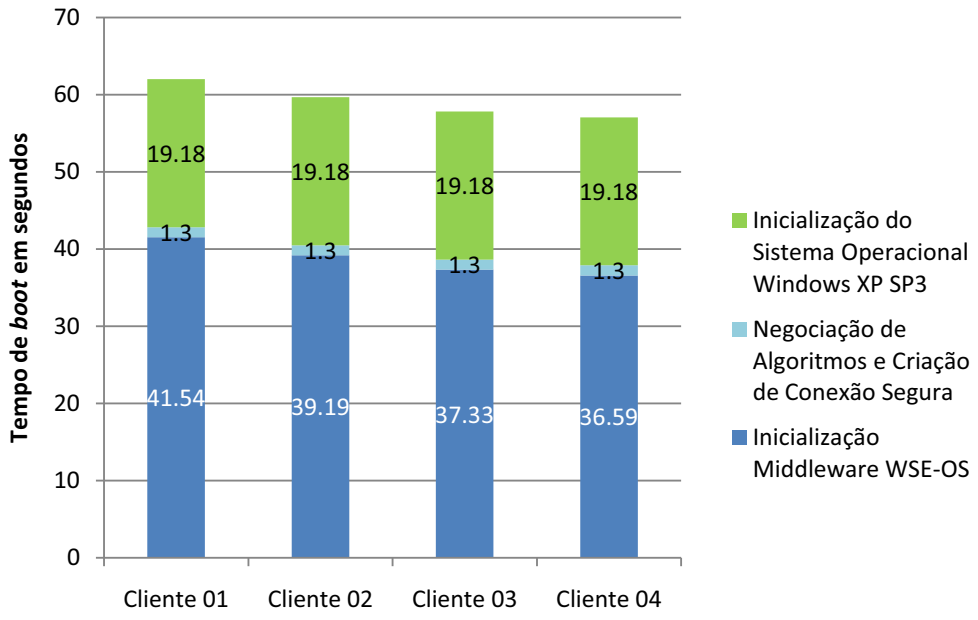


**Figura 32 - Tempo de boot do sistema WSE-OS em diferentes configurações de hardware**

Como ilustra a Figura 32, é somado ao tempo de *boot* o intervalo de negociação dos algoritmos de criptografia, integridade e compressão entre cliente-servidor que apresenta desvio padrão/intervalo de confiança de 0.12/ [1.23, 1.36] segundos. Ainda que o *middleware* esteja configurado para utilização de *caches* em memória RAM e memória *Flash Driver* USB, este proveito só foi observado a partir da segunda instanciação da imagem de referência. Conforme ilustra a Figura 33, o ganho no tempo de *boot* foi notável, com desempenho 7.16% superior ao primeiro processo de inicialização e com desvio padrão/intervalo de confiança de 0.37/ [53.45, 53.88] segundos. O modelo cliente 03, descrito na Tabela 3, foi utilizado como referência.

Como a otimização em *cache Flash Driver* USB tem armazenamento permanente para pacotes recebidos de solicitações anteriores, o desempenho dos próximos processos de inicialização será mantido com maior excelência mesmo após o desligamento do terminal cliente WSE-OS. Por outro lado, nota-se que o tempo de inicialização do *middleware* sofreu acréscimo de 0.29 segundos justamente pelo fato do pré-carregamento do *cache* em memória *flash* a partir do mediador.

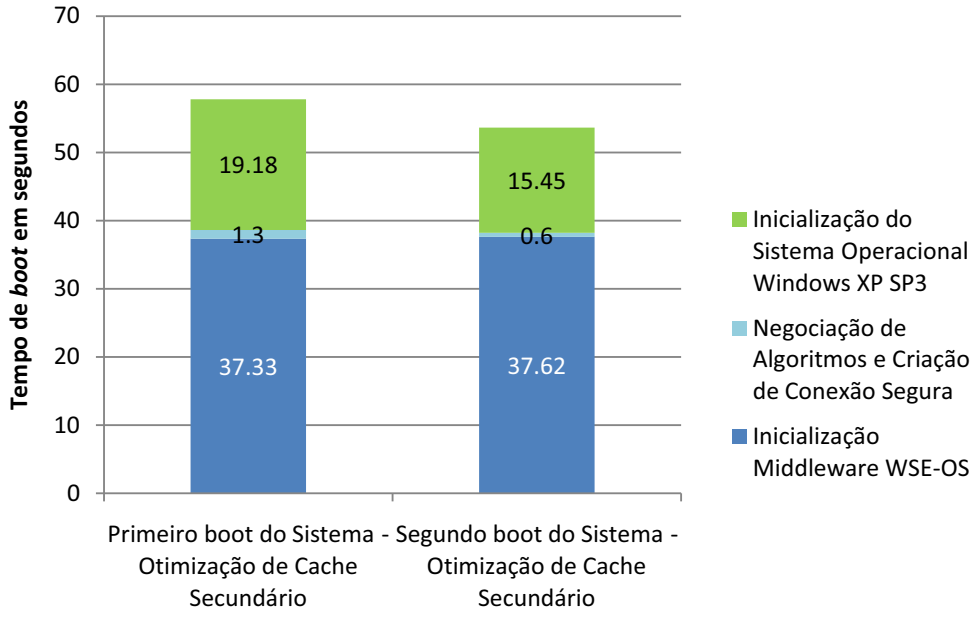


**Figura 33 - Comparação entre 1˚ e 2˚ processo de boot com utilização de *cache* otimizado**

Mesmo com o decréscimo de desempenho de *boot* do mediador, ainda é justificado o uso desta vantagem estabelecida pelo SDT. Neste caso, tanto o tempo de negociação e estabelecimento de conexão remota quanto a inicialização da imagem de referência foram menores comparados as mesmas fases do primeiro processo de *boot.* Seguindo esta linha, o *cache* em memória RAM, por não possuir memória estável, só se torna uma vantagem ao sistema após a primeira instanciação desde que não ocorra novo processo de *boot* do sistema. Estes dados serão analisados e discutidos com maior atenção na seção 6.4.3.

### 6.4.2 *Boot* Nativo X *Boot* Sistema WSE-OS

Os testes realizados em seções anteriores fornecem informações sobre o desempenho de *boot* do *middleware* WSE-OS associado com o sistema de virtualização remoto baseado em uma imagem de sistema operacional de referência para a transmissão de pacotes desta instanciação. Para uma visualização direta do nível de degradação causado pelas técnicas utilizadas, a Figura 34 compara o tempo de *boot* da imagem do sistema operacional *Windows XP Service Pack3* em execução de *hardware*

cliente 03 descrito na Tabela 3 com desvio padrão/intervalo de confiança de 0.42/ [47.60, 48.09] segundos e em sistema WSE-OS através da comunicação remota utilizando a mesma plataforma *hardware* cliente com desvio padrão/intervalo de confiança de 0.37/ [53.45, 53.88] segundos.

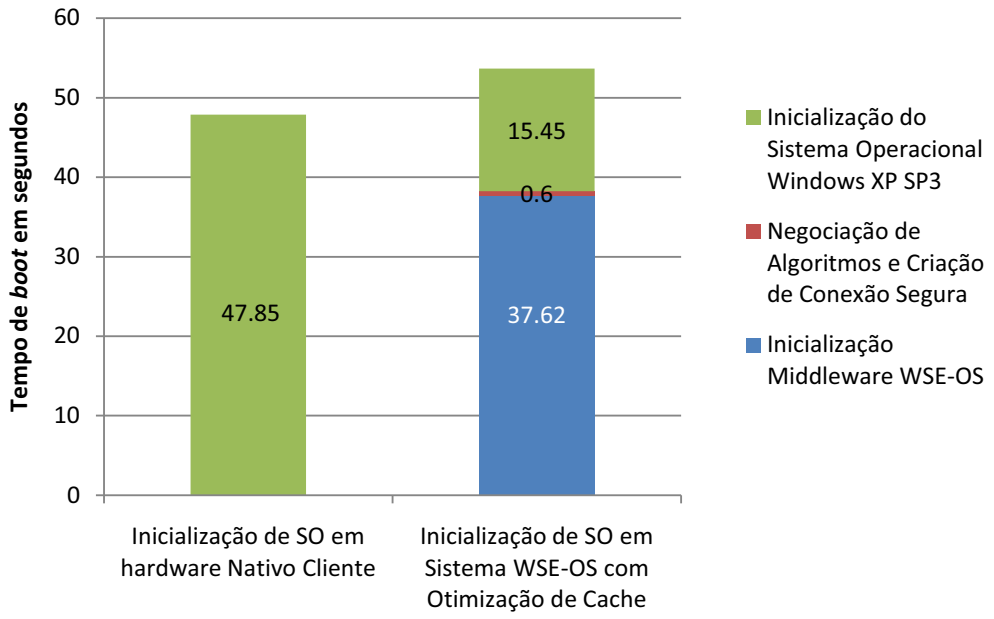


**Figura 34 - Tempo de boot SO referência: execução nativa em cliente X execução sistema WSE-OS**

Neste cenário de comparação, o sistema WSE-OS utiliza otimização no *middleware* através da memória *cache* em *Flash Driver* USB. A partir desta configuração e em execução em modelo cliente 03 descrito na Tabela 3, o sistema WSE-OS tem um acréscimo de 5.82 segundos comparado ao tempo de *boot* nativo em *hardware* cliente. Este acréscimo representa cerca de 12.16% de degradação para a instanciação de uma imagem de sistema operacional de referência pelas técnicas utilizadas no sistema WSE-OS associado com o modelo de transmissão de pacotes em rede sem fio.

### 6.4.3 Desempenho do Sistema de Transmissão Diferencial

Para uma extração de dados mais específicos do Sistema de Transmissão Diferencial de Dados e sua avaliação de desempenho, foram comparados os tempos de inicialização de quatro diferentes

aplicativos: *Easy Media Creator 10IJ* (ROXIO, 2010), *FrostWire 4.18.3* (SOURCEFORGE, 2010), *Google Chrome 8.0* (GOOGLE, 2011) e *Microsoft Word 2007* (MICROSOFT, 2007). As chamadas de instanciações remotas foram executadas em uma mesma máquina cliente através do *middleware* WSE-OS, por padrão estabelecido cliente03 descrito da Tabela 3, para que as informações pudessem ser correlacionadas. As configurações de alocação de *cache* em memória RAM e memória *Flash Driver* USB seguem os parâmetros discutidos na seção 6.2.2. A Figura 35 e 36 apresenta estes resultados. Cálculo de desvios padrões e intervalos de confiança para esses experimentos encontram-se na Tabela 6.

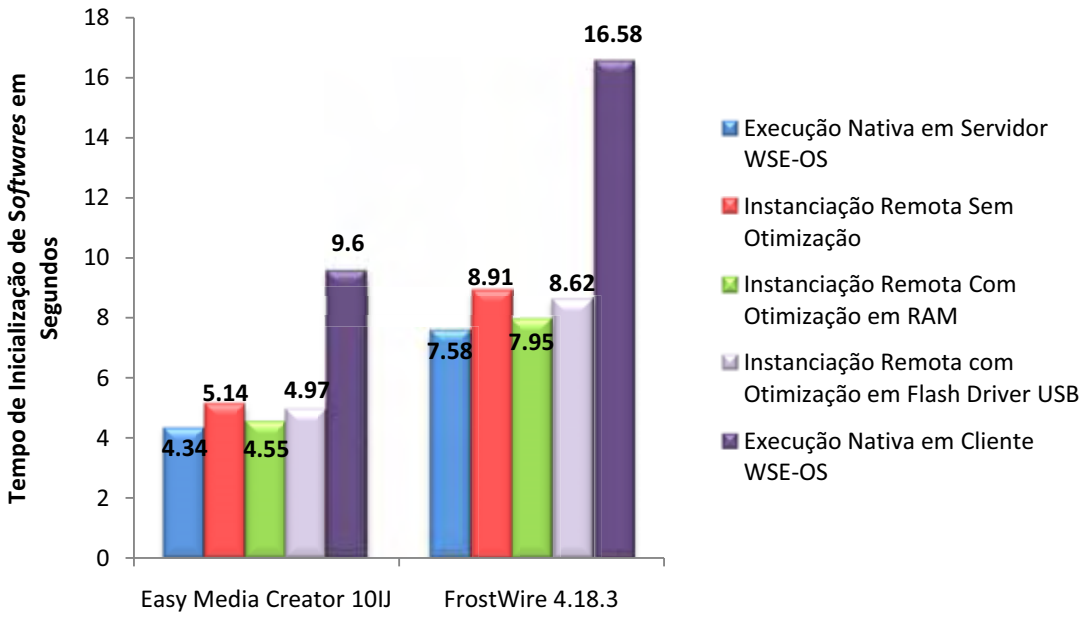


**Figura 35 - Tempo de inicialização de aplicativos: Media Creator 10IJ X FrostWire 4.18.3**

A execução nativa de cada um dos aplicativos em servidor WSE-OS e cliente modelo 3 descrito na Tabela 3, fornece o ponto de saída para a análise de desempenho do STD. A Figura 35 revela que embora exista uma latência na instanciação remota sem otimização comparada a execução nativa em servidor, o ganho de eficiência de carregamento supera a execução nativa destes *software* em configurações de *hardware* apresentado pelo modelo cliente 3 da Tabela 3. Em situações onde houve atuação do STD com o sistema de memória *cache*, tanto as otimizações em memória RAM quanto em memória *Flash Driver* USB tiveram desempenho de carregamento remoto próximos ao da execução nativa em servidor.

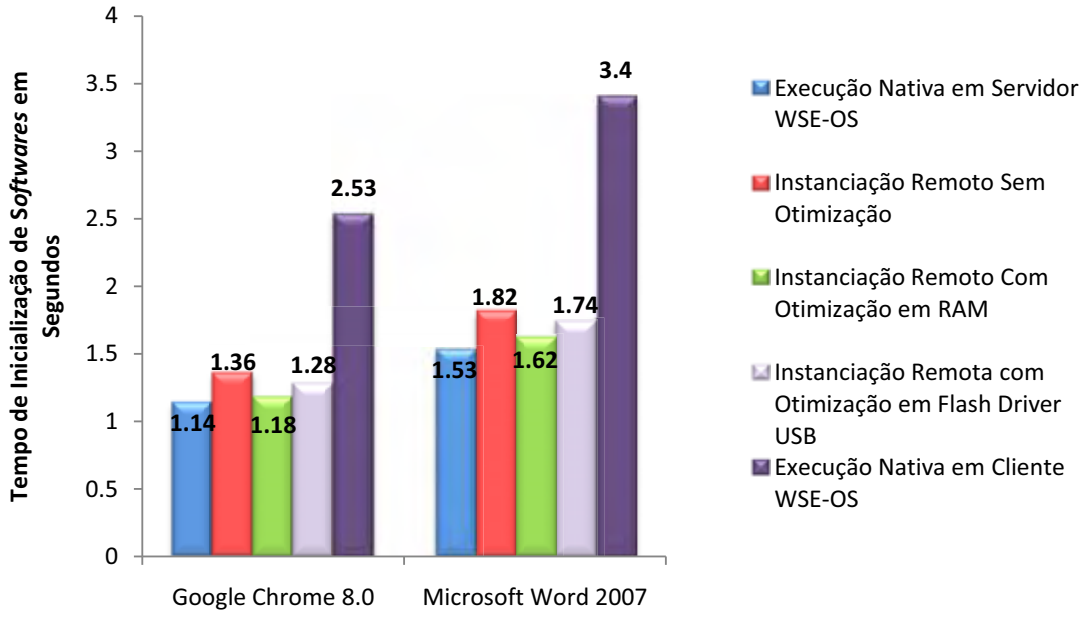


**Figura 36 - Tempo de inicialização de aplicativos: Google Chrome 8.0 X Microsoft Word 2007**

Nos quatro cenários apresentados para a instanciação remota, o carregamento dos *software através* da otimização por memória RAM tem melhor desempenho. Isto é previsto, visto que uma vez na memória principal do cliente WSE-OS, a inicialização de atualização de interface gráfica cliente tende a se tornar mais rápida do que a busca em *cache Flash Driver* USB. No entanto, o mecanismo de *marshalling/unmarshalling* do Módulo de Comunicação Cliente reflete em diferentes níveis de degradações na utilização ou não desses diferentes *caches*.

Para ilustrar as dispersões estatísticas dos experimentos de desempenho do sistema de transmissão diferencial, a Tabela 6 indica os desvios padrões e intervalos de confiança dos aplicativos *Easy Media Creator* 10IJ, *FrostWire 4.18.3, Google Chrome* 8.0 e *Microsoft Word* 2007 executados a partir de cinco modos distintos de *boot* conforme já exemplificado anteriormente: execução nativa em Servidor WSE-OS, instanciação remota sem otimização, instanciação remota com otimização em RAM, instanciação remota com otimização em *Flash Driver* USB e execução nativa em cliente WSE-OS.

**Tabela 6 - Desvio Padrão e Intervalo de Confiança para Aplicativos em Teste**

| | Desvio Padrão/Intervalo de Confiança – ENS (*) | Desvio Padrão/Intervalo de Confiança – IRSO (*) | Desvio Padrão/Intervalo de Confiança – IRCOR (*) | Desvio Padrão/Intervalo de Confiança – IRCOF (*) | Desvio Padrão/Intervalo de Confiança – ENC (*) |
|---|---|---|---|---|---|
| **P1** (^) | 0.22/ [4.21, 4.46] | 0.31/ [4.96, 5.31] | 0.25/ [4.40, 4.69] | 0.28/ [4.80, 5.13] | 0.45/ [9.33, 9.86] |
| **P2** (^) | 0.35/ [7.37, 7.78] | 0.38/ [8.68, 9.13] | 0.36/ [7.74, 8.15] | 0.35/ [8.41, 8.82] | 0.56/ [16.25, 16.90] |
| **P3** (^) | 0.16/ [1.04, 1.23] | 0.21/ [1.23, 1.48] | 0.19/ [1.06, 1.29] | 0.20/ [1.16, 1.39] | 0.41/ [2.29, 2.76] |
| **P4** (^) | 0.19/ [1.41, 1.64] | 0.23/ [1.68, 1.95] | 0.22/ [1.49, 1.74] | 0.21/ [1.61, 1.86] | 0.49/ [3.11, 3.68] |

(*) ENS = Execução Nativa em Servidor WSE-OS
IRSO = Instanciação Remota Sem Otimização
IRCOR = Instanciação Remota Com Otimização em RAM
ENC = Execução Nativa em Cliente WSE-OS
IRCOF = Instanciação Remota Com Otimização em *Flash*

(^) P1 = *Easy Media Creator 10IJ*
P2 = *FrostWire 4.18.3*
P3 = *Google Chrome 8.0*
P4 = *Microsoft Word 2007*

A partir desse experimento é possível avaliar este nível de degradação no desempenho de cada um dos cenários. Observa-se na Figura 37, que a inicialização de um aplicativo no *middleware* sem otimização tem em média uma porcentagem de degradação de 18% comparado em execução nativa servidor. Com ativação de recursos em memória *Flash Driver* USB, esta degradação fica na faixa dos 12% a 14%. Por possuir maior velocidade de acesso aos dados armazenados, o *cache* em memória RAM possui uma menor degradação no tempo de inicialização destes aplicativos, com rendimento até 16% superior ao mediador não otimizado para *caches*. Estas porcentagens aproximadas são ilustradas pela Figura 37.

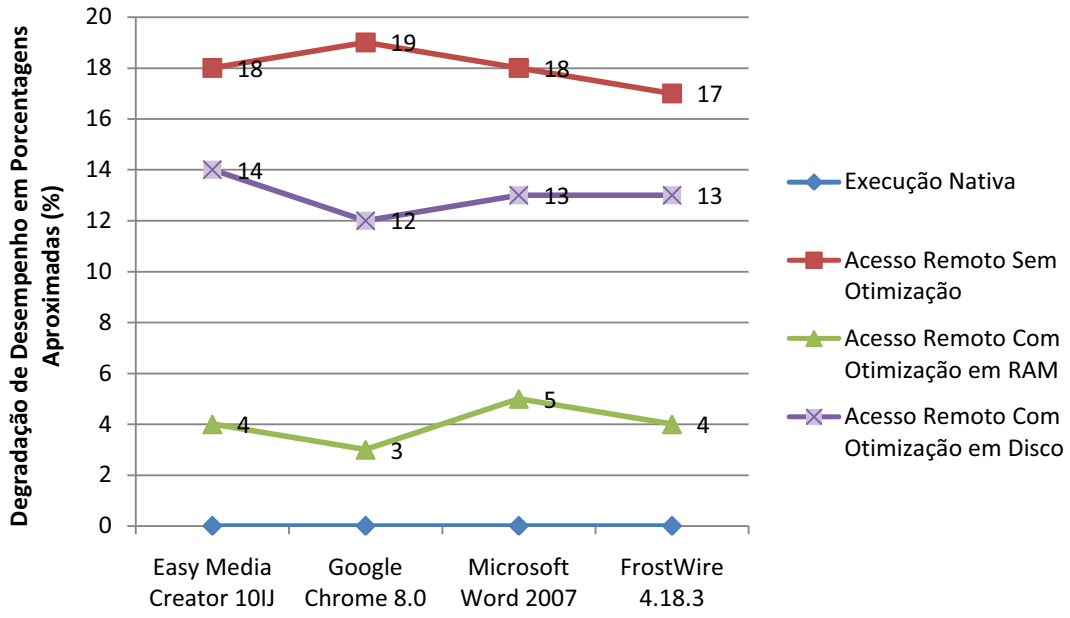


**Figura 37 - Porcentagem aproximada de degradação em função da otimização no middleware**

A importância da utilização deste sistema em memória *caches* se torna mais evidente quando o número de *threads* no servidor aumenta devido a diversas solicitações simultâneas de clientes WSE-OS. Neste caso, o gerenciamento de *cache* em cada *middleware* WSE-OS tem grande influencia sobre a velocidade de inicialização de atualização de interface gráfica cliente. Considerando as configurações de *hardware* cliente03 da Tabela 3, a Tabela 7 aponta diferentes degradações referente ao número de clientes conectados simultaneamente ao servidor WSE-OS para a inicialização do aplicativo *FrostWire* 4.18.3. No pior cenário, uma diferença de 28.63 segundos para a instanciação do aplicativo em um *middleware* que possui configuração de *cache* em memória RAM.

As alocações de *caches* contribuem para que não ocorra um gargalo de processamento para criação e distribuições de pacotes do servidor ao cliente WSE-OS. Com os dados já armazenados em *cache*, o cliente WSE-OS apenas aguarda instruções para atualização de interface gráfica utilizando arquivos já em memória, ficando, deste modo, com parcela majoritária de degradação em função da execução concorrente de diversos *software* virtualizados no servidor.

**Tabela 7 - Tempo de Boot em segundos para instanciação da aplicação FrostWire 4.18.3**

| CENÁRIOS | ***Middleware* WSE-OS Sem Otimização de *Cache* / (*) DP** | **Intervalo de Confiança *Middleware* Sem Otimização (*Student* – 90%)** | ***Middleware* WSE-OS Com Alocação de 16MB RAM / (*) DP** | **Intervalo de Confiança *Middleware* Com Otimização RAM (*Student* – 90%)** | ***Middleware* WSE-OS Com Alocação de 64MB *Flash* / (*) DP** | **Intervalo de Confiança *Middleware* Com Otimização *Flash* (*Student* – 90%)** |
|---|---|---|---|---|---|---|
| **01 CA (^)** | 8.91/0.12 | [8.84, 8.97] | 7.95/0.11 | [7.88, 8.01] | 8.62/0.09 | [8.56, 8.67] |
| **05 CA (^)** | 9.92/0.18 | [09.81, 10.02] | 8.12/0.14 | [8.03, 8.20] | 9.48/0.16 | [9.38, 9.57] |
| **10 CA (^)** | 15.80/0.23 | [15.66, 15.93] | 13.16/0.19 | [13.04, 13.27] | 14.52/0.21 | [14.39, 14.64] |
| **20 CA (^)** | 23.82/0.19 | [23.70, 23.93] | 19.58/0.27 | [19.42, 19.73] | 20.88/0.24 | [20.74, 21.01] |
| **30 CA (^)** | 61.45/0.33 | [61.25, 61.64] | 49.48/0.39 | [49.25, 49,70] | 50.84/0.42 | [50.59, 51.08] |
| **40 CA (^)** | 132.03/0.52 | [131.72, 132.33] | 103.40/ 0.48 | [103.12,103.67] | 104.76/ 0.54 | [104.44,105.07] |

(*) DP = Desvio Padrão; (^) CA = Conexões Ativas

Como o ambiente é baseado na comunicação *request-reply,* os eventos disparados pelo cliente são encaminhados para o servidor WSE-OS. Embora textos, componentes gráficos, arquivos de áudio e vídeo estejam memória *cache*, estes só serão processados pelo Módulo de Comunicação Cliente após receber *reply* do servidor WSE-OS notificando que a informação a ser utilizada já está em *cache* cliente. Por este motivo, o servidor WSE-OS mantém em seu Módulo de Comunicação Servidor um sistema de *cache* sincronizado com sistema de *cache* cliente.

Isto faz com que, ao invés de transmissão de pacotes com componentes gráficos, arquivos de áudio e vídeo, sejam concedidos a transmissão de pequenos pacotes apenas com instruções para utilização dos arquivos em *cache* cliente. Nesta situação, se um arquivo de vídeo de 100MB já está em memória *cache* cliente, o evento disparado pelo cliente para a abertura deste mesmo vídeo implicará em uma resposta de poucos *kbytes* pelo servidor, situação oposta caso a verificação do Módulo de Comunicação Servidor não encontrasse correspondência deste evento em *cache* servidor, transmitindo para a rede de comunicação sem fio um arquivo de 100MB.

Diante disto, os dados da Tabela 7 mostram que a utilização de *cache* em memória *Flash Driver USB* oferece uma redução média de aproximadamente 10.9% no tempo de inicialização do aplicativo, enquanto que o *cache* em memória RAM atinge uma redução média aproximada de 17.42%. Como já esclarecido anteriormente, estes ganhos decorrem em função da liberação de banda devido à redução de pacotes transmitidos, diminuição de processamento cliente pelo fato destes dados não necessitarem de processo de desencriptação, descompressão e verificação de integridade e eliminação de criação de pacotes onerosos pelo servidor do sistema. É importante levar em consideração que estes valores para *caches* primários (RAM) e secundários (*Flash*) são válidos a partir da segunda instanciação do aplicativo. Logo na primeira instanciação de aplicativos, os *caches* irão armazenar todas as solicitações e respectivas respostas referentes ao aplicativo executado. Portanto, o desempenho na primeira chamada de execução de um aplicativo se equivale ao de um *middleware* sem otimização em *cache* RAM ou *Flash*. A Figura 38 ilustra estas situações em função do número de *boot* do *middleware* e instanciações remotas para o aplicativo *Google Chrome 8.0* em configuração de *hardware* cliente03 da Tabela3.

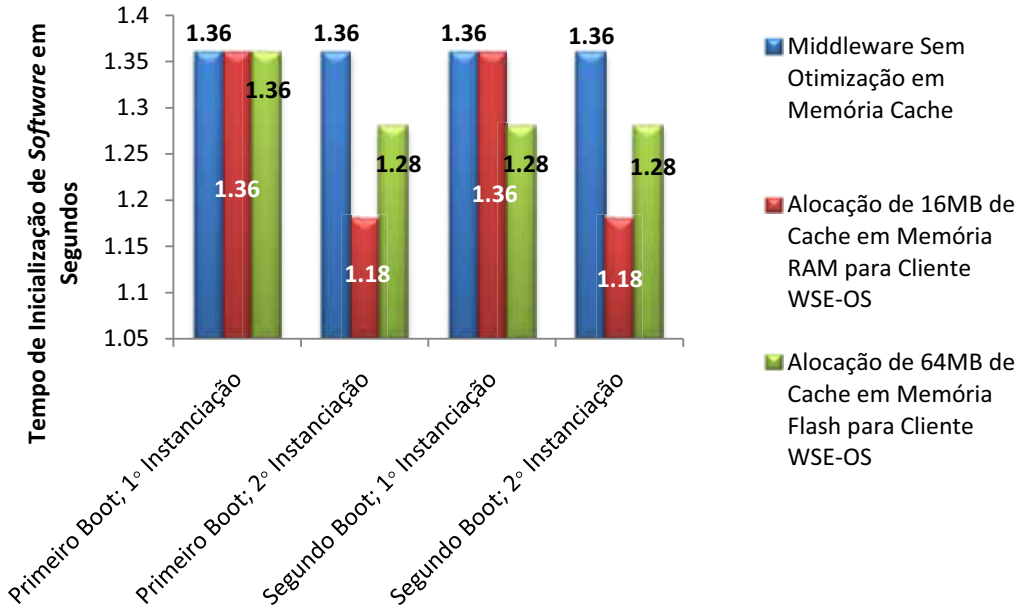


**Figura 38 - Funcionamento de *caches* em função do número de boot e instanciações de software**

Nota-se que o desempenho no primeiro processo de *boot* para a primeira instanciação do aplicativo é igual para todos os cenários. Por outro lado, já na segunda instanciação o desempenho é superior nos casos onde há otimizações de *caches*. Pelo fato da memória RAM ser volátil, após o desligamento do terminal cliente todas as informações retidas em *cache* primário serão apagadas. Ao contrário disto, o *cache* em memória secundária se mantém persistente para os próximos processos de *boot*, tanto para a instanciação remota de um SO quanto para aplicativos já invocados. Por este motivo, a Figura 38 ilustra o processo de segundo *boot* e primeira instanciação com melhor desempenho para otimização de *cache* em memória *Flash Driver* USB. A partir da segunda instanciação, o uso de *cache* em memória RAM tem desempenho 8% superior ao *cache* em memória *Flash Driver* USB.

## 6.5 Avaliação de Escalabilidade WSE-OS

Embora a análise de desempenho da escalabilidade do ambiente proposto seja parte integrante das considerações sobre a Camada de Gerenciamento WSE-OS vistas em (DIGIERE, 2011), é importante para este trabalho conhecer o ponto limiar de saturação de virtualização concorrente aplicada a uma mesma

imagem de sistema operacional sobre *hardware* servidor e baseando a execução do *middleware* WSE-OS em um cliente modelo. Para isto, foram criados diferentes cenários para a medição do tempo de inicialização da máquina virtual servidora para a instanciação do SO de referência no qual os resultados apresentados são de medições do cliente03 descrito na Tabela 3. A Tabela 8 ilustra os valores do tempo de *boot* levando em consideração diferentes quantidades de máquinas com solicitações remotas para virtualização da imagem de referência através da comunicação TCSC.

O desempenho para configuração de servidor utilizado no teste é plausível para até quarenta máquinas concorrentes em suas solicitações de inicialização de sistema operacional virtualizado. No cenário de quarenta e cinco máquinas concorrentes para a mesma solicitação de imagem, a degradação ultrapassa qualquer padrão observado nos outros casos. Isto é previsto e ocorre pelo fato da maquina virtual trabalhar com até 50% da memória principal de *hardware* servidor para a virtualização, onde os outros 50% da memória é reservado para o Sistema Operacional Hospedeiro do Servidor WSE-OS (ORACLE, 2011). Com o limiar ultrapassado, o desempenho desta execução torna relativamente inviável para o uso corporativo e para estações de trabalhos que exigem rapidez na inicialização de sistemas.

**Tabela 8 - Tempo de Boot de SO para Solicitações Concorrentes (em Segundos)**

| Cenários | *Middleware* WSE-OS Sem Otimização em *Cache* / (*) DP | Intervalo de Confiança (*Student* – 90%) |
|---|---|---|
| **01 Conexão Ativa** | 19.18 / 0.25 | [19.04, 19.33] |
| **05 Conexões Ativas** | 23.27 / 0.28 | [23.11, 23.43] |
| **10 Conexões Ativas** | 26.95 / 0.31 | [26.77, 27.13] |
| **15 Conexões Ativas** | 30.46 / 0.36 | [30.25, 30.67] |
| **20 Conexões Ativas** | 37.55 / 0.41 | [37.31, 37.79] |
| **25 Conexões Ativas** | 55.91 / 0.47 | [55.64, 56.18] |
| **30 Conexões Ativas** | 167.00 / 0.57 | [166.67, 167.33] |
| **35 Conexões Ativas** | 201.53 / 1.23 | [200.82, 202.24] |
| **40 Conexões Ativas** | 357.82 / 1.45 | [356.98, 358.66] |
| **45 Conexões Ativas** | 1129.91 / 12.34 | [1122.76, 1137.06] |

(*) DP = Desvio Padrão

Para ilustrar esta situação limite, a Figura 39 fornece um gráfico, referente aos dados da Tabela 8, que correlaciona a degradação do tempo de *boot* de SOs remotamente virtualizados em função do uso de memória RAM para instanciação daqueles SOs em Servidor WSE-OS. Como previsto por (ORACLE, 2011), é possível notar que a alocação para máquinas virtuais acima de 50% da memória total disponível em Servidor WSE-OS reflete em uma degradação no tempo de inicialização de SO que inviabiliza o uso.

De fato, a memória RAM em Servidor WSE-OS é o principal gargalo para o escalonamento de clientes no ambiente. No caso de replicação do ambiente em um servidor de 32GB, o número de clientes conectados concorrentemente aumentaria desde que a memória em Servidor WSE-OS alocada para cada máquina virtual (MV) mantivesse o padrão estabelecido nos testes deste capítulo (192MB para MV). Nesta situação, haveria 16GB para a alocação de máquinas virtuais e instanciação de SOs. É compreensível que outros parâmetros devem ser considerados como pontos limitantes de escalabilidade. Em um cenário hipotético com 1000 clientes, o processamento Servidor WSE-OS associado com o BPAC (Base de Ponto de Acesso Central – Roteador WSE-OS) seriam outros fatores limitantes a ser considerados.

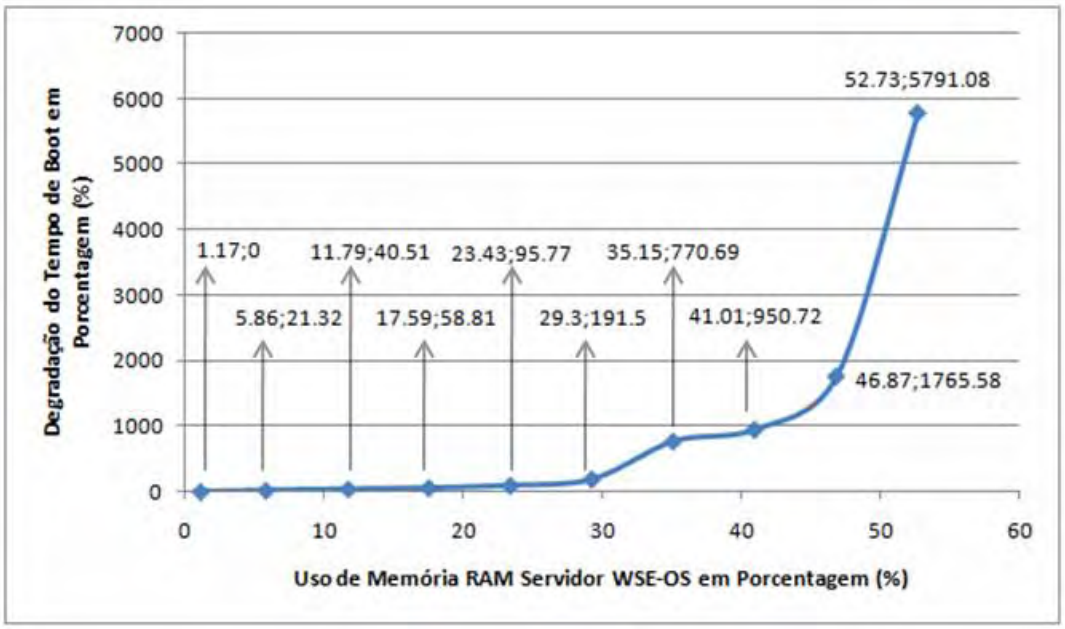


**Figura 39 - Degradação no Tempo de Boot em função de Memória utilizada em Servidor 16GB**

Para ilustrar o desempenho do ambiente WSE-OS em configurações distintas de servidores, a Figura 40 segue a mesma linha de desenvolvimento da Figura 39 para indicar a degradação no tempo de *boot* do SO de referência em função da memória RAM utilizada em mesmo conjunto de *hardware* do teste anterior, mas com apenas 4GB de memória principal em Servidor WSE-OS. Nestas condições, e considerando a alocação de 192MB para cada máquina virtual, a escalabilidade do ambiente permanece atrelada aos 50% de alocação da memória principal servidor para virtualização. Diante disto, a escalabilidade nesse cenário cai para dez conexões ativas, ou seja, 1920MB alocados por MVs em um total de 2048MB disponíveis para virtualização. Por outro lado, observa-se que o comportamento da degradação do tempo de *boot* em função da porcentagem de memória RAM utilizada em Servidor WSE-OS segue o mesmo padrão apresentado pela Figura 39. É importante notar que as essas porcentagens de degradações no tempo de *boot* do SO de referência entre os servidores de 16GB e 4GB são semelhantes e visivelmente compatíveis.

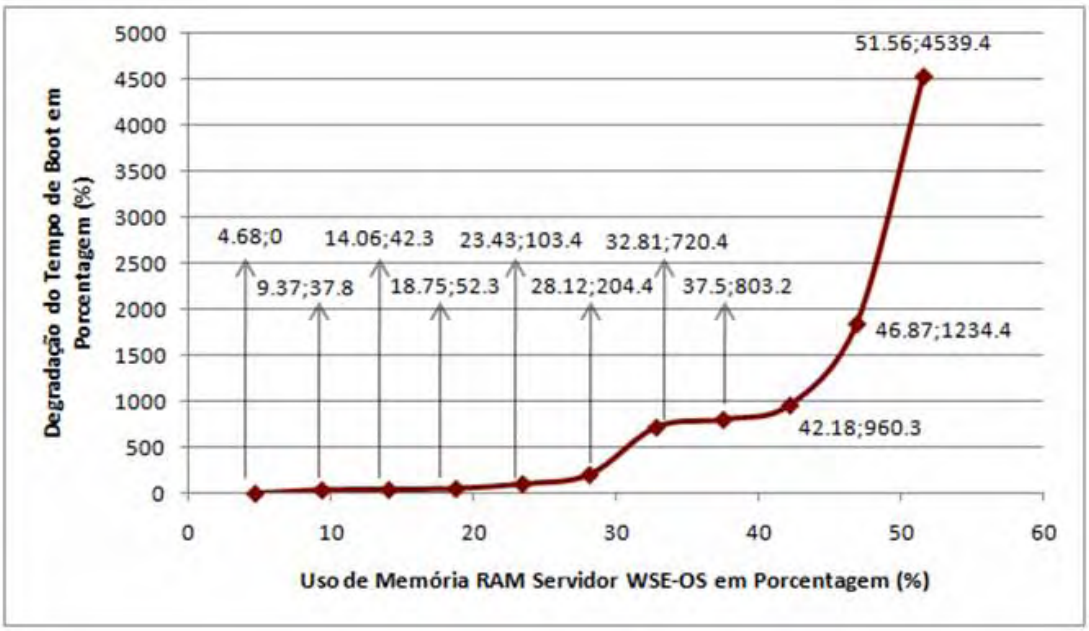


**Figura 40 - Degradação no Tempo de Boot em função de Memória utilizada em Servidor 04GB**

Tanto a Tabela 8 quando a Tabela 7 ilustrada na seção 6.4.3 indicam o tempo de *boot* concorrente de uma mesma imagem de SO e aplicativo como referência para visualização da degradação causada pelo processamento de solicitações simultâneas. Esta situação é a mais crítica possível em relação ao

processamento servidor e uma das mais onerosas para a rede de comunicação sem fio do ambiente, pois são nestes momentos que ocorrem o carregamento em memória servidor de diversas imagens virtualizadas de SO/aplicativos e distribuição de pacotes para diferentes clientes do ambiente.

Por outro lado, a dinâmica de funcionamento para clientes WSE-OS logo após a fase de carregamento em memória servidor de SO/aplicativos virtualizados remotamente, não demonstra atrasos significativos para as solicitações mesmo que sejam simultâneas de vários clientes. É compreensível que programas que exigem maior alocação de memória e processamento em servidor como, por exemplo, programas editores de vídeos e imagens, apresentarão maior latência de resposta. Casos contrários a esta situação e pelo fato dos aplicativos/SO já estarem em memória RAM servidor, o processamento de requisição cliente para os *software* em uso apresenta comportamento próximos ao de uma execução nativa.

## 6.6 Considerações Finais

Os experimentos realizados nortearam configurações adequadas para que o *middleware* WSE-OS trabalhe em um ambiente de compartilhamento sem fio em sincronicidade com o servidor WSE-OS. De acordo com as atividades de testes desenvolvidos neste projeto, é possível afirmar que o uso de *cache* em memória RAM otimizou o desempenho de operações *request-reply* para aplicativos em média de 17.42% em ambiente com várias conexões remotas concorrentes. Por outro lado, o uso de *cache* em memória *Flash* garante que, após o primeiro *boot* do sistema WSE-OS, o processo de inicialização do sistema e carregamento de um SO tenha um rendimento 7.16% superior.

O desempenho do Sistema de Transmissão Diferencial de Dados pode influenciar de forma significativa o potencial de escala de execução do ambiente. O acréscimo no tempo de *boot* dos aplicativos devido ao número de computadores clientes incorporados ao ambiente, pode ser subtraído pelo uso das otimizações locais de cada mediador. Isto favorece o desempenho global do ambiente, tornando o processo de inicialização de aplicativos próximos aos valores de execução nativa, além de permitir a ampliação do número de clientes sem que haja um aumento excessivo e oneroso de transmissão de pacotes na rede referentes às suas solicitações. Os experimentos constatam que o modelo TCSC com acesso remoto e sistema de virtualização processe a inicialização de uma imagem de SO, já somado o

tempo de *boot* do *middleware* e com otimização em memória *Flash Driver* USB, com uma latência adicional de apenas 5.82 segundos comparado a uma inicialização nativa em *hardware* de mesmo cliente. Um acréscimo que representa menos de 12.2% de degradação em inicialização pelo sistema WSE-OS.

A proposta de modularização do *middleware* torna viável a possibilidade de, em um trabalho futuro, efetuar modificações em qualquer um dos módulos que o compõe. Isto faz com que a adaptação às novas necessidades tecnológicas ou diferentes ambientes de execução seja realizada de modo prático sem afetar o sistema mediador por inteiro. Todos estes dados comprovam a viabilidade do uso do *middleware* WSE-OS como parte integrante de um ambiente de compartilhamento sem fio para sistemas operacionais e propõe o uso do sistema WSE-OS como ferramenta de gerenciamento de computadores baseado em rede de comunicação sem fio, permitindo administração centralizada, além de prontificar instalação e manutenção de aplicativos distribuídos.

# 7. CONCLUSÃO

## 7.1 Conclusões e Discussões

A associação entre tecnologias concebidas para o aceleramento de transações de conexões X *Window System 11,* técnicas de virtualização de *hardware* e acesso remoto seguro permite a construção de uma arquitetura distribuída como base de um sistema de gerenciamento de ambientes computacionais capaz de replicar imagens de sistemas operacionais em um ambiente de comunicação sem fio. Ao contrário do que ocorre com soluções de gerenciamento centralizado com enlaces estruturados sobre conexões físicas, o WSE-OS oferece um ambiente sem penalização da mobilidade e ubiqüidade do poder computacional.

O obstáculo, entretanto, é dispor estes benefícios em um *middleware* otimizado trabalhando em sincronicidade com uma camada de gerenciamento servidor capazes de oferecer escalabilidade, flexibilidade e facilidade de instalação e manutenção de aplicativos distribuídos para arquiteturas de computadores de baixo custo.

Através de técnicas de virtualização e, com auxílio de virtualização assistida por *hardware*, o modelo possibilita disseminar a execução de dados relativos a imagens de sistemas operacionais em computadores clientes através de uma única configuração de *software* sobre plataforma servidor baseando em comunicação TCSC. Desta forma, além de atender instanciações remotas de clientes com um decréscimo aceitável, se comparado ao desempenho de execução nativa servidor, o modelo faz uso da capacidade ociosa de processamento de servidor alocando individualmente administrações e execuções de aplicativos de diferentes clientes WSE-OS.

As técnicas de gerenciamento de memória *cache* e a implementação de um sistema de compressão otimizado para tipos de dados aplicados ao *middleware* WSE-OS contribuem para que não ocorra um

gargalo de processamento no servidor WSE-OS e distribuição de pacotes pela Base de Ponto de Acesso Central devido ao aumento de *threads* por solicitações simultâneas de clientes ao servidor, permitindo ampliação do número de clientes sem que haja um aumento excessivo e oneroso de transmissão de pacotes mesmo quando o ponto de saturação da escalabilidade do ambiente esteja comprometido.

O estudo indica que uma solução como o WSE-OS, baseada em replicação de sistemas operacionais sobre um enlace de comunicação sem fio, é capaz de oferecer um *middleware* associado a uma camada de gerenciamento servidor como base de um sistema de gestão para família de processadores x86 por meio da comunicação entre objetos distribuídos.

Por ser um sistema baseado em infraestruturas virtuais de *desktops* (VDI), esta construção outorga benefícios de manipulação remota para dados estruturados sem que o enlace de comunicação sem fio seja degradado. Os testes realizados sobre o ambiente do projeto revelam que a degradação de desempenho para a execução do sistema e instanciação de um SO de referencia não ultrapassa 12.2% do tempo de execução nativa em máquina cliente para uma conexão ativa e, para casos de instanciação de *software*, um ganho médio de 52.58% em tempo de inicialização.

Embora introduza certa degradação em função do sistema de virtualização servidor, o WSE-OS é uma opção adequada e viável para a replicação de sistemas operacionais sobre enlaces de comunicação sem fio através do modelo de gerenciamento centralizado de modo a atender ambientes com configuração heterogênea de *hardware.*

Com a evolução do poder computacional, tanto para máquinas clientes quanto para servidores, associados com a redução de custos de componentes de hardware indicam a tendência para que a degradação imposta por estes limitantes tenha diminuição ao longo do desenvolvimento e pesquisa. Isto sugere a possibilidade do modelo de gerenciamento através da virtualização centralizada e do *middleware* WSE-OS apresentarem, em sincronicidade, todos os benefícios do estudo com eficiência ainda mais próxima do desempenho nativo de um computador.

## 7.2 Contribuições deste Projeto

O desenvolvimento do WSE-OS trouxe diversas contribuições para estudos acadêmicos e implantação como ferramenta de gerenciamento centralizado sobre redes de comunicações sem fio. Dentre estes pontos, pode-se citar:

- Um estudo bibliográfico sobre tecnologias e arquiteturas de sistemas baseados na comunicação entre objetos distribuídos para o desenvolvimento de aplicativos distribuídos com extensão de suporte ao paradigma *publish-subscriber;*
- Proposta de um *middleware* modular baseado em *kernel* Linux para gerenciamento de recursos locais, sistema de acesso remoto seguro e tecnologias concebidas para o aceleramento de transações de conexões *X Window 11* através de um Sistema de Transmissão Diferencial de Dados;
- Análise comparativa de desempenho entre execuções nativas e em *middleware* WSE-OS para sistemas operacionais e aplicativos. Análise de comportamento de diferentes configurações de memória *cache* sobre instanciação e *boot* de *software.*

## 7.3 Trabalhos Futuros

Com o indicativo de trabalhos futuros a partir do sistema WSE-OS são citados:

- Extrapolação de execução local do sistema WSE-OS para execução em redes de maior alcance como, por exemplo, padrão IEEE 802.16 WiMAX, e através da Internet, permitindo acesso ao usuários do sistema a partir de qualquer computador como cliente WSE-OS;
- Extensão e adaptação de *middleware* para execução em dispositivos móveis como, por exemplo, *Smartphones, Tablet Pcs, PDAs,* Celulares, permitindo que usuários tenham acesso a um sistema operacional remotamente virtualizado;
- Aprimoramento no mecanismo de acesso a dispositivos removíveis e implementação de uma interface gráfica de execução automática para que os usuários tenham uma interação transparente e direta com estes dispositivos, evitando o processo manual de navegação até as pastas compartilhadas entre cliente e servidor nas quais contém os arquivos do dispositivo removível;

- Criação de um mecanismo estratégico para distribuições de mídias contínuas sob demanda em enlace de comunicação sem fio, para que a utilização remota de multimídia tenha eficiência em desempenho;
- Replicação de Servidores WSE-OS e múltiplos pontos de acessos (BPAC) para aumentar escalabilidade do ambiente e desempenho de inicialização de aplicativos e sistemas operacionais associado com a implementação de um sistema automatizado de distribuição e equilíbrio de cargas responsável pela migração de MVs em servidores ociosos;
- Explorar possibilidades de armazenamento do *Middleware* WSE-OS nos clientes do ambiente sem a necessidade de uso de *boot* pela porta USB (*Universal Serial Bus*), pré estabelecendo o sistema mediador em periféricos de *hardware* como, por exemplo, placa de rede sem fio.

# BIBLIOGRAFIA

ARDENCE. **Ardence *Software* Streaming Platform: *Desktop* Edition**. 2007. Disponível em: < http://www.intervalzero.com>. Acesso em: Maio 2010.

BACON, J. **Distributed event-based systems.** Proceedings of second international conference on Distributed event-based systems. 2008. New York, USA, p. 383–402.

BARATTO, A. R.; KIM, N. L.; NIEH, J. **THINC: A Virtual Display Architecture for *Thin-client* Computing.** ACM Symposium on Operating Systems Principles. 2005. Brighton, United Kingdom, p. 277 - 290.

BOOK, M., GRUHN, V. **An Instant Message-Driven User Interface Framework for Thin Client Applications.** Internacional Conference on Automated Software Engineering. 2006. Tokyo, Japan. P.257-260.

CHERVENAK, A.; FOSTER, I.; KESSELMAN, C.; SALISBURRY, C.; TUECKE, S. **The data *Grid*: Towards an architecture for the distributed management and analisys of large scientific datasets**. Journal of Network and Computer Applications. 2000. Academic Press, Vol. 23, Issue 3, p. 187 -200.

**CITRIX.** 2008**. Provisioning Server Desktop.** Disponível em: < http://www.citrix.com.br/products/provisioning_server_*desktop*s.php > Acesso em: Março 2008.

COOPERSMITH, A. **X.Org Home.** 2010. Disponível em: <http://www.x.org/wiki/Home>. Acesso em: Abril 2010.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Sistemas Distribuídos: Conceitos e Projeto**. São Paulo: Bookman Editora, 2007. 784p.

CREPALDI, L. G.; DIGIERE, A. R. **A Management Tool for the Replication of Operating Systems in Wireless Communication Networks.** 35th Annual IEEE Computer Software and Applications Conference. COMPSAC'11: The Computed World: Software Beyond the Digital Society. 2011. Munich, Germany. A ser publicado.

CRUZ, D. **FLEXLAB: *Middleware* de virtualização de *hardware* para gerenciamento centralizado de computadores em rede.** 2008. Monografia de Mestrado – Universidade Estadual Paulista "Julio de Mesquita Filho". UNESP/Bauru.

COFFMAN, D. **A client-server architecture for state-dependent dynamic visualizations on the web.** Proceedings of the 19th internacional conference on World wide web. 2010. New York, USA, p. 458-471.

DARTWARE. **InterMapper Flows.** 2011. Disponível em: < http://www.intermapper.com/products/intermapper-flows/features > Acesso em: 05 dez. 2010

DICK, K. **Distributed objects and the Web**. Object Magazine, Englewood, 28 jun 1997. V.15, n.4, p.19.

DEUTSH, P.; GAILLY, J. **ZLIB Compressed Data Format Specification version 3.3.** RFC 1950. Maio 1996.

DIGIERE, A. R. **Camada de Gerenciamento para Comunicação entre Computadores baseada em Redes Sem Fio (WSE-OS)**. 2011. Monografia de Mestrado - Universidade Estadual Paulista "Julio de Mesquita Filho", UNESP/Bauru (a ser publicado).

FARHAT, C.; ELIAN, S. **Estatística Básica.** São Paulo: LCTE Editora, 2008. 240p.

GALLAUGHER, John. **The Critical Choice of Client Server Architecture: A Comparison of Two and Three Tier Systems.** 1996. Disponível em: <http://www2.bc.edu/~gallaugh/ research/ism95/cccsa.html>. Acesso em: 13 abr. 2010.

GOOGLE. **Google Chrome 8.0: Get a Fast, Free Web Browser.** 2011. Disponível em: < http://www.google.com/chrome?hl=en> Acesso em: Fevereiro 2011.

GRAUMANN, D., LARA, W., HIGHTOWER, J. e BORRIELLO, G. **Real-world implementation of the Location Stack: The Universal Location Framework**. *Proceedings of the 5th IEEE Workshop on Mobile Computing Systems & Applications*, Monterey. 2003. CA, EUA, Outubro, pgs 122-128.

GRIMM, R.; SIRER, E.; BERSHAD, B.; GREGORY, A.; MCDIRMID, **S. Distributed Virtual Machines: A System Architecture for Network Computing**. Dept. of Computer Science & Engineering University of Washington. Seattle, USA. 2001.

HALETKY, E. **VMware vSphere and Virtual Infrastructure Security: Securing the Virtual Environment.** San Francisco: Prentice Hall PTR, 2009. 552 p.

HAN, Li.; Jyri, S. **Research on Context-Aware Mobile Computing.** 22nd Conference on Advanced Information Networking and Applications. AINAW. 2008. Okinawa. P. 24-30.

HENDERSON, T., KOTZ, D. **The Changing Usage of a Mature Campus-Wide Wireless Network.** ACM Computer Networks: The International Journal of Computer and Telecommunications Networking. 2008. New York, USA. Vol. 52, Issue 14, p. 2690 – 2712.

HENRY, M. **PXE Manageability Technology for EFI**. Intel Developer Update Magazine, Intel. Outubro de 2000, p. 3. Disponível em http://www.intel.org/technology/magazine/systems/it10004.pdf. Acesso em 30 de Março de 2010.

INTEL. **Pre*boot* Execution Environmet (PXE) Specification Version 2.1**. 2010. Disponível em http://www.pix.net/*software*/pxe*boot*/archive/pxespec.pdf. Último acesso em 04 de Maio de 2010.

JRMI. **Java Remote Method Invocation.** Sun Microsystems. 2010. Disponível em: <http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp > Acesso em 26 abr. 2010.

JONES, M. T. **Emulação do Sistema com o QEMU**. 2007. Disponível em: <http://www.ibm.com/developerworks/linux/library/l-linuxvirt/?ca=dgr-lnxw01>. Acesso em: mar. 2010.

JONSSON, J.; KALISKI, B. **Public-key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications**. **Version 2.1.** RFC 3447. Fevereiro de 2003.

KIVINEN, T.; KOJO, M. **More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE).** RFC 3526. Maio 2003.

KLEINROCK, L. **Nomadicity: anytime, anywhere in a disconnected world.** 1997. *Mobile Networks and Applications*, Vol. 1, No. 4, pgs 351-357.

KRAWCZYK, H.; BELLARE, M.; CANETTI, R. **HMAC: Keyed-Hashing for Message Authentication.** RFC 2104, February 1997.

KSHEMKALYANI, A.; SINGHAL, M. **Distributed Computing: Principles, Algorithms and Systems.** New York: Cambridge University Press, 2008. 756p.

LAI, A., NIEH, J. **On the Performance of Wide-Area Thin-Client Computing.** ACM Transactions on Computer Systems (TOCS). 2006. New York, USA. Vol. 24, Issue 2, p.175 – 209.

LEINER, B., CERF, V., CLARK, D., KAHN, R. **A brief history of the internet.** ACM SIGCOMM Computer Communication Review. 2009. Academic Press, Vol. 39, Issue 5, p. 22-31.

LOBO, E. **Uma solução do Problema de Horário Escolar Via Algoritmo Genético Paralelo.** 2005. Dissertação de Doutorado em Modelagem Matemática e Computacional. CEFET-Minas Gerais. Disponível em: < http://www.mmc.cefetmg.br/info/downloads/D006-EduardoLuizMirandaLobo2005.pdf > Acesso em: 26 abr. 2010.

LULIAN, U.; LOAN, T.; CORNELIU, S.; ION, C.; **Analyse of the MPEG-4 Compressed Streams.** World Scientific and Engineering Academy and Society (WSEAS). Transactions on Communications. 2009. Wisconsin, USA. Vol. 8. Issue 9. P. 1002-1011.

MASUDA, H., MURATA, K. **Low TCO and high-speed network infrastruture with virtual technology.** ACM SIGUCCS fall conference on User services conference. 2009. New York, USA. Academic Press, Vol. 28, Issue 3, p. 321-324.

MENDONÇA, P. **Uma Proposta de Co-Escalonamento Adaptativo para um Ambiente de Computação Oportunista de Recursos Distribuídos.** 2008. Dissertação de Mestrado em Ciência da Computação UFSC. Florianopolis, Santa Catarina. Disponível em < http://www.tede.ufsc.br/teses/PGCC0837-D.pdf > Acesso em 26 abr. 2010.

MEYER, U.; WETZEL S. **A man-in-the middle attack.** Workshop on Wireless Security. Proceedingd of the ACM workshop on Wireless security. 2009. Philadelphia, USA. Pages: 90 – 97.

MICROSOFT. **Word 2007: For all your Revolutionary Ideas.** 2007. Disponível em: < http://office.microsoft.com/en-us/word/> Acesso em: Fevereiro 2011.

MOLINARI, E. **Padrões de Projeto para Desenvolver Aplicações Distribuídas com CORBA e JAVA.** 2002. Dissertação de Mestrado em Ciência da Computação UFSC. Florianopolis, Santa Catarina. Disponível em: < http://www.tede.ufsc.br/teses/PGCC0316.pdf > Acesso em 26 abr. 2010.

MORGADO, E.; CRUZ, D. I.; TWANI, E.; **Paving the Way for a Dynamic and Mature ICT Infrastructure in Education: A Case for Schools in Emerging Markets.** Proceedings of the International Conference on Engineering and Technology Education – INTERTECH. 2008. Santos, Sao Paulo. 2008, p. 79.

MORIMOTO, C. E. **Servidores Linux: Guia Prático**. Porto Alegre: Sul Editores, 2008. 735 p.

NARAYAN, S., ARDHAM, S. **Impacto f Wireless IEEE802.11n Encryption Methods on Network Performance of Operating Systems.** 2nd International Conference on Emerging Trends in Engineering and Technology (ICETET). Nagpur, India 2010. p. 1178 – 1183.

NAKANISHI, T. **Análise de desempenho de mecanismos de controle de concorrência em banco de dados**. Rio de Janeiro. Tese (Doutorado em Computação) - Pontifícia Universidade Católica do Rio de Janeiro (PUC/RIO), 1981.

**NISTI.** Digital Signature Standard (DSS). Federal Information Processing Standards Publication. FIPS PUB 186-3. Junho 2009.

**NISTI.** Data Encryption Standard (DES). Federal Information Processing Standards Publication. FIPS PUB 46-3. Outubro 2008.

**ORACLE CORPORATION**. Oracle VM Virtualbox, User Manual. 2011. Disponível em: <http://dlc.sun.com.edgesuite.net/virtualbox/4.0.2/UserManual.pdf>. Acesso em: fev. 2011.

PARDALOS, P. e Mastorakis, M. **The use of MIMO technologies in wireless communication networks.** CIT'09 Proceedings of the 3rd International Conference on Communications and information technology. 2009. Wisconsin, USA. p. 139-145.

PATRICK, M., DROMS, R. **Dynamic Host Configuration Protocol Relay Agent Information Option.** RFC 2132. Abril 2001.

PONNEKANTI, S. e FOX, A. **Interoperability among Independetly Evolving Web Services.** IFIP 5th ACM/IFIP/USENIX International Middleware Conference. 2004. Toronto, Canadá. Vo.1 p. 331-351.

RANI, B.; Bansal, R.K.; Bansal, S. **Comparison of JPEG and SPIHT image compression algorithms using objective quality measures.** Multimedia, Signal Processing and Communication Technologies. 2009. IMPACT'09. p. 90-93.

REGIS, S. **Getting Started with NX.** 2007. Disponível em: < http://www.nomachine.com/documents/getting-started.php >. Acesso em: Outubro 2009.

ROSE, L. Anonimato e Impunidade. **Liberdade e Censura: A ética da Aética da Internet**. Sociedade Brasileira de Estudos Interdisciplinares da Comunicação, Setembro de 2007. Disponível em http://www.adtevento.com.br/INTERCOM/2007/resumos/R0211-1.pdf. Acesso em 07 de Abril de 2010.

ROXIO. **Easy Media Creator 10 Deluxe Suite.** 2010. Disponível em: < http://www.roxio.com/enu/products/creator/deluxe/overview.html > Acesso em: Fevereiro 2011.

ROY, N., Das, S. **Enhancing Availability of Grid Computacional Services to Ubiquitous Computing Applications.** IEE transactions on Parallel and Distributed Systems. 2009. Austin, USA. Vol. 20, N.7, p. 953-967.

SADOSKI, D. **Three tier software architectures**. 2002. Disponível em: < http://www.sei.cmu.edu/str/descriptions/threetier.html#34492 > < http://www.exforsys.com/tutorials/application-development/three-tier-software-architectures/1.html >. Acesso em: 14 abr. 2010.

SADOSKI, Darleen. **Client/Server Software Architectures – An Overview.** 2001. Disponível em: < http://www.sei.cmu.edu/str/descriptions/clientserver_body.html > < http://www.iwi.uni-hannover.de/lv/seminar_ss03/Linck/html-sources/Client-Server%20Software%20Architectures--An%20Overview.htm >. Acesso em: 13 abr. 2010.

SISOFTWARE. **SANDRA Software Benchmark.** 2010. Disponível em: < http://www.sisoftware.net > Acesso em: 19 out. 2010.

SAYDAN, A.; KUROSE, J.F.; ROSS, K.W. **Redes de Computadores e a Internet: Uma Nova Abordagem**. São Paulo: Addison-Wesley, 2003. p. 572.

SOURCEFORGE. **FrostWire: The Fastest Bit Torrent/Gnutella P2P Program.** 2010. Disponível em: http://www.frostwire.com/ Acesso em: Fevereiro 2011.

SZEREDI, M. **SSH Filesystem. Avaliable**. 2010. Disponível em: http://fuse.sourceforge.net/sshfs.html. Acesso em: Janeiro 2011.

TAMOSOFT. **CommView for WiFi.** 2010. Disponível em: http://www.tamos.com/products/commview/ Acesso em: 19 jan. 2011.

TANENBAUM, Andrew S. **Computer Network**, 4nd Edition. 2002. Saddle River, New Jersey: Prentice Hall. p. 912.

TANENBAUM, Andrew S.; VAN STEEN, Marrten. **Sistemas Distribuídos: Princípios e Paradigmas.** 2007. Upper Saddle River, New Jersey: PEARSON. p. 416.

TSAI, P. **Content-based Retrievel of MP3 Music Objects.** Proceedings of the tenth international conference on Information and knowledge management. 2001. New York, USA. P. 506 – 511.

TSAI, W.; LEE, C. **A New Steganographic Method Based on Information Sharing via PNG Images.** The 2nd International Conference on Computer and Automation Engineering (ICCAE). 2010. Singapore. P. 807-811

ULRIKE, T., DOMINIK, R., FRANZ, K., SAHIL, S., PETER, R. **A new software/hardware architecture for real time image processing of wide area airborne camera images.** Journal of Real-Time Image Processing. 2008. Disponível em: < http://www.springerlink.com/content/408m035553278523/ > Acesso em 30 de abr. 2010.

VMWARE. **Virtual *Desktop* Infra-structure.** 2007. Disponível em: http://www.vmware.com/pdf/virtual_*desktop*_infrastructure_wp.pdf. Acesso em 31 de Março de 2010.

WEISER, M. **Some computer science issues in ubiquitous computing.** *Comms* 1993. ACM, Vol. 36, No 7, pgs 74-84.

YLONEN, T. **The Secure Shell (SSH) Protocol Architecture**, RFC 4251, 2006a. Disponível em: http://tools.ietf.org/html/rfc4251 . Acesso em 09 set. 2010.

YLONEN, T. **The Secure Shell (SSH) Transport Layer Protocol** , RFC 4253, 2006b. Disponível em: http://tools.ietf.org/html/rfc4253 . Acesso em 11 set. 2010.

YLONEN, T. **The Secure Shell (SSH) Authentication Protocol**, RFC 4252, 2006c. Disponível em: http://tools.ietf.org/html/rfc4252 . Acesso em 13 set. 2010.

YLONEN, T. **The Secure Shell (SSH) Connection Protocol**, RFC 4254, 2006d. Disponível em: http://tools.ietf.org/html/rfc4254. Acesso em 16 set. 2010.

ZHOU, Y.; ZHANG, Y.; XIE, Y. **Virtual Disk based Centralized Management for Enterprise Networks.** ACM Special Interest Group on Data Communication (SIGCOMM). 2006. Pisa, Italia. P. 23.

# ANEXO A

**Lista de placa de rede sem fio suportadas pelo *middleware* WSE-OS.**
**Alguns dos produtos descritos estão agrupados por família de fabricação.**

**.** Marvell 8XXX PCI/PCIe

. Cisco/Airnet 34X/ 35X/ 4500/ 4800 ISA/PCI/PCMCIA

. Atmel 802.11n PCI/USB Card

. Planet WL3501 PCMCIA Card

. Intersil GT/ Disette/ Indigo PCI/CardBus

. USB ZD1201 based Wireless

. RNDIS USB

. RealTek 8180/8185 PCI

. RealTek 8187/8187b USB

. AmdTek Amd8211

. Atheros Wireless Card AGN

. BroadCom 43XX

. BroadCom 43XX Legacy

. Intel PRO/Wireless 2100 Network Connection

. Intel PRO/Wireless 2200BG/2915AB6

. Intel Next GEN AGN

. Intel WiFi 4965 AGN

. Intel WiFi 5000 AGN

. Intel PRO 3945AB6

. Intel MulticoMM 3200

. Marvel 8XXX Libertas WLAN

. Hermes Chipset 802.11b

. Softmac Prism54 Support

. Ralink Drive Support PCI/PCMCIA/USB

. TI WL 12XX

. ZyDAS ZD1217/ZD1211B USB

# ANEXO B

**Implementação da Interface *Middleware* WSE-OS.**
**Arquivo em *Shell Script* com chamadas *XDialog*.**
**Arquivo "Interface_sew_os.sh".**

```
# Interface Base Gerenciamento – WSE-OS

#FUNÇÃO INICIALIZAÇÃO
start_I() {
f_menu
teste=$?
while [ $teste -ne 0 ]
do
    f_menu
    teste=$?
done
}
#=============================================================
#FUNÇÃO MENU PRINCIPAL
f_menu() {
    menu=$( Xdialog --stdout --no-tags --no-cancel --center --backtitle
'*************************************************************************
\nBem-Vindo ao Wireless Sharing Environment - Operating
Systems\n*************************************************************
********' --title 'MENU PRINCIPAL SEW-OS' --menu '\n\nEste é o Menu Principal
do Middleware SEW-OS. As Funcionalidades estão Listadas no Menu abaixo.\nAntes
de Conectar ao Servidor SEW-OS,\n entre em Contato com o Administrador do
Sistema caso ainda não tenha um Usuário e Senha.\nSerá necessário o Cadastro
de Usuário para que a Autenticação ocorra com Sucesso.\n\n\n\n\n\nEscolha
Alguma das Opções:' 50 50 0 1 'Conectar ao Servidor SEW-OS' 2 'Opções
Avançadas' 3 'Desligar Terminal' )
}

#=============================================================
#FUNÇÃO LOGIN SEW-OS (user)
login_sew_os_user() {
    user=$( Xdialog --stdout --left --backtitle
'*********************************************************
\nWireless Sharing Environment - Operating
Systems\n*********************************************************' --title
'LOGIN USUÁRIO SEW-OS' --inputbox '\nEntre em Contato com o Administrador do
Sistema caso ainda não tenha um Cadastro de Usuário no Servidor SEW-OS.\nO
Nome de Usuário é necessário para que ocorra a Autenticação de Acesso.\nUma
Senha de Acesso, correspondente ao Usuário informado neste Campo, será
Solicitada no Próximo Passo.\nEm caso de Cadastro Remoto, assinale a Opção
"Login as a Guest User" na próxima tela e entre com o Login reservado
"User_sew".\n\n\n\n\n\n\nEntre com o Nome de Usuário Cadastrado no Servidor
SEW-OS:' 1000 1000 )
}


#=============================================================
```

```
#FUNÇÃO MENU OPÇÕES AVANÇADAS
f_menu_avan() {
    menu_avan=$( Xdialog --stdout --backtitle
'************************************************************
\nWireless Sharing Environment - Operating
Systems\n************************************************************' --no-
tags --cancel-label 'Voltar' --title 'OPÇÕES AVANÇADAS DE CONFIGURAÇÃO DO
MIDDLEWARE SEW-OS' --menu '\n\nAqui estão as Opções Avançadas de Configuração
do Middleware SEW-OS.\nAs Configurações estabelecidas são as Padrões para a
Conexão com o Servidor.\nAs mudanças desses Parâmetros são válidas. Em caso de
dúvida, entre em contato com o Administrador do Sistema.\n\n\n\n\nEscolha
Alguma das Opções:' 1000 1000 0 I 'Configuração de Cache em Memória
Secundária' II 'Configuração de Cache em Memória Primária' III 'Configuração
de Imagem' IV 'Configuração de Rede')
}

#============================================================
#FUNÇÃO CACHE EM DISCO
f_cache_disc() {
  item_cache_disc=$( Xdialog --stdout --backtitle
'************************************************************
\nSharing Environment Wireless - Operation
Systems\n************************************************************' --no-
tags --title 'CONFIGURAÇÃO DE CACHE' --radiolist 'Selecione o Valor Desejado
para o Cache em Disco' 1000 1000 0 0 '    0 Mbytes de Cache em Disco' off 32
'  32 Mbytes de Cache em Disco' off 64 '  64 Mbytes de Cache em Disco' on 128
'128 Mbytes de Cache em Disco' off 256 '256 Mbytes de Cache em Disco' off 512
'512 Mbytes de Cache em Disco' off )
}

#============================================================
#FUNÇÃO CACHE EM MEMÓRIA
f_cache_mem() {
  item_cache_mem=$( Xdialog --stdout --backtitle
'************************************************************
\n Wireless Sharing Environment- Operating
Systems\n************************************************************' --no-
tags --title 'CONFIGURAÇÃO DE CACHE' --radiolist 'Selecione o Valor Desejado
para o Cache em Memória' 1000 1000 0 4 '    4 Mbytes de Cache em Memória RAM'
off 8 '    8 Mbytes de Cache em Memória RAM' off 16 '  16 Mbytes de Cache em
Memória RAM' on 32 '  32 Mbytes de Cache em Memória RAM' off 64 '  64 Mbytes
de Cache em Memória RAM' off 128 '128 Mbytes de Cache em Memória RAM' off )
}

#============================================================
#FUNÇÃO CONFIGURAÇÃO DE REDE IP
f_conf_rede_IP() {
  rede_ip=$( Xdialog --stdout --backtitle
'************************************************************\n
Wireless Sharing Environment - Operating
Systems\n************************************************************' --title
'CONFIGURAÇÃO DE REDE' --inputbox 'Entre com o Número IP do Servidor SEW-OS:
(Ex.: XXX.XXX.XXX.XXX) ' 1000 1000 )

}
```

```
#============================================================
#FUNÇÃO CONFIGURAÇÃO DE REDE PORTA
f_conf_rede_PO() {
  rede_port=$( Xdialog --stdout --backtitle
'************************************************************
\nWireless Sharing Environment - Operating
Systems\n************************************************************' --title
'CONFIGURAÇÃO DE REDE' --inputbox 'Entre com o Número de Porta do Servidor
SEW-OS: ' 1000 1000 )
}

#============================================================
#FUNÇÃO CONFIGURAÇÃO DE IMAGEM
f_conf_img() {
  compressao=$(Xdialog --stdout --backtitle
'************************************************************
\nWireless Sharing Environment - Operating
Systems\n************************************************************' --no-
tags --title 'CONFIGURAÇÃO DE IMAGEM' --radiolist 'Escolha o Tipo de
Compressão a ser Utilizado pelo Middleware SEW-OS:' 1000 1000 0 0 'Desativar
Compressão de Imagens' off 1 'Ativar Apenas Compressão JPEG' off 2 'Ativar
Apenas Compressão RGB' off 3 'Ativar Compressão JPEG e RGB' on)
}

#============================================================
#FUNÇÃO RESTAURA PADRÃO REDE
f_default_rede(){
  rede_ip="192.168.1.101"
  rede_port="22"
}

#============================================================
#FUNÇÃO RESTAURA PADRÃO CACHE
f_default_cache(){
  item_cache_disc="64"
  item_cache_mem="16"
}

#============================================================
#FUNÇÃO RESTAURA PADRÃO CACHE
f_default_img(){
  compressao="3"
}

#============================================================
#FUNÇÃO PROCESSAMENTO
proces_dados() {

/home/mestrado/.nx/config
cat /home/mestrado/.nx/config/SegSEW-OS.nxs|sed
"s/CACHE_DISC/$item_cache_disc/g">/home/mestrado/.nx/config/temp.nxs
cat /home/mestrado/.nx/config/temp.nxs|sed
"s/CACHE_MEM/$item_cache_mem/g">/home/mestrado/.nx/config/temp2.nxs
cat /home/mestrado/.nx/config/temp2.nxs|sed
"s/SEW_OS_IP/$rede_ip/g">/home/mestrado/.nx/config/temp3.nxs
cat /home/mestrado/.nx/config/temp3.nxs|sed
"s/SEW_OS_PORT/$rede_port/g">/home/mestrado/.nx/config/temp4.nxs
```

```
cat /home/mestrado/.nx/config/temp4.nxs|sed
"s/SEW_OS_COMPR/$compressao/g">/home/mestrado/.nx/config/temp5.nxs
cat /home/mestrado/.nx/config/temp5.nxs|sed
"s/SEW_OS_user/$user/g">/home/mestrado/.nx/config/exec.nxs

rm /home/mestrado/.nx/config/temp.nxs
rm /home/mestrado/.nx/config/temp2.nxs
rm /home/mestrado/.nx/config/temp3.nxs
rm /home/mestrado/.nx/config/temp4.nxs
rm /home/mestrado/.nx/config/temp5.nxs


nxclient --session /home/mestrado/.nx/config/exec.nxs
}

#============================================================
direction="99"
f_default_cache
f_default_rede
f_default_img

while : ; do
case "$direction" in
    "99")
        start_I
        direction="$menu"
        ;;
    "1")
        login_sew_os_user
            if [ "$?" -ne "0" ]; then direction="99"
        else
            direction="10"
         fi
        ;;
    "2")
        f_menu_avan
        if [ "$?" -ne "0" ]; then direction="99"
        else
        direction="$menu_avan"
    fi
    ;;
    "I")
    f_cache_disc
        if [ "$?" -ne "0" ]; then direction="2"
                    f_default_cache
        else
        Xdialog --backtitle
'*********************************************************
\nWireless Sharing Environment - Operating
Systems\n*********************************************************' --title
'CONFIGURAÇÃO DE CACHE EM DISCO' --msg 'CACHE EM DISCO ALTERADO COM SUCESSO.'
1000 1000
        direction="2"
    fi
        ;;
    "II")
    f_cache_mem
```

```
        if [ "$?" -ne "0" ]; then direction="2"
                        f_default_cache
        else
        Xdialog --backtitle
'************************************************************
\nWireless Sharing Environment - Operating
Systems\n************************************************************' --title
'CONFIGURAÇÃO DE CACHE EM MEMÓRIA RAM' --msg 'CACHE EM MEMÓRIA RAM ALTERADO
COM SUCESSO.' 1000 1000
        direction="2"
    fi
        ;;
    "III")
    f_conf_img
        if [ "$?" -ne "0" ]; then direction="2"
                        f_default_img
        else
        Xdialog --backtitle
'************************************************************
\nWireless Sharing Environment - Operating
Systems\n************************************************************' --title
'CONFIGURAÇÃO DE IMAGEM' --msg 'COMPRESSÃO DE IMAGEM ALTERADA COM SUCESSO.'
1000 1000
        direction="2"
    fi
    ;;
    "IV")
    Xdialog --left --backtitle
'************************************************************
\nWireless Sharing Environment - Operating
Systems\n************************************************************' --title
'CONFIGURAÇÃO DE REDE SEW-OS' --cancel-label 'Padrão SEW-OS' --center --yesno
'Alterações de IP e Porta do Servidor SEW-OS podem ser Realizadas.\nEntre em
Contato com o Administrador do Sistema para a Configuração Correta dos
Parâmetros.\n\n\n\n\n\n\n\n\n\nAperte o Botão "Sim" para Continuar a
Configuração.\nAperte o Botão "Padrão SEW-OS" para Configurações Originais do
Sistema.' 1000 1000

    if [ "$?" -ne "0" ]; then direction="2"
                f_default_rede
    else
    f_conf_rede_IP
        if [ "$?" -ne "0" ]; then direction="2"
                        f_default_rede
            else
        f_conf_rede_PO
            if [ "$?" -ne "0" ]; then direction="2"
                        f_default_rede
            else
        Xdialog -backtitle

'************************************************************
\nWireless Sharing Environment - Operating
Systems\n************************************************************' --title
'CONFIGURAÇÃO DE REDE' --msg 'CONFIGURAÇÃO DE REDE ALTERADA COM SUCESSO.' 1000
1000
        direction="2"
```

```
        fi
        fi
    fi
        ;;

    "3")
    Xdialog --left --backtitle
'************************************************************
\nWireless Sharing Environment - Operating
Systems\n************************************************************' --title
'DESLIGAR TERMINAL' --cancel-label 'Voltar Ao Menu SEW-OS' --center --yesno
'Deseja Desligar o Terminal?\n\n\n\n\nAperte o Botão "Sim" para Desligar a
Máquina.\nAperte o Botão "Voltar ao Menu SEW-OS" para Cancelar esta Ação.'
1000 1000
    if [ "$?" -ne "0" ]; then direction="99"
    else shutdown -h now
    fi
    ;;
    "10")
    proces_dados
    direction="11"
    ;;
    "11")
    Xdialog --backtitle
'************************************************************
\nWireless Sharing Environment - Operating
Systems\n************************************************************' --title
'CONEXÃO CLIENTE-SERVIDOR' --msg 'SESSÃO FINALIZADA.\nA SUA CONEXÃO COM O
SERVIDOR FOI DESFEITA.\nPRESSIONE O BOTÃO "OK" PARA VOLTAR AO MENU PRINCIPAL.'
1000 1000
    rm /home/mestrado/.nx/config/exec.nxs
    direction="99"
    ;;
esac

done
```

# ANEXO C

**Arquivo de Configurações Módulo de Comunicação Cliente.**
**Configurações do Sistema de Transmissão Diferencial de Dados mais Configuração de Rede.**
**Arquivo "SegSEW-OS.nxs"**

```
<!DOCTYPE NXClientSettings>
<NXClientSettings application="nxclient" version="1.3" >
<group name="Advanced" >
<option key="Cache size" value="CACHE_MEM" />
<option key="Cache size on disk" value="CACHE_DISC" />
<option key="Current keyboard" value="true" />
<option key="Custom keyboard layout" value="" />
<option key="Disable DirectDraw" value="false" />
<option key="Disable ZLIB stream compression" value="false" />
<option key="Disable deferred updates" value="false" />
<option key="Enable HTTP proxy" value="false" />
<option key="Enable SSL encryption" value="true" />
<option key="Enable response time optimisations" value="false" />
<option key="Grab keyboard" value="false" />
<option key="HTTP proxy host" value="" />
<option key="HTTP proxy port" value="8080" />
<option key="HTTP proxy username" value="" />
<option key="Remember HTTP proxy password" value="false" />
<option key="Restore cache" value="true" />
<option key="StreamCompression" value="" />
</group>
<group name="Environment" >
<option key="CUPSD path" value="/usr/sbin/cupsd" />
</group>
<group name="General" >
<option key="Automatic reconnect" value="true" />
<option key="Command line" value="/home/gustavo/teste.sh" />
<option key="Custom Unix Desktop" value="application" />
<option key="Desktop" value="console" />
<option key="Disable SHM" value="false" />
<option key="Disable emulate shared pixmaps" value="false" />
<option key="Link speed" value="wan" />
<option key="Remember password" value="false" />
<option key="Resolution" value="fullscreen" />
<option key="Resolution height" value="600" />
<option key="Resolution width" value="800" />
<option key="Server host" value="SEW_OS_IP" />
<option key="Server port" value="SEW_OS_PORT" />
<option key="Session" value="unix" />
<option key="Spread over monitors" value="false" />
<option key="Use default image encoding" value="0" />
<option key="Use render" value="true" />
<option key="Use taint" value="true" />
<option key="Virtual desktop" value="true" />
<option key="XAgent encoding" value="true" />
<option key="displaySaveOnExit" value="true" />
<option key="xdm broadcast port" value="177" />
```

```
<option key="xdm list host" value="localhost" />
<option key="xdm list port" value="177" />
<option key="xdm mode" value="server decide" />
<option key="xdm query host" value="localhost" />
<option key="xdm query port" value="177" />
</group>
<group name="Images" >
<option key="Disable JPEG Compression" value="0" />
<option key="Disable all image optimisations" value="false" />
<option key="Disable backingstore" value="false" />
<option key="Disable composite" value="false" />
<option key="Image Compression Type" value="SEW_OS_COMPR" />
<option key="Image Encoding Type" value="0" />
<option key="Image JPEG Encoding" value="false" />
<option key="JPEG Quality" value="6" />
<option key="RDP Image Encoding" value="3" />
<option key="RDP JPEG Quality" value="6" />
<option key="RDP optimization for low-bandwidth link" value="false" />
<option key="Reduce colors to" value="" />
<option key="Use PNG Compression" value="true" />
<option key="VNC JPEG Quality" value="6" />
<option key="VNC images compression" value="3" />
</group>
<group name="Login" >
<option key="Auth" value="EMPTY_PASSWORD" />
<option key="Guest Mode" value="false" />
<option key="Guest password" value="" />
<option key="Guest username" value="" />
<option key="Login Method" value="nx" />
<option key="Public Key" value="-----BEGIN RSA PRIVATE KEY-----
MIIBuwIBAAKBgQCXv9AzQXjxvXWC1qu3CdEqskX9YomTfyG865gb4D02ZwWuRU/9
C3I9/bEWLdaWgJYXIcFJsMCIkmWjjeSZyTmeoypI1iLifTHUxn3b7WNWi8AzKcVF
aBsBGiljsop9NiD1mEpA0G+nHHrhvTXz7pUvYrsrXcdMyM6rxqn77nbbnwIVALCi
xFdHZADw5KAVZI7r6QatEkqLAoGBAI4L1TQGFkq5xQ/nIIciW8setAAIyrcWdK/z
5/ZPeELdq70KDJxoLf81NL/8uIc4PoNyTRJjtT3R4f8Az1TsZWeh2+ReCEJxDWgG
fbk2YhRqoQTtXPFsI4qvzBWct42WonWqyyb1bPBHk+JmXFscJu5yFQ+JUVNsENpY
+Gkz3HqTAoGANlgcCuA4wrC+3Cic9CFkqiwO/Rn1vk8dvGuEQqFJ6f6LVfPfRTfa
QU7TGVLk2CzY4dasrwxJ1f6FsT8DHTNGnxELPKRuLstGrFY/PR7KeafeFZDf+fJ3
mbX5nxrld3wi5titTnX+8s4IKv29HJguPvOK/SI7cjzA+SqNfD7qEo8CFDIm1xRf
8xAPsSKs6yZ6j1FNklfu
-----END RSA PRIVATE KEY-----
" />
<option key="User" value="SEW_OS_user" />
</group>
<group name="Services" >
<option key="Audio" value="false" />
<option key="IPPPort" value="631" />
<option key="IPPPrinting" value="false" />
<option key="Shares" value="false" />
</group>
<group name="VNC Session" >
<option key="Display" value="0" />
<option key="Remember" value="false" />
<option key="Server" value="" />
</group>
<group name="Windows Session" >
<option key="Application" value="" />
```

```
<option key="Authentication" value="2" />
<option key="Color Depth" value="8" />
<option key="Domain" value="" />
<option key="Image Cache" value="true" />
<option key="Password" value="EMPTY_PASSWORD" />
<option key="Remember" value="true" />
<option key="Run application" value="false" />
<option key="Server" value="" />
<option key="User" value="" />
</group>
<group name="share chosen" >
<option key="Share number" value="0" />
</group>
</NXClientSettings>
```