UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

FLEXVAPS: UM SISTEMA DE GERENCIAMENTO DE *VIRTUAL APPLIANCES* PARA MÁQUINAS VIRTUAIS HETEROGÊNEAS

DISSERTAÇÃO DE MESTRADO

Diego Luís Kreutz

Santa Maria, RS, Brasil

2009

FLEXVAPS: UM SISTEMA DE GERENCIAMENTO DE *VIRTUAL APPLIANCES* PARA MÁQUINAS VIRTUAIS HETEROGÊNEAS

por

Diego Luís Kreutz

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de
Mestre em Computação

Orientador: Prof. Dr. Benhur de Oliveira Stein (UFSM)

Co-orientador: Profª Drª Andrea Schwertner Charão (UFSM)

Santa Maria, RS, Brasil

2009

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Informática

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

FLEXVAPS: UM SISTEMA DE GERENCIAMENTO DE *VIRTUAL APPLIANCES* PARA MÁQUINAS VIRTUAIS HETEROGÊNEAS

elaborada por
Diego Luís Kreutz

como requisito parcial para obtenção do grau de
Mestre em Computação

COMISSÃO EXAMINADORA:

Profª Drª Andrea Schwertner Charão (UFSM)
(Presidente/Co-orientador)

Prof. Dr. Maurício Lima Pilla (UFPel)

Profª Drª Iara Augustin (UFSM)

Santa Maria, 23 de Outubro de 2009.

“A imaginação é mais importante que o conhecimento.” — ALBERT EINSTEIN

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

***FLEXVAPS*: UM SISTEMA DE GERENCIAMENTO DE *VIRTUAL APPLIANCES* PARA MÁQUINAS VIRTUAIS HETEROGÊNEAS**

Autor: Diego Luís Kreutz
Orientador: Prof. Dr. Benhur de Oliveira Stein (UFSM)
Co-orientador: Prof[a] Dr[a] Andrea Schwertner Charão (UFSM)
Local e data da defesa: Santa Maria, 23 de Outubro de 2009.

Com o desenvolvimento e a proliferação dos monitores de máquinas virtuais, surgiu um novo conceito, uma nova tendência na área de virtualização de sistemas: *virtual appliance*. Este, segundo a origem da definição, é um pacote de dados que pode ser distribuído eletronicamente. Hoje, existem repositórios livres e comerciais de *virtual appliances* no mercado. Esses pacotes são constituídos por um sistema operacional e aplicativos configurados e otimizados para resolver um determinado problema computacional. Há uma grande variedade de *virtual appliances*, para os mais diversos monitores de máquinas virtuais, disponíveis aos usuários finais. Essa diversidade leva a máquinas virtuais heterogêneas, ou seja, instâncias de *virtual appliances* que são executadas sobre diferentes monitores de máquinas virtuais. Sendo assim, o objeto deste trabalho é desenvolver uma solução para o gerenciamento de *virtual appliances* para máquinas virtuais heterogêneas, algo ainda pouco explorado. O objetivo é proporcionar aos usuários finais a possibilidade de gerenciar ambientes diversos, como laboratórios de informática, aglomerados de computadores, redes de computadores e ambientes de pesquisa e produção em geral, utilizando-se da diversidade, flexibilidade e disponibilidade dos *virtual appliances*. Criar e manter esses pacotes de dados é algo mais prático e eficaz, tanto para administradores de sistemas, quanto para usuários finais. Ademais, compartilhar os diferentes *virtual appliances* evita desperdícios de tempo e espaço, como costuma ocorrer com abordagens tradicionais. Neste trabalho é proposto, prototipado e avaliado um sistema de gerenciamento de *virtual appliances* para máquinas virtuais heterogêneas. Este sistema demonstrou-se viável para o gerenciamento de repositórios de *VAPs* em parques de máquinas, simplificando o trabalho dos usuários finais e dos administradores de sistemas.

Palavras-chave: Virtualização, virtual appliance, modelo de gerenciamento de virtual appliances, sistemas em rede.

ABSTRACT

Master's Dissertation
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

FLEXVAPS: A VIRTUAL APPLIANCE MANAGEMENT SYSTEM FOR HETEROGENEOUS VIRTUAL MACHINES

Author: Diego Luís Kreutz
Advisor: Prof. Dr. Benhur de Oliveira Stein (UFSM)
Coadvisor: Prof[a] Dr[a] Andrea Schwertner Charão (UFSM)

Virtual appliance is a new concept derived from the virtual machine monitors world. It is considered a data package that can be electronically distributed. Today, there area free and commercial virtual appliance repositories available on the market. These data packages are mostly composed by pre-configured and pre-optimized operating systems and applications joined to solve a specific computing problem. The growing virtual machine monitor market has lead to a wide virtual appliance variety available to end users. This diversity naturally results in heterogeneous virtual machines, ie, instances of virtual appliances that run on different virtual machine monitors. Thus, this work has as main goal propose and prototype a virtual appliance management solution for heterogeneous virtual machines, something less explored until now. The aim is to provide end users the ability of managing different environments, using virtual appliances, such as computer labs, clusters, networks and other kinds of computing environments. Virtual appliance creation and maintenance is more practical and effective for both system administrators and end users. Moreover, the sharing of different virtual appliances avoids the waste of time and disk space, as usually occurs with traditional approaches. In this work it is proposed, prototyped and evaluated a system for managing virtual appliances for heterogeneous virtual machines. This system has proved to be feasible for using *VAP* repositories to manage networked machines and systems, simplifying the work of end users and system administrators.

Keywords: virtual appliances, virtual machine monitors, heterogeneous virtual machines, management system.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CVS	Concurrent Version System
DHCP	Dinamic Host Configuration Protocol
DNS	Domain Name Service
GB	Giga Bytes
GPL	General Public License
IP	Internet Protocol
LDAP	Lightweight Directory Access Protocol
MAC	Media Access Control
MB	Mega Bytes
MMV	Monitor de Máquinas Virtuais
NFS	Network File System
PAM	Pluggable Authentication Modules
SMTP	Simple Message Transport Protocol
SO	Sistema Operacional
SSH	Secure Shell
SVP	Servidor Virtual Privado
URI	Universal Resource Identifier
URL	Universal Resource Locator
VAN	Virtual Appliance Network
VAP	Virtual Appliance
WWW	World Wide Web
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Virtualização tem ressurgido como uma solução para uma gama cada vez maior de problemas. Neste cenário, os monitores de máquinas virtuais (*MMV*) vêm proliferando-se rapidamente entre diferentes tipos de usuários. Eles tornaram possíveis aplicações e ambiente praticamente impensáveis na forma tradicional, isto é, com equipamentos reais. Nesse sentido, muitas pesquisas vêm sendo desenvolvidas e têm gerado soluções baseadas em virtualização para os mais variados tipos de problemas, indo desde simples experimentos de software até ambientes distribuídos complexos e totalmente virtualizados.

Com a proliferação das soluções de virtualização, novos conceitos e propostas de infraestruturas de computação começaram a surgir. *Virtual Appliance* (*VAP*) é uma das novas definições que permeiam a área de virtualização. Um *virtual appliance* é um encapsulamento de um estado de uma máquina virtual [64]. É um pacote pré-configurado e pré-otimizado para uma aplicação ou solução em específico. Segundo Sapuntzakis [46], um *virtual appliance* é o estado de um *appliance* real (o conteúdo dos discos do *appliance*), bem como a descrição do *hardware* (exemplo: dois adaptadores Ethernet, 256 *MB* de *RAM*, dois discos rígidos, etc.). Um *VAP* é similar a um *appliance* físico qualquer, porém sem hardware [45]. Ele pode empacotar desde uma simples aplicação, até um conjunto completo de software, formando uma infraestrutura de computação para soluções e serviços dos mais variados tipos.

Ainda segundo Sapuntzakis [45], um *VAP* pode ser uma rede de *Virtual Appliances*, que é denominada de *virtual appliance network* (*VAN*). Por exemplo, um *VAP* corporativo pode ser constituído de diferentes *VAPs*. Uma rede local tradicionalmente é composta por servidores como DNS, DHCP, LDAP, SMTP, servidores de arquivos e serviços de *firewall*. Nesse contexto, agrupando os *VAPs* em uma *VAN*, o custo final de implantação e adminis-

tração torna-se menor, pois a *VAN* fornece tudo o que é necessário para operacionalizar o básico de uma rede.

Dado esse contexto e a diversidade de novas soluções, surgiram projetos e pesquisas como Collective [45, 11, 29, 46], frameworks virtuais para *data centers* [64, 63, 61], infraestruturas virtuais para GRIDs [73, 9, 18], soluções virtuais na área de serviços online [12, 63], pesquisas aplicando virtualização à computação autonômica [33, 61, 62], soluções de automatização para cópia e ativação de imagens virtuais [23], ambientes como o PDS [3] e o Redar [77], entre outros [57, 59, 58, 60, 72, 53, 4, 27, 40, 10, 31]. Porém, soluções como Collective e os *frameworks* virtuais para *data centers* não provêem recursos para o gerenciamento de *virtual appliances* para máquinas virtuais heterogêneas, foco do presente trabalho. O projeto Collective, por exemplo, trabalha com máquinas virtuais do *MMV*Vmware. Os *frameworks* virtuais para *data centers* trabalham ou com Vmware ou com Xen.

O objetivo deste trabalho é apresentar uma solução para o gerenciamento de *virtual appliances* para máquinas virtuais heterogêneas. Para que isso seja possível, é necessário que a solução forneça níveis de flexibilidade e gerenciabilidade capazes de transpor as diferentes peculiaridades dos diversos *MMVs*.

1.1 Motivação

A proliferação de *virtual appliances* deve-se principalmente a fatores como:

- facilidade de criação e manutenção de *VAPs*;
- praticidade na distribuição dos *VAPs* (pacotes de dados que podem ser transmitidos pela rede);
- grande diversidade e disponibilidade de *MMVs*, para os mais variados tipos de sistemas e ambientes computacionais;
- redução das demandas físicas de *hardware* específico para a avaliação e uso de sistemas e soluções de *software*;
- flexibilidade, provida pelos *MMVs*, de criação, uso e atualização de máquinas virtuais.

Dados esses fatores, é importante a existência de uma infraestrutura para o gerenciamento de contextos de aplicação de *virtual appliances* em ambientes computacionais diversos [1].

Em suma, as pesquisas e soluções anteriormente citadas deixam a desejar em aspectos fundamentais para laboratórios temáticos. Outros contextos, como *data centers* e ambientes de pesquisa e produção em geral, carecem de soluções de gerenciamento de ambientes virtuais e máquinas virtuais. Existem soluções, na maioria das vezes proprietárias, para o gerenciamento de máquinas virtuais homogêneas, de um mesmo *MMVs*, mas praticamente inexistem soluções que trabalham com a diversidade e heterogeneidade de *MMVs*, objeto de estudo deste trabalho.

Alguns dos pontos pouco explorados pelas soluções existentes são:

- o suporte à heterogeneidade de soluções de virtualização (monitores de máquinas virtuais);
- o gerenciamento de níveis de acesso e grupos de usuários;
- a definição e modelagem de uma arquitetura de gerenciamento de *virtual appliances* para máquinas virtuais heterogêneas;
- a presença de recursos que permitam ao usuário especificar os seus requisitos e as suas demandas de acordo com a disponibilidade e ou necessidades de *virtual appliances*.

Sendo assim, este trabalho busca suprir esta lacuna com a proposição e modelagem de uma infraestrutura para o gerenciamento de *virtual appliances* em ambientes computacionais, explorando os aspectos há pouco relacionados.

1.2 Objetivos e Contribuição

O trabalho tem como principal objetivo desenvolver um sistema para o gerenciamento flexível de *virtual appliances* em ambientes heterogêneos, que demandam soluções distintas para os diferentes tipos de problemas computacionais. Com isso, busca-se proporcionar uma estrutura capaz de suprir as exigências mais comuns de ambientes reais diversos, heterogêneos por natureza.

[1] A diversidade implica invariavelmente em heterogeneidade, no sentido da necessidade de soluções e ambientes distintos para os diferentes problemas e contextos computacionais da atualidade.

A solução proposta tem por objetivo atender aos seguintes requisitos:

- disponibilizar um repositório global de *VAPs* aos usuários de uma rede;
- permitir aos usuários o acesso simples e transparente ao repositório global;
- possibilitar o gerenciamento de ambientes computacionais diversos, atendendo a uma gama de demandas específicas por sistemas;
- possibilitar a criação e o gerenciamento de domínios de *VAPs*[2], a serem disponibilizados a grupos específicos de usuários (exemplo: alunos da disciplina de Rede de Computadores);
- possuir uma arquitetura de *software* que utiliza preferencialmente ferramentas e soluções existentes e conhecidas por administradores de sistemas, para o gerenciamento e disponibilidade de *VAPs*;
- diminuir o ônus dos administradores de sistema no gerenciamento de ambientes computacionais diversos, heterogêneos por natureza;
- flexibilizar aos usuários o uso de ambientes computacionais heterogêneos de forma mais simples e prática, fazendo uso dos recursos e vantagens inerentes aos *MMVs*;
- permitir um maior compartilhamento de soluções virtuais entre usuários de sistemas, aumentando a quantidade de recursos e soluções disponíveis para todos, o que irá refletir diretamente na produtividade, pois os usuários poderão focar-se mais nos problemas a serem resolvidos e menos na preparação de ambientes e infraestruturas.

A principal contribuição do trabalho é a definição e prototipação de uma arquitetura básica para o gerenciamento de *virtual appliances* em ambientes computacionais diversos. Essa arquitetura é capaz de atender a conjuntos variados de demandas dos usuários e das organizações, representando uma alternativa prática para gerenciar contextos que envolvem demandas heterogêneas de ambientes que precisam ser disponibilizados aos usuários das organizações.

[2]Domínios de *VAPs*, no contexto deste trabalho, corresponde a grupos de *VAPs* associados a grupos de usuários, ou seja, com acesso e disponibilidade restritos.

Adicionalmente, o trabalho também apresenta um rápido sumário do estado da arte em virtualização de sistemas. No decorrer do texto, são apresentadas diferentes tecnologias e casos de aplicação de virtualização, demonstrando a importância e a tendência de crescimento dessa área.

1.3 Organização do Texto

O trabalho está organizado em seis capítulos. O capítulo 2 apresenta uma revisão de literatura sobre os principais assuntos e trabalhos relacionados com a solução proposta. O objetivo é relacionar as principais tendências e aplicações na área de virtualização e apresentar os trabalhos correlatos.

No capítulo 3, é apresentada a solução proposta, discriminando-se suas características e peculiaridades que tornam possível o gerenciamento de diferentes ambientes de computação virtuais. Para exemplificar melhor a aplicabilidade da arquitetura de sistema proposta, são enumerados alguns cenários de aplicação.

O capítulo 4 descreve o desenvolvimento de um protótipo da arquitetura proposta. Inicia com uma análise de tecnologias de desenvolvimento e prossegue com a apresentação da implementação propriamente dita. O fechamento do capítulo é realizado com a apresentação de exemplos de configuração de uso do sistema.

Uma avaliação do trabalho, proposta e prototipação, é apresentada no capítulo 5. Os dados apresentados nesse capítulo servem de indicadores dos resultados atingidos com o desenvolvimento do trabalho como um todo.

Por fim, o capítulo 6 apresenta a conclusão do trabalho, com as principais inferências sobre a proposta original e os resultados atingidos. O capítulo é fechado com uma relação de trabalhos futuros.

2 FUNDAMENTAÇÃO

Este capítulo apresenta os principais conceitos e fundamentações sobre máquinas virtuais, monitores de máquinas virtuais e *virtual appliances*. Esses elementos constituem a base e parte da motivação do trabalho.

2.1 Máquinas Virtuais

2.1.1 Conceitos

Uma máquina virtual representa uma camada de abstração. A abstração pode ir desde um simples conjunto de instruções até uma máquina física por completo. A figura 2.1 ilustra uma camada de abstração entre os recursos físicos (hardware) e os sistemas operacionais. Em um sistema convencional pode-se executar apenas um único sistema operacional sobre uma plataforma de hardware. A inclusão de uma camada abstrata intermediária permite a execução simultânea de vários sistemas operacionais sobre uma mesma infraestrutura física. Esse é um dos pontos explorados pelas tecnologias de virtualização.

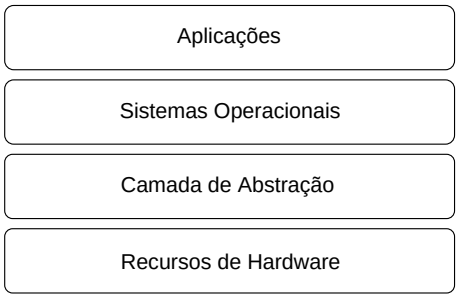

Figura 2.1: Camada de abstração entre o hardware e os sistemas operacionais

Os objetivos das máquinas virtuais são variados, incluindo desde a flexibilidade no desenvolvimento e manutenção de software até a criação de um ambiente complexo de infraestrutura de software capaz de emular um ambiente real de grandes proporções. Um exemplo de um cenário desse gênero é a criação de um ambiente de computação dis-

tribuída virtual sobre uma infraestrutura limitada e compartilhada [43]. Isto é possível graças a tecnologias como as máquinas virtuais.

As máquinas virtuais podem ser caracterizadas em dois tipos básicos: **processos** e **sistemas**. Nesta perspectiva, é importante compreender a diferença do significado de "máquina" tanto da do ponto de vista de processo quanto de sistema. Do ponto de vista de um processo executando um programa de usuário, a máquina consiste em um espaço lógico de endereçamento de memória com instruções e registradores em nível de usuário, que permitem a execução do código pertencente ao processo. Por outro lado, sob a ótica de um sistema operacional e todas as aplicações que este suporta, todo o sistema executa sobre uma máquina. Um sistema é um ambiente de execução completo capaz de suportar vários processos simultaneamente [49].

Da mesma forma que existem perspectivas de "máquina" sob o ponto de vista de processos e sistemas, existem também máquinas virtuais em nível de processo e sistema [49]. Uma máquina virtual caracterizada como processo executa um único processo. Por outro lado, uma máquina virtual em nível de sistema provê um ambiente completo e consistente que permite a execução de diferentes processos de usuário[49]. A figura 2.2 apresenta a diferença entre máquinas virtuais processo e sistema. Como pode ser observado, uma máquina virtual processo executa um único processo. Por outro lado, uma máquina virtual sistema é capaz de executar múltiplos processos. É comum uma máquina virtual sistema estar executando um sistema operacional e seu conjunto de aplicações.

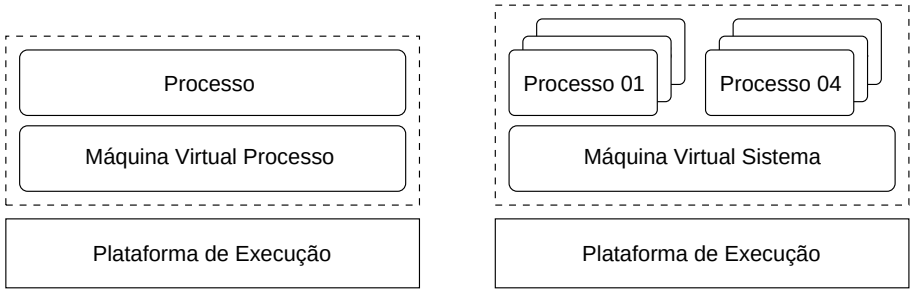

Figura 2.2: Esquerda: máquina virtual processo. Direita: máquina virtual sistema.

O uso de máquinas virtuais data da década de 60 [74]. Contudo, apenas há alguns anos o uso de virtualização ganhou novas proporções e aplicações. Com isso, surgiram, ou foram difundidos, novos conceitos e técnicas. Um desses conceitos, já empregado pela IBM na década de 60 [74], é o monitor de máquinas virtuais (*MMV*)[42]. Ele pode ser caracterizado como a peça-chave de um sistema de virtualização pelo fato de ser o

responsável pelo gerenciamento das máquinas virtuais.

2.1.2 Monitores de Máquinas Virtuais (*MMVs*)

A função de um *MMV* é monitorar e controlar as máquinas virtuais de um determinado ambiente. Essas máquinas virtuais podem conter diferentes sistemas operacionais executando sobre uma mesma plataforma de hardware. Um *MMV* é responsável por interceptar os acessos a recursos do sistema e, consequentemente, a execução de instruções privilegiadas. A figura 2.3 aborda as camadas desde o hardware até as aplicações. Como pode ser observado, entre os sistemas operacionais e o hardware há uma nova camada, o monitor de máquinas virtuais. Sem ele não seria possível executar simultaneamente múltiplos sistemas operacionais sobre uma mesma máquina.

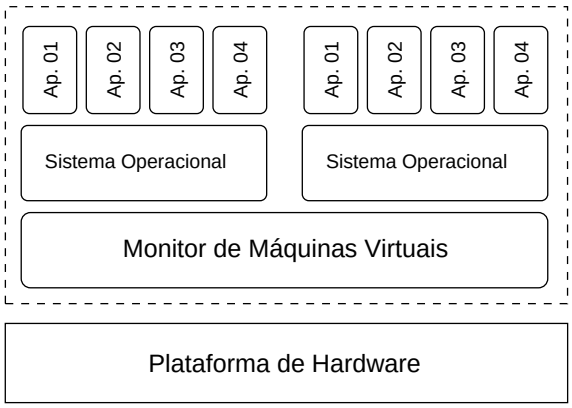

Figura 2.3: Ilustração de um Monitor de Máquinas Virtuais (*MMV*)

Uma máquina virtual nada mais é que uma instância de um sistema sob controle do *MMV*. Na figura 2.4 é apresentado um *MMV* controlando N instâncias de um sistema operacional e suas aplicações, ou seja, N máquinas virtuais.

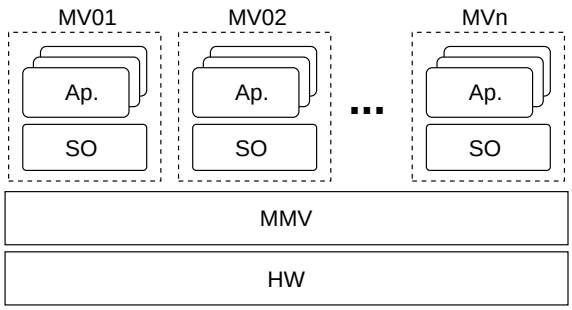

Figura 2.4: N máquinas virtuais sob o controle de um MMV

Um *MMV* gerencia o acesso aos recursos de *hardware* disponíveis no sistema físico. O objetivo é compartilhar os recursos de maneira que vários sistemas operacionais possam

ser executados simultaneamente sobre um mesmo ambiente, tirando o máximo proveito do mesmo. Um exemplo de compartilhamento é apresentado na figura 2.5, onde três sistemas operacionais distintos estão simultaneamente aproveitando um mesmo conjunto de componentes físicos.

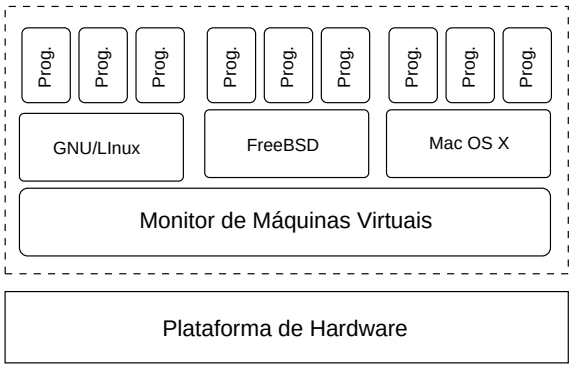

Figura 2.5: Uso compartilhado de uma mesma plataforma de hardware

Um *MMV* é um componente de *software* que provê um conjunto de interfaces virtuais aos sistemas operacionais clientes. O conjunto de interfaces providas ao cliente e a instância deste em execução caracterizam uma máquina virtual.

2.1.3 Classificação de *MMVs*

Os *MMVs* podem ser classificados em três categorias: emulação, paravirtualização e virtualização nativa. As diferenças entre elas são basicamente o nível de abstração provido pela categoria.

Na **emulação**, simulação completa do sistema, ou ainda virtualização completa com recompilação dinâmica, a máquina virtual simula o hardware por completo. Com isso, sistemas operacionais para CPU diferentes, sem modificação alguma, podem ser executados. Um exemplo poderia ser o *GNU/Linux* IA32 sendo executado sobre o VMware em um Mac OS X sobre a plataforma física PowerPC. Como ocorre a emulação, um aspecto negativo é o custo computacional dessa simulação do hardware, que muitas vezes pode ser bastante onerosa. A figura 2.6 apresenta um exemplo de emulação. No caso, o *MMV* VMware Server está executando sobre um sistema *GNU/Linux* e emulando um ambiente de hardware para um sistema operacional cliente, FreeBSD. O usuário, além de poder utilizar o ambiente do FreeBSD, pode continuar utilizando em paralelo normalmente o seu sistema *GNU/Linux*.

Com **paravirtualização** a máquina virtual não simula dispositivos de hardware. Ao

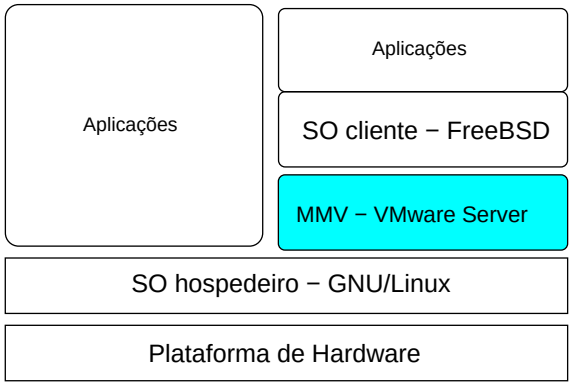

Figura 2.6: Emulação, simulação completa do sistema

invés disso, ela oferece uma *API* especial que leva a modificações nos sistemas operacionais. Um dos pontos negativos dessa abordagem é o fato de o sistema operacional ser modificado. Essa pode ser uma política pouco interessante, pois sabe-se que um sistema operacional tradicional é um conjunto de componentes de software complexo e difícil de ser cuidadosamente mantido. Trazer mais modificações, principalmente se elas causarem um impacto significativo na arquitetura do sistema operacional, pode ser uma solução pouco atrativa para desenvolvedores e mantenedores desses sistemas.

Um exemplo de paravirtualização é o Xen [6, 15]. Ele consiste em uma solução que permite o compartilhamento de recursos computacionais de forma significativamente mais eficiente quando comparado com soluções baseadas em emulação. A figura 2.7 ilustra a arquitetura paravirtualizada do Xen. A cópia do sistema operacional em execução no denominado domínio 0 [1] é responsável pela parte de gerenciamento das demais cópias dos sistemas operacionais. A camada Xen é uma *API* que controla a interface entre os sistemas operacionais e o hardware propriamente dito.

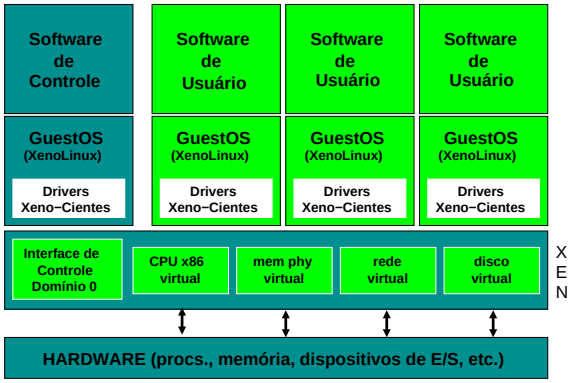

Figura 2.7: Paravirtualização [7]

[1] Domain0

Por fim, a **virtualização nativa**, ou virtualização completa, leva em conta que a máquina virtual simula apenas o hardware necessário para permitir que um sistema operacional, sem modificações, seja executado de forma isolada. Nesse caso, o sistema operacional deve ser projetado para o mesmo tipo de *CPU* do sistema hospedeiro. O termo virtualização nativa é também utilizado algumas vezes para designar a assistência de hardware que é utilizada através de tecnologia de virtualização. Uma ilustração da virtualização nativa pode ser vista na figura 2.8. Alguns componentes de hardware são emulados por completo, enquanto que outros, como os de entrada e saída, são utilizados de forma compartilhada com o sistema hospedeiro. Além disso, alguns componentes sequer precisam ser emulados, pois são nativamente criados para o suporte a determinadas funcionalidades utilizadas por máquinas virtuais.

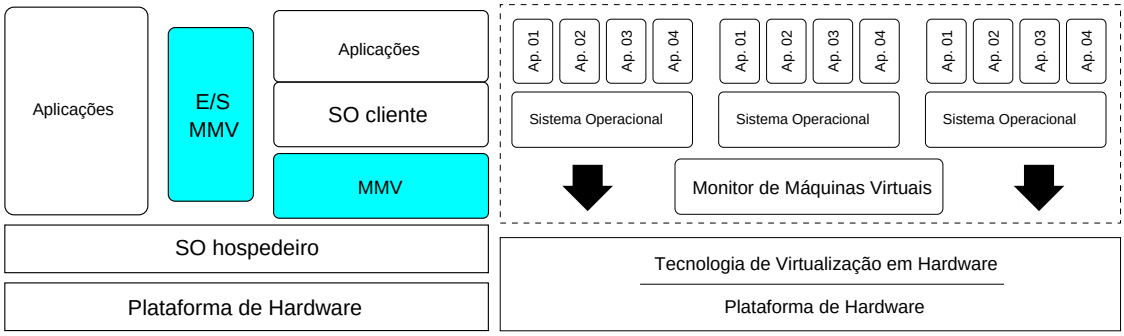

Figura 2.8: Virtualização nativa, ou virtualização completa, ou virtualização híbrida

2.2 Aplicações Atuais de Monitores de Máquinas Virtuais

As aplicações de virtualização, em especial ambientes criados com o uso de *MMVs* e *MVs*, são bastante variadas. Muitas áreas de ensino, pesquisa e produção têm utilizado essas novas tecnologias para resolver problemas diversos. Eis algumas áreas que estão aplicando a tecnologia de virtualização para resolver ditos problemas: **Computação de Alto Desempenho** [44]; **Redes Locais e Distribuídas** [19, 19]; **Desenvolvimento e Testes de Software** [17, 30]; **Segurança** [16, 66, 32, 47]; **Ensino** [2, 35]; **Compartilhamento de Recursos** [37, 44]; **Computação Autonômica** [34, 37, 34, 13]; entre diversos outros exemplos de aplicação atual dos *MMVs* [28].

A seguir são discriminadas aplicações de ambientes virtuais em cenários de redes locais, redes distribuídas e atividades de ensino. Em todos os casos, a virtualização traz benefícios para a criação e manutenção de ambientes computacionais que dificilmente

seriam viáveis sem a disponibilidade e utilização de virtualização.

Redes Locais e Distribuídas. Ferramentas de virtualização podem ser utilizadas em laboratórios de redes para simplificar e reduzir os custos de operacionalização e gerenciamento [19]. Em particular, ferramentas como Virtual Network User Mode Linux (*VNUML*) [19] podem ser utilizadas para construir cenários virtuais complexos de redes.

As demandas de um laboratório de redes, para cursos como administração de redes, são peculiares, pois os alunos precisam ter acesso a sistemas e ferramentas como *DNS*, *DHCP*, *HTTP*, *SMTP*, tabelas de rotas, filtros de pacotes e `tcpdump`. O maior problema desse contexto é o fato de os alunos precisarem de acesso de administrador para lidar com esses sistemas e ferramentas. Logo, ferramentas como *VNUML* emergem como uma alternativa perspicaz, capaz de prover um ambiente virtual controlado para laboratórios de redes de computadores.

Ensino. Arquitetura de computadores é uma das disciplinas da Computação que tem por objetivo apresentar as diferenças e similaridades entre estilos arquiteturais de processadores. Um dos problemas enfrentados por muitos cursos é a disponibilidade de recursos físicos para realização de experimentos e testes por parte dos alunos. Uma solução para esse problema pode ser o uso de máquinas virtuais [2]. Neste cenário, os estudantes utilizam os *assemblers* virtuais e executam os programas nas correspondentes máquinas virtuais, podendo, assim, acompanhar a execução dos programas passo-a-passo, em velocidade normal.

Disciplinas de sistemas operacionais ensinam muito mais aos estudantes quando oportunizam experiências práticas em projetos de *kernel* com sistemas operacionais reais [35]. Tornar práticas e viáveis essas experiências pode ser uma tarefa um pouco complicada, pois o desenvolvimento de núcleos de sistemas operacionais envolve a disponibilidade de equipamentos e ambientes de desenvolvimento. Graças às novas técnicas de virtualização, como Vmware e Qemu, as dificuldades podem ser suprimidas através da criação de um ambiente virtual para o desenvolvimento, experimento e teste de sistemas operacionais. Esta solução é adotada na prática no departamento de Ciência da Computação da universidade de Columbia [35]. A idéia foi criar um laboratório virtual para o desenvolvimento e teste de sistemas operacionais. A solução adotada foi utilizar a tecnologia de virtualização da Vmware. Os alunos têm acesso remoto às máquinas virtuais, onde estão os disponíveis as ferramentas de desenvolvimento e teste de sistemas operacionais.

Esse ambiente viabiliza projetos de sistemas operacionais e possibilita, de forma simples e prática, boas experiências de "mãos-na-massa" aos estudantes.

2.3 *Software* de Virtualização

As próximas seções discriminam alguns dos principais *MMVs* disponíveis no mercado [22], seja de forma comercial, ou como *software* livre. Como poderá ser observado, cada *MMV* possui algumas particularidades que o diferencia dos demais. Esse cenário caracteriza-se por uma multiplicidade grande, altamente heterogênea, de *software* para gerenciar máquinas virtuais.

2.3.1 Software de Máquinas Virtuais de Hardware

Xen. É um *MMVs*, que utiliza a técnica de paravirtualização, para arquiteturas x86 [6]. Ele permite que múltiplos sistemas operacionais, comumente utilizados, compartilhem *hardware* convencional com eficiência e segurança. Para chegar a esse ponto, é provida uma abstração idealizada de máquina virtual. Sistemas operacionais como *GNU/Linux* e FreeBSD podem ser portados para utilizar essa abstração idealizada com esforço mínimo [6].

KVM. Kernel-based Virtual Machine[70] é uma infraestrutura de software no *kernel* do Linux para suporte a virtualização. Em linhas gerais, o KVM é uma combinação entre um *driver* de virtualização Linux em modo *kernel* e um programa em nível de usuário (Qemu) modificado, que faz a interface com o *driver*.

WMware. Na versão Workstation[1], pode ser considerado um dos *MMVs* que surgiram devido à necessidade de virtualizar sistemas operacionais para arquiteturas x86 sem que estes tenham que sofrer alguma modificação. O Vmware utiliza tradução binária para prover virtualização x86 completa [1].

Denali. O principal objetivo do projeto Denali [66, 67] é prover suporte em nível de sistema para a execução de múltiplos serviços de Internet, pouco confiáveis, em uma mesma infraestrutura física. No caso do Denali, o núcleo é denominado *kernel* de isolamento. Ele é o responsável pelo isolamento entre os sistemas, segmentando os domínios de aplicações pouco confiáveis.

Qemu. Qemu [8, 54, 50] é um emulador de máquina. Ele é capaz de executar sistemas operacionais-alvos (como Windows ou Linux) e todas as suas aplicações em uma

máquina virtual. Qemu é uma aplicação que pode ser executada em diferentes sistemas operacionais hospedeiros, como Linux, Windows e Mac OS X. As *CPUs* da máquina alvo e da hospedeira podem ser diferentes [8].

Parallels. Parallels Workstation utiliza a tecnologia de *hypervisor* [71], que é uma pequena camada de software entre o computador hospedeiro e o sistema operacional primário. O *hypervisor* controla diretamente alguns dos recursos de hardware da máquina e prove uma interface para o acesso deles tanto pelo sistema operacional primário, quanto pelos monitores de maquinas virtuais.

Quadro Comparativo

A tabela 2.1 apresenta um quadro comparativo de características dos sistemas de virtualização apresentados no decorrer da seção 2.3.1. Além disso, são incluídas algumas características complementares, como arquitetura de processador suportada. A maioria dos dados, com exceção da coluna *Modo de operação* e alguns detalhes que foram ajustados e/ou complementados, foram extraídos de [68].

Nome	Criador	CPU suportada	CPU emulada	SO hospedeiro	SO cliente	Método de operação	Licença
Xen	Universidade de Cambridge	Intel x86 e AMD64	Intel x86 e AMD64	NetBSD, Linux, Solaris	Linux, NetBSD, FreeBSD, OpenBSD, Solaris, entre outros	paravirtualização e porte ou virtualização por hardware	GPL
KVM	KVM	Intel e ADM x86 com suporte a virtualização	x86/AMD64	Linux	Linux e Windows	virtualização em *kernel*	GPL2
VMware	VMware	Intel x86 e AMD64	Intel x86 e AMD64	Linux e Windows	DOS, Windows, Linux, FreeBSD, Netware, Solaris e Aplicações Virtuais	paravirtualização e virtualização	Proprietária
Denali	Universidade de Washington	Intel x86	Intel x86	Denali	IIwaco e NetBSD	paravirtualização e porte	não encontrada
Qemu	Frabrice Bellard e outros desenvolvedores	Intel x86, AMD64, IA-64, PowerPC, Alpha, SPARC 32 and 64, ARM, S/390, M68k	Intel x86, AMD64, ARM, SPARC 32 and 64, PowerPC, MIPS	Windows, Linux, OS X, Solaris, FreeBSD, OpenBSD, BeOS	Linux, Windows, FreeBSD, Solaris, entre outros	recompilação dinâmica	GPL/LGPL
Parallels	Parallels Inc.	Intel x86 e Intel VT-x	Intel x86	Linux e Windows	Windows, Linux, FreeBSD, OS/2, eComStation, MS-DOS, Solaris	virtualização e *hypervisor* leve	Proprietária

Tabela 2.1: Software de máquinas virtuais de hardware: sumário comparativo

Como pode ser observado na tabela, existem diferentes características, como arquiteturas de processador suportadas, específicas a determinados sistemas de virtualização. Logo, a escolha do sistema também dependerá da infraestrutura de hardware disponível.

Outros detalhes sobre os *MMVs* apresentados nesta seção podem ser vistos no relatório técnico que precedeu esta dissertação [28]. O trabalho detalha em maior profundidade decisões arquiteturais, bem como aplicações, dos softwares de máquinas virtuais de *hardware*.

2.3.2 Software de Virtualização em nível de Sistema Operacional

User-mode Linux. Um *kernel* é um programa como outro qualquer. Essa é a filosofia adotada pelo User-mode Linux (UML) [41, 14], que nada mais é que um núcleo de sistema operacional executando como um outro programa qualquer [26].

Virtuozzo. O objetivo do Virtuozzo [56, 48] é criar ambientes virtuais isolados, ou contêiners, em um único servidor físico, com uma única instância de sistema operacional [51]. Algumas características providas pelo Virtuozzo incluem níveis altos de densidade, desempenho e gerenciamento.

Linux-VServer. Assim como o Virtuozzo e o FreeBSD Jails, o Linux-VServer [52, 50, 21, 36] oferece virtualização baseada em contêiners [56]. O conceito básico por trás do Linux-VServer consiste em uma solução capaz de segmentar o espaço de usuário em unidades distintas, às vezes denominadas de servidores virtuais privados (*SVP*), de forma que cada *SVP* seja semelhante a um servidor real para os processos nele contidos [52].

OpenVZ. OpenVZ [22] é uma solução de virtualização em nível de sistema operacional, construído sobre o Linux. OpenVZ é capaz de criar servidores virtuais privados (*SVPs*), isolados, em um único servidor físico. Os *SVPs* podem ser gerenciados de forma independente. Essa característica provê liberdade e independência. Um *SVP* pode ser reiniciado, prover acesso em nível de super usuário, contas de usuário, endereços *IP*, memória, processos, sistemas de arquivos, aplicações, bibliotecas de sistema, arquivos de configuração e todas as demais infraestruturas de software que um sistema operacional convencional é capaz de prover.

FreeBSD Jails. O FreeBSD Jail [65, 32, 25] provê mecanismos que permitem o particionamento do ambiente do sistema operacional. Contudo, um dos objetivos das Jails é manter a simplicidade de um modelo Unix baseado no "root".

Solaris Containers. Um contêiner de recursos é uma entidade abstrata do sistema operacional que, logicamente, contém todos os recursos de sistema que estão sendo utilizados por uma aplicação para realizar uma atividade em particular [5]. Como exemplo,

poderia ser citada uma conexão *HTTP* gerenciada por um servidor Web. Os recursos para gerenciar a conexão incluem tempo de *CPU* destinado à conexão, objetos de *kernel* como *sockets*, blocos de controle de protocolo e *buffers* de rede utilizados pela conexão. Todos esses recursos ficariam logicamente associados ao contêiner do servidor Web.

FreeVPS. FreeVPS é uma solução de virtualização livre [39, 69] baseada em três conceitos principais: contextos, espaços de nomes (de disco) privados e virtualização da pilha de rede. Os contextos têm como principal função proteger os servidores virtuais entre eles. Um processo executando em um determinado contexto pode utilizar apenas os recursos associados ao contexto, como arquivos, espaço em disco e endereço de rede (*IP*).

FVM. O Feather-weight Virtual Machine (FVM) [76] é uma arquitetura em nível de sistema operacional para criação e gerenciamento de máquinas virtuais para aplicações Windows. As máquinas virtuais FVM compartilham todos os recursos possíveis da máquina hospedeira, mantendo isolamento entre elas e entre a máquina hospedeira. Um dos aspectos que recebeu bastante atenção no projeto do FVM foi a minimização do tempo de latência de invocação de uma nova máquina virtual. Além disso, outra preocupação foi relativa ao suporte a um grande número de máquinas virtuais através da minimização dos requisitos de recursos de uma máquina virtual.

Quadro Comparativo

A tabela 2.2 apresenta um quadro comparativo de características dos sistemas de virtualização apresentados no decorrer da seção 2.3.2. Além disso, são incluídas algumas características complementares, como arquitetura de processador suportada. A maioria dos dados, com exceção da coluna *Modo de operação* e alguns detalhes que foram ajustados e ou complementados, foram extraídos de [68].

Como pode ser observado na tabela, existem diferentes características, como sistemas operacionais suportados, específicas a determinados sistemas de virtualização. Logo, a escolha do sistema também poderá depender do sistema operacional requerido pelas aplicações que serão utilizadas.

Outros detalhes sobre os *MMVs* apresentados nesta seção podem ser vistos no relatório técnico citado na seção anterior [28]. O trabalho detalha em maior profundidade decisões arquiteturais, bem como aplicações, dos softwares de máquinas virtuais de *hardware*.

Nome	Criador	CPU suportada	CPU emulada	SO hospedeiro	SO cliente	Método de operação	Licença
User Mode Linux	Jeff Dike e outros desenvolvedores	Intel x86 e PowerPC	Intel x86 e PowerPC	Linux	Linux	porte	GPL2
Virtuozzo	SWsoft	Intel x86, AMD64, IA-64, PowerPC64	Intel x86, AMD64, IA-64	Linux e Windows	Diversas distribuições GNU/Linux e Windows	virtualização em nível de sistema operacional	GPL
Linux-VServer	Projeto comunitário	Intel x86, AMD64, IA-64, Alpha, PowerPC/64, PA-RISC/64, SPARC/64, ARM, S/390, SH/66, MIPS	compatível	Linux	Várias distribuições GNU/Linux	virtualização em nível de sistema operacional	GPL2
OpenVZ	Projeto comunitário, suportado pela SW-soft	Intel x86, AMD64, IA-64, PowerPC64, SPARC64	Intel x86, AMD64, IA-64, PowerPC64, SPARC64	Linux	Diversas distribuições GNU/Linux	virtualização em nível de sistema operacional	GPL
FreeBSD Jails	FreeBSD	Intel x86	compatível	FreeBSD	FreeBSD	virtualização em nível de sistema operacional	FreeBSD
Solaris Containers	Solaris	SPARC 32 e 64	SPARC 32 e 64	Solaris	Solaris	virtualização em nível de sistema operacional	proprietária
FreeVPS	PSoft	Intel x86 e AMD64	compatível	Linux	Várias distribuições GNU/Linux	virtualização em nível de sistema operacional	GPL2
FVM	Yang Yu e outros desenvolvedores	Intel x86	Intel x86	Windows	Aplicações Windows	virtualização em nível de sistema operacional	não encontrada

Tabela 2.2: Software de virtualização em nível de sistema operacional: sumário comparativo

2.4 *Virtual Appliances*

2.4.1 Conceituação

Um *virtual appliance* é um pacote de dados capaz de encapsular um sistema operacional e aplicações. Estas podem ser o mais variadas possível, indo desde simples ferramentas do sistema até servidores de aplicação, como `Apache2` e *JAS*.

Segundo Ying [64], um *virtual appliance* é um encapsulamento de um estado de uma máquina virtual. É um pacote pré-configurado e pré-otimizado para uma aplicação ou solução em específico.

Segundo Sapuntzakis [46], um *virtual appliance* é o estado de um *appliance* real (o conteúdo dos discos do *appliance*) bem como a descrição do *hardware* (exemplo: dois adaptadores Ethernet, 256 *MB* de *RAM*, dois discos rígidos, etc.). Como um *virtual appliance* é somente dados, ele pode ser distribuído eletronicamente, o que facilita o compartilhamento de soluções entre os usuários finais.

Uma das grandes vantagens dos *appliances* é a facilidade de administração [45]. Um *virtual appliance* (*VAP*) é como um *appliance* físico, mas sem o *hardware*, sendo assim,

um *VAP* é como *software* e pode ser armazenado e comercializado eletronicamente. Essa característica torna os *VAPs* uma alternativa atrativa para resolver problemas e demandas heterogêneas de sistemas e ambientes computacionais, naturalmente heterogêneos.

2.4.2 Composição

Um *VAP* é composto por um conjunto de elementos básicos que formam um pacote de dados. No contexto deste trabalho, este pacote é composto por um arquivo de configuração, dependente de *MMV*, e um disco virtual, que pode ser materializado em um arquivo. Estes elementos são ilustrados na figura 2.9.

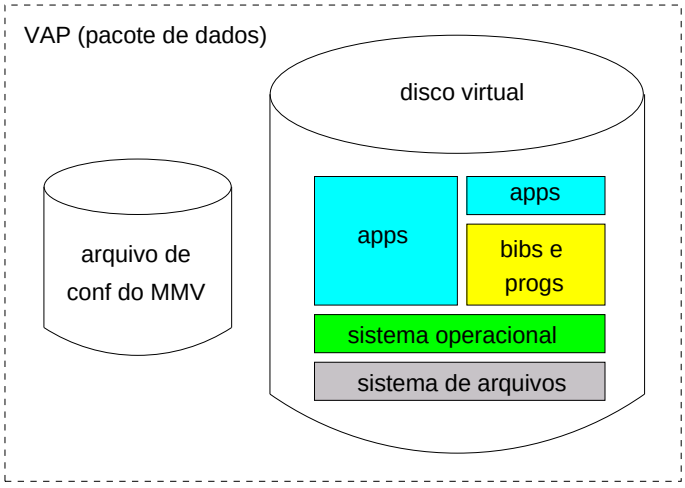

Figura 2.9: Conteúdo de um *VAP*

O conteúdo do disco virtual é organizado em quatro camadas. A primeira e a segunda camadas são compostas pelo sistema de arquivos e pelo sistema operacional, respectivamente. As bibliotecas e as ferramentas de sistema formam a terceira camada. A quarta camada é composta pelas aplicações em nível sistema e ou usuário. Neste último nível é onde ficam as soluções e ferramentas que caracterizam a existência do *VAP*, ou seja, criam o ambiente ou a plataforma necessária à resolução de algum problema em específico.

2.4.3 Exemplos e Aplicações

Os *VAPs* possuem uma gama variada de aplicações. A seção 2.2 apresenta uma relação de aplicações de *MMVs*. Em praticamente todos esses casos podem ser aplicados *VAPs*, sendo que cada *VAP* constituiria-se na solução pré-configurada e pré-otimizada para facilitar o gerenciamento, a manutenção e a administração de uma das aplicações de *MMVs*.

Um exemplo prático, normalmente um problema clássico dos mais diversos tipos de instituições e organizações, é o gerenciamento de laboratórios de informática. A variabilidade de aplicações demandadas pelos usuários é bastante grande e crescente, o que torna complicada a tarefa de administração desses laboratórios. Os *VAPs* podem representar uma alternativa para amenizar esse problema. Considerando o contexto de uso de *VAPs*, um laboratório de informática precisaria apenas possuir um sistema base, hospedeiro, como um *GNU/Linux*, com *MMVs* disponíveis. O usuário, a partir de um repositório global de *VAPs*, poderia escolher o sistema (*VAP*) desejado e solicitar a instanciação do mesmo. Esse ambiente é ilustrado na figura 2.10.

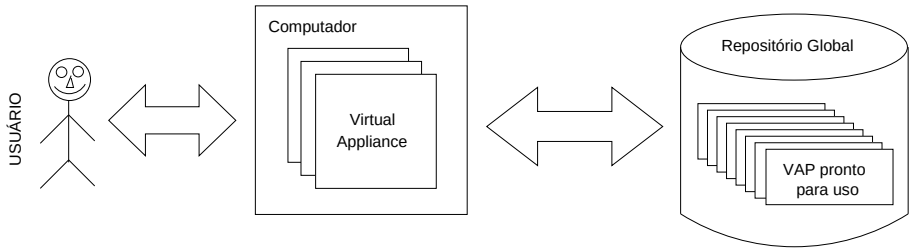

Figura 2.10: Gerenciando um laboratório de computação com *VAPs*

O uso de *VAPs* em laboratórios de informática oferece agilidade e flexibilidade aos administradores e usuários. Os administradores podem rápida e facilmente criar, manter e disponibilizar novos *VAPs*, enquanto que os usuários possuem flexibilidade de escolher o *VAP* (ambiente) desejado e a agilidade no atendimento as suas requisições (criação de novos ambientes ou soluções). Ademais, o uso de *VAPs* diminui também alguns problemas tradicionais, como *vírus/worms/trojans* e sistemas falhos devido ao mau uso, como aqueles gerados pela remoção de arquivos de sistema, excesso de dados na partição do sistema operacional e instalação de aplicações não autorizadas. Como o *VAP* é copiado da *cache* local ou do repositório global, a cada vez que o usuário solicita a sua ativação o sistema está sempre limpo, idêntico ao original instalado pelo administrador do laboratório.

Uma das grandes vantagens do uso de *VAPs* é que as aplicações e soluções podem ser facilmente transportadas para diferentes ambientes e utilizadas por usuários diversos, sem necessidade de grande re-trabalho, que normalmente permeia a constituição manual e individual dos sistemas necessários a um determinado ambiente ou contexto de uso. Essa vantagem dos *VAPs* traz ganhos globais para todos, pois os usuários podem concentrar-se mais em resolver problemas novos ao invés de gastar seu tempo recriando soluções, plataformas e ambientes computacionais.

2.4.4 Heterogeneidade de Ambientes

Na seção 2.4.3 é apresentado um contexto de laboratórios de informática, que tipicamente demanda requisitos como heterogeneidade de sistemas.

Considerando o uso de *VAPs*, a heterogeneidade pode se manifestar também de outras formas. De fato, a heterogeneidade pode ser relativa tanto aos *VAPs*, que podem ser compostos pelos mais variados tipos de sistemas e possuir as mais variadas finalidades, quanto aos *MMVs*, que podem ser necessários para viabilizar a inicialização de determinados *VAPs*. **Essas são duas das abordagem de heterogeneidade que fazem parte do foco deste trabalho**.

Uma outra abordagem do trabalho é lidar com cenários que demandam simultaneamente múltiplos *VAPs* para criar o ambiente final dos usuários. Exemplos desse tipo de cenário incluem laboratórios de pesquisa, como nas áreas de computação paralela e distribuída, ambientes de produção, como *data centers*, e ambientes de desenvolvimento de *software*. Em todos esses casos, normalmente é necessário a existência de ambientes do tipo servidor e cliente, cujas funções e usos diferem de forma significativa. Um servidor de aplicação de uma equipe de desenvolvimento é caracterizado como um ambiente destinado à execução e teste prático das soluções em desenvolvimento. Já um ambiente de desenvolvimento é caracterizado como um sistema que possui as ferramentas necessárias e úteis ao desenvolvimento da solução ou aplicação. Como pode ser observado, as funções são distintas, porém, há uma correlação entre elas, pois o fechamento do projeto (entrega do produto em desenvolvimento) depende da disponibilidade do sistema servidor e do sistema cliente.

Outro exemplo, considerando um laboratório de pesquisa, é um aglomerado de computadores, onde existem tipicamente dois tipos de sistemas. Um deles é o servidor de dados, máquina que armazena os dados e as credenciais dos usuários e permite o acesso aos demais sistemas, denominados nós. Os nós são compostos por sistemas clientes, tradicionalmente caracterizados como unidades de processamento disponíveis para a solução dos problemas dos usuários. Instalar e manter ambientes desse gênero requer a disponibilidade de, pelo menos, dois sistemas distintos. Em um contexto de um aglomerado de computadores virtual, aumentar o número de nós de maneira automática e dinâmica pode ser algo desejável. Isso pode ser atingido através de uma solução simples para o gerenciamento de *VAPs*, como proposto neste trabalho.

A solução proposta neste trabalho tem por objetivo lidar com os diferentes contextos de heterogeneidade mencionados nesta seção. Para isso, sua arquitetura é composta por componentes e funcionalidades que proporcionam um bom grau de flexibilidade e adaptabilidade para o gerenciamento de diversificados tipos de ambientes de computação que fazem uso de soluções de virtualização de sistemas.

2.4.5 Trabalhos Correlatos

Há trabalhos correlatos cujo objetivo também é prover alternativas de infraestrutura computacional baseadas em virtualização, mais especificamente *virtual appliances*. Porém, o foco dessas soluções é mais específico e, portanto, menos geral.

O *outsource* de *data centers* tem emergido como uma alternativa viável e interessante para hospedar aplicações e serviços de diferentes organizações e companhias. Neste contexto, um ponto crítico é o isolamento das diferentes aplicações do *data center*, alocando os recursos dinamicamente entre elas. Essa alocação de recursos sob demanda precisa ser eficiente e, para resolver este problema, Wang [64] propõe um *framework* autonômico para o provisionamento de recursos, baseado em *virtual appliances*, em *data centers* virtualizados. A solução proposta tem como foco a manutenção das aplicações isoladas de um *data center* de maneira a distribuir e otimizar ao máximo o uso dos recursos, visando sempre ao atendimento dos acordos de níveis de serviços (*SLAs*). Neste sentido, o trabalho propõe um modelo de otimização, baseado em uma solução analítica inovadora, para o provisionamento dinâmico de recursos às máquinas virtuais do ambiente.

A solução proposta é composta por quatro ambientes: execução, planejamento, análise e monitoramento. O ambiente de execução é formado por um repositório de *virtual appliances*, um gerenciador de *appliances* (IBM Tivoli Provisioning Manager), um *pool* de servidores heterogêneos e um atuador de reconfiguração. Em um primeiro momento, o *virtual appliance* é carregado do repositório para a máquina-alvo. Num segundo momento, *scripts* e metadados são utilizados para customizar o *appliance*. Por fim, o *VAP* é ativado como um servidor no novo ambiente.

A ambiente de execução baseia-se em uma solução homogênea de virtualização (exemplo: VMware ESX) sobre um ambiente de servidores físicos heterogêneos. Os *virtual appliances* que serão instanciados nesse contexto são diversos em termos de aplicações e finalidades, tendo potencialmente demandas dinâmicas heterogêneas de recursos. É

neste ponto que concentra-se a solução proposta, ou seja, produzir mecanismos e ciclos de reconfiguração e otimização no uso dos recursos disponíveis.

Uma vez que a virtualização quebra a dependência em relação ao *hardware* e isola as máquinas virtuais dos detalhes dos servidores físicos hospedeiros, imagens virtuais podem ser movidas de uma plataforma de hospedagem para outra [23]. Um dos maiores desafios do provisionamento automático de imagens virtuais é manipular o sistema operacional, a rede e as customizações específicas de aplicações [23, 24]. De maneira a simplificar as atividades necessárias para a clonagem de máquinas virtuais, o trabalho de He [23] propõe um processo de três fases, etapas através de cada atividade necessária. O processo é viabilizado através de um *framework* contendo uma solução de ativação e dois *scripts* de ativação para VMware ESX e Xen, respectivamente. A solução proposta tem por objetivo demonstrar a preparação e a ativação de imagens virtuais para servidores de aplicação IBM WebSphere.

A figura 2.11 ilustra o processo do *framework* de preparação e ativação automática de máquinas virtuais. O processo é constituído de três fases: 1) preparação; 2) criação e cópia; e 3) auto-ativação.

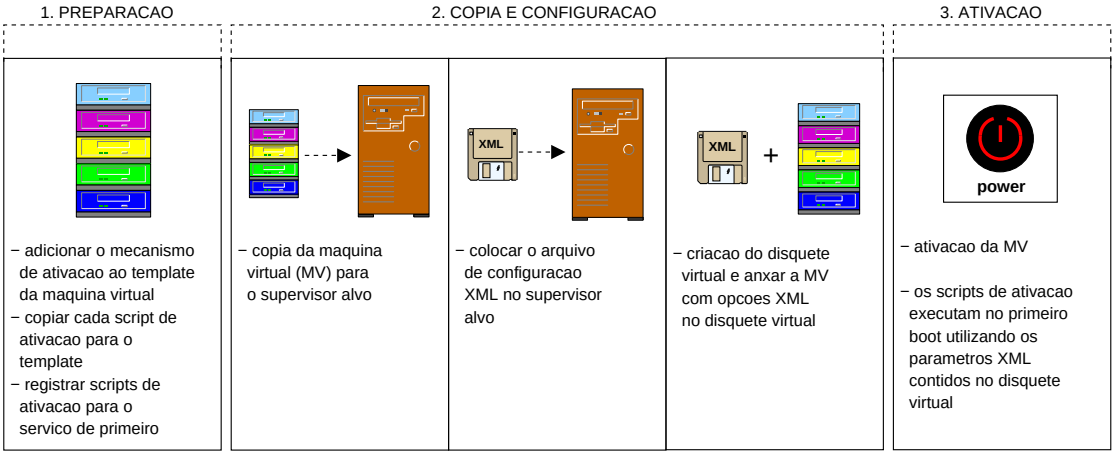

Figura 2.11: Processo de preparação e ativação automática de máquinas virtuais [23]

O processo todo de preparação e ativação é baseado em configurações *XML* armazenadas em um disquete virtual. Este é acoplado à imagem da *MV*. No primeiro *boot* do sistema da máquina virtual, os *scripts* de configuração e ativação serão executados, refinando e finalizando a configuração do sistema.

O modelo proposto e prototipado disponibiliza *scripts* para ativação automática de máquinas virtuais VMware e Xen. Esses *scripts* baseiam-se na existência de um disquete

virtual acoplado à máquina virtual, para que as devidas configurações sejam processadas na primeira inicialização do sistema.

Collective é um sistema que entrega *desktops* gerenciados a usuários de computadores de pessoais (*PCs*) [11]. Na arquitetura do sistema, administradores de sistema são responsáveis pela criação e manutenção dos ambientes de *desktop*, ou *virtual appliances*, o que inclui o sistema operacional e todas as aplicações instaladas. O *PC*, no contexto do Collective, executa *software*, denominado *transceiver* de *virtual appliances*, que faz *cache* e executa localmente as cópias mais recentes dos *virtual appliances*, e continuamente realiza *backup* dos dados dos usuários em um repositório da rede. Esse modelo provê a vantagem do gerenciamento central, como melhor segurança e menor custo de gerenciamento.

A figura 2.12 ilustra a arquitetura proposta pelo Collective. O sistema baseia-se em dois repositórios globais, centrais, um para armazenamento de *VAPs* e outro para os dados dos usuários. Os *transceivers* irão executar nos *PCs*, utilizando os *virtual appliances* contidos no repositório central. O meio proposto e utilizado para isso é a Internet. Com isso, um dos focos da solução é na eficiência na transmissão dos dados dos *virtual appliances* através da rede.

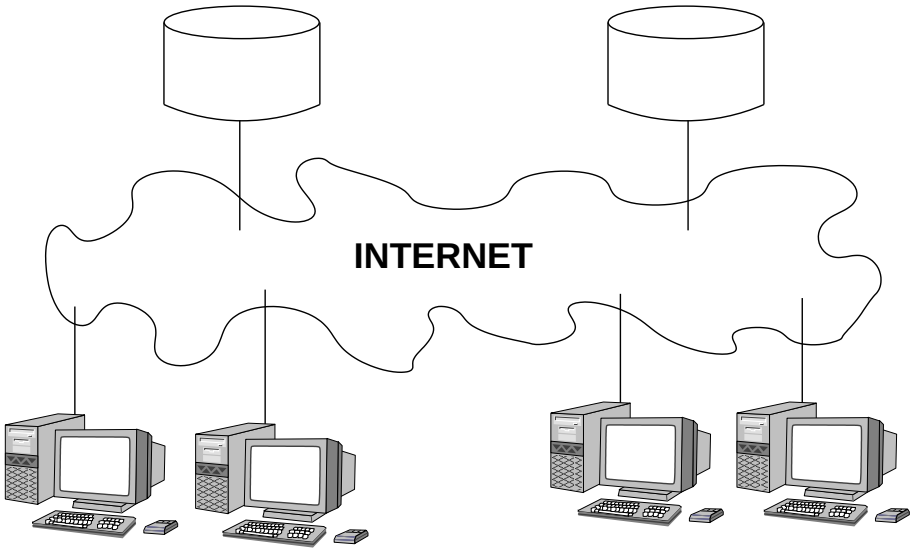

Figura 2.12: Arquitetura do Collective [11]

Um *appliance* encapsula o estado de um computador em uma máquina virtual que consiste no seguinte [11]:

- os discos de sistema são criados pelo administradores e contêm o sistema operaci-

onal e as aplicações do *appliance*. Como parte do modelo, o conteúdo dos discos é tornado idêntico ao conteúdo publicado pelo administrador a cada *boot* do sistema.

- os discos dos usuários mantém seus dados privados persistentes, como arquivos e configurações de usuários.
- o *backup* dos discos de dados temporários dos usuários não é realizado. Esses discos são utilizados para dados temporários como *caches* e arquivos temporários.
- uma imagem da memória mantém o estado de um *appliance* suspenso.

2.5 Considerações

Os monitores de máquinas virtuais representam uma das grandes tendências atuais da computação[20, 42]. Eles surgiram na década de 60 e 70[42, 74] e continuam causando um grande impacto nos sistemas de computação de um modo geral. Esse fato está atrelado principalmente às grandes evoluções de hardware que ocorreram durante as últimas décadas.

Neste capítulo foram apresentados os conceitos, aplicações e os principais monitores de máquinas virtuais disponíveis. Xen, VMware, FreeBSD-Jails, KVM, UML e OpenVZ são exemplos de mecanismos de virtualização capazes de estender o uso de infraestruturas de computação tradicionais através do compartilhamento de recursos. A gestão do uso dos recursos disponíveis é, de forma geral, realizada de maneira segura, isolada e flexível.

Com a evolução das ferramentas de virtualização emergiram novas aplicações. Elas são as mais variadas, indo desde ambientes de ensino até sistemas de linha de produção. Essa vasta gama de aplicações, brevemente explorada no decorrer do texto, demonstra ainda mais a tendência e o grande potencial dos monitores de máquinas virtuais. Em diferentes cenários reais eles são capazes de resolver desde problemas de segurança básicos, até problemas de desempenho e escalabilidade.

Apesar das soluções e pesquisas disponíveis, ainda há bastante espaço para investigações e desenvolvimento na área de virtualização [42, 55]. Segurança, desempenho, virtualização em nível de hardware e diferentes áreas de aplicação são alguns dos pontos que merecem maiores investigações.

Por fim, foram apresentados, também, os conceitos e algumas aplicações de *virtual appliances*, uma das novas tendências na área de virtualização de sistemas e ambientes de

computação. O uso de *VAPs* tem se proliferado entre usuários e empresas. Atualmente, já existem repositórios online [59, 57], tanto de fabricantes quanto da comunidade em geral, e há uma grande perspectiva de continuidade e aumento progressivo na distribuição e uso desses pacotes de sistemas pré-configurados e otimizados para determinados fins. Alguns fabricantes estão, inclusive, disponibilizando seus produtos na forma de *virtual appliances*. Por serem pacotes de dados, os *VAPs* podem facilmente ser compartilhados e distribuídos em redes locais, ou mesmo pela Internet, além de representar uma forma interessante de resolver diferentes tipos de problemas.

3 *FLEXVAPS*: APRESENTAÇÃO DA ARQUITETURA

Este capítulo descreve a proposta de solução, denominada *FlexVAPs*, para o gerenciamento de *VAPs* em ambientes heterogêneos. No contexto do presente trabalho, um ambiente heterogêneo é aquele formado por uma diversidade de monitores de máquinas virtuais e respectivas máquinas virtuais. Não há preocupação alguma com a heterogeneidade física, tradicional, das máquinas, pois uma vez disponível o monitor de máquinas virtuais, a ativação de máquinas virtuais, através de *virtual appliances*, torna-se transparente e independente do *hardware* físico. O objetivo é prover uma infraestrutura de software básica capaz de gerenciar ambientes diversos, com suas peculiaridades e características intrínsecas a um contexto em particular. Neste sentido, as próximas seções descrevem a idéia e a arquitetura da solução proposta.

3.1 Introdução

A proposta do trabalho é projetar uma arquitetura básica capaz de gerenciar conjuntos de *virtual appliances*, com o objetivo de facilitar o gerenciamento de ambientes computacionais virtuais heterogêneos. Uma alternativa de viabilizar uma arquitetura desse gênero é através do uso de *VAPs* e, respectivamente, de *MMVs*.

A figura 3.1 apresenta uma visão abstrata, sumarizada, da proposta de sistema modelada neste trabalho. Pode ser observado que existe um ambiente composto por um ou mais repositórios de *VAPs* e máquinas que recebem sob demanda (de acordo com as demandas dos usuários) *VAPs* para ativação, inicialização e execução.

O objetivo da solução é prover meios de atender às diferentes demandas dos usuários de maneira flexível e prática. A seguir são relacionados alguns dos requisitos, inerentes à administração e à utilização de soluções heterogêneas em laboratórios temáticos de informática, que a solução proposta pretende preencher.

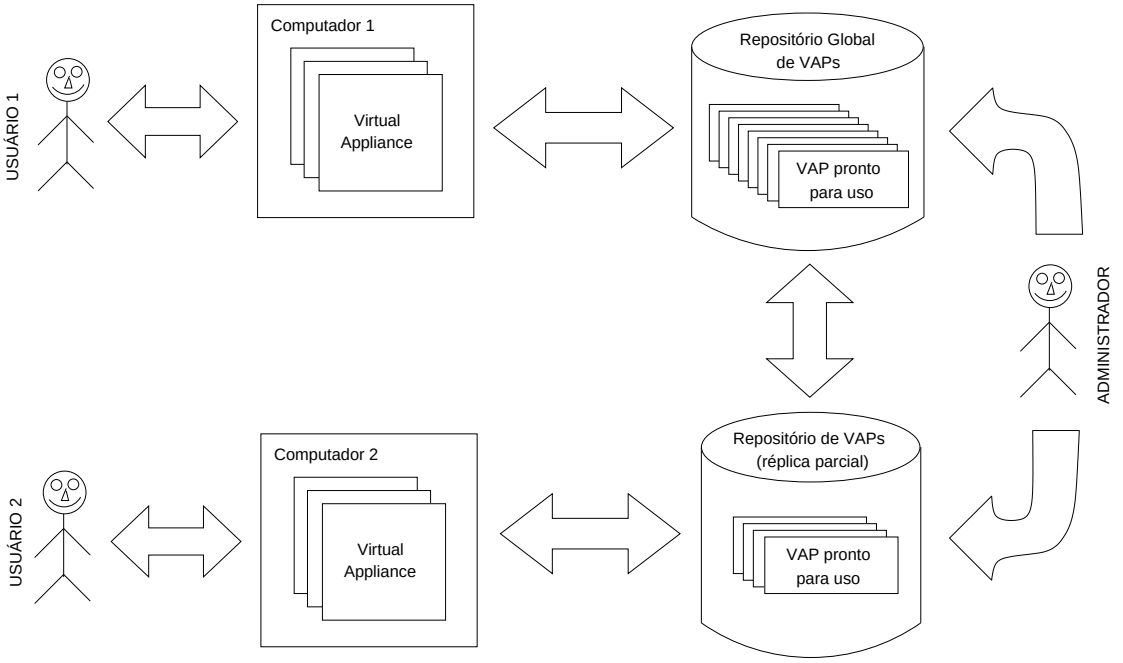

Figura 3.1: Infraestrutura de VAPs

- ser integrável a contextos de gerenciamento de redes, ou seja, preferencialmente, utilizar ferramentas e tecnologias conhecidas e utilizadas por administradores de sistemas;
- suportar variabilidade de *VAPs*, o que inclui diferentes *MMVs*, podendo atender às demandas dos mais diversos cenários de computação;
- permitir aos administradores do sistema a rápida publicação de *VAPs*;
- dispor de uma interface simples e prática aos usuários (um usuário deve ser capaz de selecionar e instanciar um *VAP* em poucos passos);
- permitir a instanciação de ambientes computacionais virtuais diversos, tornando possível o uso da solução em diferentes cenários;
- viabilizar o atendimento rápido das demandas encaminhadas pelos usuários, ou seja, um administrador do sistema deve conseguir rapidamente preparar e publicar um *VAP* com um sistema que atenda às necessidades do usuário.

Esses requisitos representam, em ambientes tradicionais, alguns dos principais empecilhos aos administradores e usuários dos laboratórios e ambientes temáticos da computação. Ambientes diversos, como sistemas para portais *WWW*, serviços de rede em geral, entre outras coisas, podem ser atendidos pela solução proposta neste trabalho.

3.2 Arquitetura da Solução

3.2.1 A Idéia

Os componentes básicos da solução, apresentados nas seções 3.2.2 e 3.2.3, reúnem características como o suporte a ambientes e soluções de virtualização (*MMVs*) diversificados e diferentes possibilidades de atender às demandas dos usuários através de *VAPs* e *MMVs* mais adequadas às suas demandas, que tornam possível o gerenciamento e a aplicação da solução aos mais diversos tipos de contextos e ambientes computacionais.

3.2.2 O Papel dos *Virtual Appliances*

Os componentes primordiais da solução são denominados de *Virtual Appliances* (*VAPs*). Um *VAP* é um conjunto de software com uma determinada finalidade, encapsulado como um pacote de dados. As figuras 3.2, 3.3 e 3.4 representam as possibilidades de configuração de um *VAP*. No primeiro caso, o *VAP* é constituído de um sistema operacional (*guest*), um conjunto de bibliotecas e aplicações. Estas podem ser variadas, indo desde simples editores de texto até soluções de software mais complexas, como pacotes de Inteligência de Negócio (*Business Inteligence - BI*) [38]. Na seqüência, figura 3.3, é representado um caso similar, porém, considerando a utilização de *MMVs* paravirtualizados. Por fim, o último caso (figura 3.4) considera que o próprio *MMV* está integrado ao *VAP*, gerando apenas dependência em relação ao sistema operacional hospedeiro na(s) máquina(s) de destino.

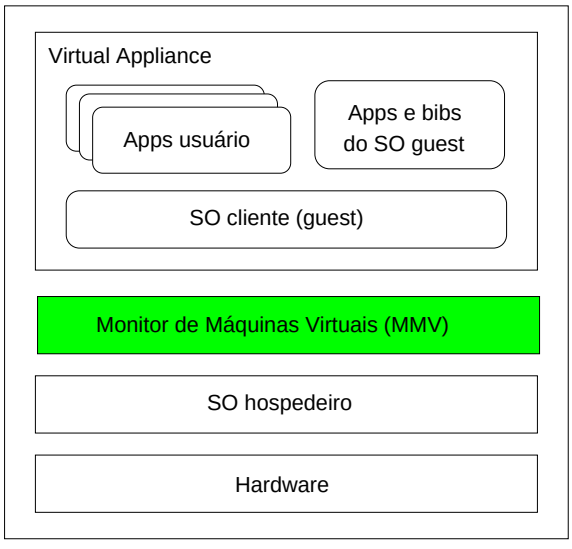

Figura 3.2: Virtual Appliance 1

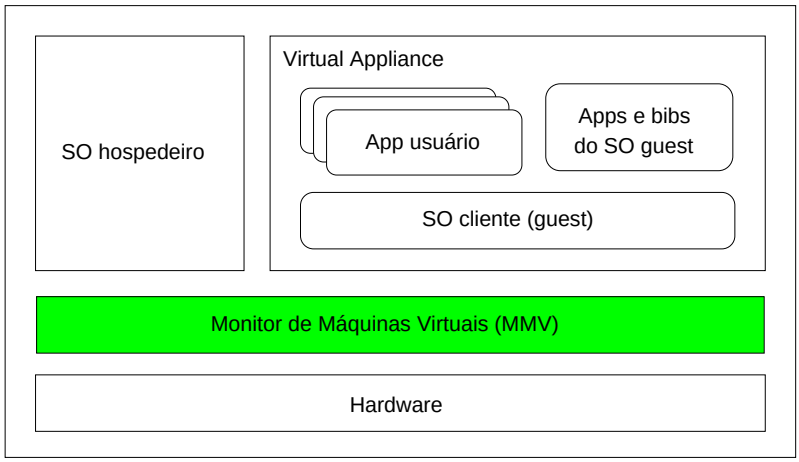

Figura 3.3: Virtual Appliance 2

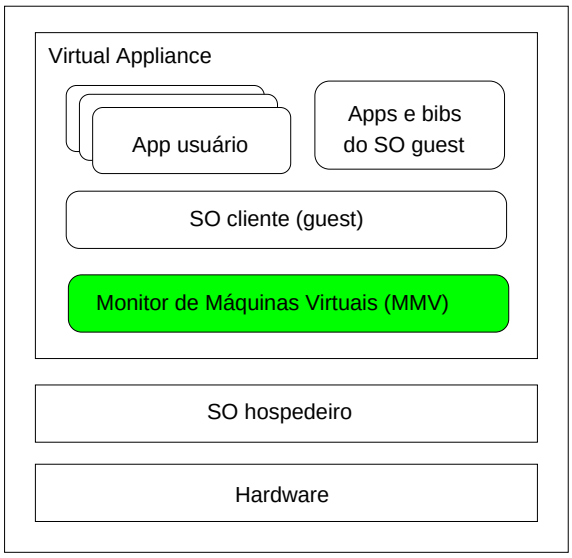

Figura 3.4: Virtual Appliance 3

No *FlexVAPs*, um *VAP* é descrito através de arquivos de configuração *XML*. Essa descrição é realizada pelo administrador do sistema a partir de uma solicitação de algum usuário. Uma vez descrito e preparado, o *VAP* vai para o repositório. O usuário, por sua vez, irá fazer o *download* do *VAP*, como ilustrado na figura 3.1, ativando-o no sistema local.

A figura 3.5 apresenta uma máquina física no contexto da solução proposta neste trabalho. As *VAPs* podem ser utilizadas por usuários locais à máquina ou por usuários remotos. Neste último caso, como exemplos, podem ser citadas a criação de um ambiente de rede virtual para teste de protocolos de rede e a criação de um ambiente virtual destinado à computação paralela e distribuída.

A heterogeneidade gerenciada pelo sistema proposto está nos *VAPs*, que podem de-

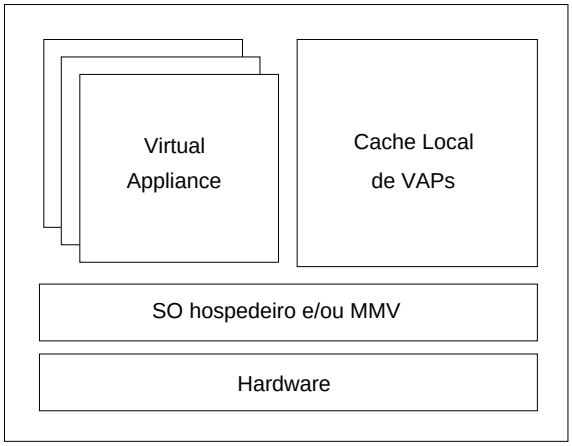

Figura 3.5: Um host com VAPs

mandar diferentes monitores de máquinas virtuais.

3.2.3 Visão Geral da Arquitetura

A arquitetura básica da solução proposta pode ser visualizada na figura 3.6. Como pode ser observado, há dois grupos macros de componentes: o repositório global e o *host* hospedeiro.

O repositório global contém as informações e os dados essenciais para o funcionamento do sistema. A idéia é que ele concentre todas as partes vitais, dinâmicas e flexíveis do sistema, incluindo os *VAPs*, arquivos e informações de configuração, autenticação e controle de usuários, entre outros elementos.

A comunicação entre os componentes do *host* hospedeiro e o repositório global é realizada apenas através do protocolo *HTTP*. Essa característica facilita a transposição de *firewalls* e padroniza os mecanismos de comunicação do sistema.

3.2.4 Componentes do *Host* Hospedeiro

O grupo de componentes do computador é composto por arquivos de configuração (*XML*), *cache component*, *VAP cache*, *deployment agent* e *VMM component*. Cada componente é especializado em uma atividade e/ou necessário para garantir a flexibilidade e o desempenho do sistema.

Os arquivos de configuração são necessários aos demais componentes do sistema local (computador). Eles contêm informações básicas de configuração para os demais componentes locais e também informações sobre o repositório e configurações globais.

O *cache component* é o componente responsável pela manutenção e atualização da

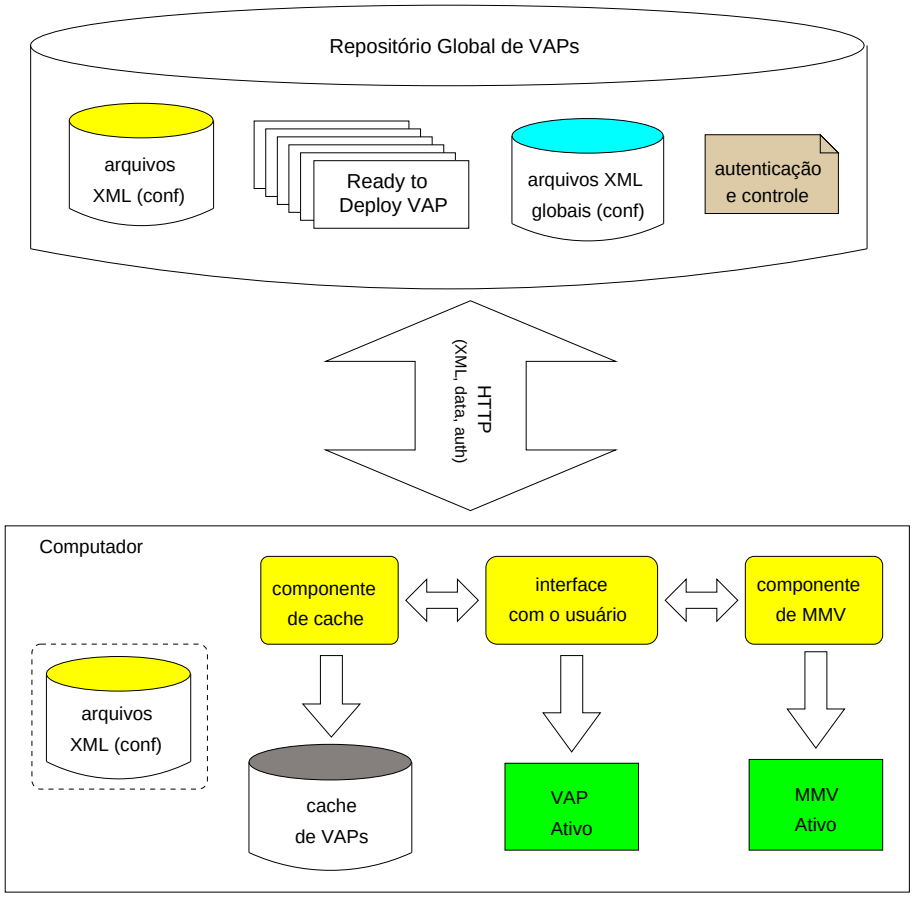

Figura 3.6: Arquitetura básica do *FlexVAPs*

cache local. Basicamente, este componente recebe requisições dos usuários do computador, consulta o repositório global e baixa ou atualiza as *VAPs* na *cache* local sempre que necessário.

A *VAP cache* é uma *cache* local para armazenamento das *VAPs* baixadas do repositório global. A principal função dessa *cache* é diminuir o tráfego da rede e aumentar o desempenho local do sistema. O *VAP* é copiado do repositório global para o sistema local apenas uma única vez. Os acessos subseqüentes a esse *VAP*, enquanto ele não sofrer modificações no repositório global, são realizados localmente.

O *VMM component* é um componente especializado no *download* e instalação de novos *MMVs*. Esse componente é especialmente útil quando o usuário quer executar um *VAP* sem suporte de *MMV* no sistema local. Nesses casos, se desejado pelo usuário, desde que este também tenha as permissões adequadas no sistema, o *VMM component* irá analisar e procurar pelos *MMVs* mais adequados à execução do novo *VAP*. Em caso positivo, o *VMM component* irá instalar um novo *MMV* capaz de comportar o novo *VAP*.

O *deployment agent* é a parte crucial disponível no *host* hospedeiro. É através dele que

os demais componentes, como o *cache component* e o *VMM component*, são utilizados. O *deployment agent* é a interface entre o usuário final e o sistema. Através dele o usuário pode selecionar a *VAP* desejada e solicitar que ele seja ativado no *host* hospedeiro. Quando o *VAP* pode ser mantido na *cache* local, o *deployment agent* faz uma solicitação ao *cache component* para que a *cache* local seja atualizada se necessário. Quando o sistema local não possui o *MMV* necessário à ativação do *VAP*, o componente realiza uma solicitação ao *VMM component*, que analisa a disponibilidade e autorização para instalação do *MMV* apropriado à ativação da *VAP*.

3.2.5 Componentes do Repositório Global

O grupo de componentes do repositório global contém informações, mecanismos e recursos necessários para a operação e manutenção do sistema como um todo. Os componentes básicos são: arquivos de configuração (*XML*), *VAPs* prontas para *download* e ativação, arquivos *XML* de configuração global e mecanismos de controle e autenticação de usuários.

Os arquivos de configuração são utilizados para manter e atualizar configurações locais dos *hosts* hospedeiros. Isso facilita, simplifica e dinamiza o trabalho dos administradores de sistemas, pois um laboratório de informática, por exemplo, poderia ter suas configurações atualizadas de forma dinâmica e automática.

Os *VAPs* são os arquivos necessários para instanciar as máquinas virtuais nos *hosts* hospedeiros. Um *VAP* inclui os dados e as configurações necessárias à ativação e manutenção da máquina virtual.

Os arquivos de configuração global contêm conjuntos de informações pertinentes à manutenção dos repositórios globais, à distribuição de arquivos entre conjuntos de máquinas que compõe o repositório global, arquivos de configuração que descrevem diferentes tipos de informações, como tipos e comandos de compressão e descompressão de dados, configuração dos possíveis grupos de *VAPs*, entre outros elementos.

Os mecanismos de autenticação são responsáveis por permitir - ou não -, que um usuário acesse e utilize um determinado conjunto de *VAPs*. Eles oferecem os recursos básicos para controlar, no caso dos *VMM components*, a autorização de instalação de novos *MMVs* nos *hosts* hospedeiros. Um usuário não autorizado será impedido de baixar e ativar *VAPs* de grupos privados, bem como instalar *MMVs* complementares.

3.2.6 *VAP* Pronta para Ativação

O *VAP* é o componente essencial do sistema. Toda a infraestrutura, tanto do repositório global, quanto do *host* hospedeiro, gira em torno da distribuição, uso, configuração e ativação de *VAPs*.

Um *VAP* é um pacote de dados necessário à instanciação de uma *MV*. Este pacote pode ser específico, destinado a um determinado *MMV*, ou genérico, podendo ser utilizado em um conjunto de diferentes *MMVs*, como VMware, Xen, VirtualBox e Qemu. Em todos os casos, ele é constituído por um pacote que contém elementos básicos, como sistema operacional e aplicativos.

3.2.7 Exemplo de Aplicação

A fim de melhor ilustrar o funcionamento da arquitetura da proposta de sistema para gerenciar *VAPs* em parques de máquinas, segue um exemplo prático e simples. A figura 3.7 exibe o caso de um usuário, em seu computador, preparando experimentos diversos. Supõe-se que o usuário, neste momento, esteja iniciando a preparação de uma aula prática de banco de dados. Supõe-se também que o usuário não tenha o banco de dados instalado em sua máquina e não deseja fazê-lo, visto que irá utilizar apenas uma única vez, para uma única aula. Além disso, ele não deseja sobrecarregar desnecessariamente a sua máquina.

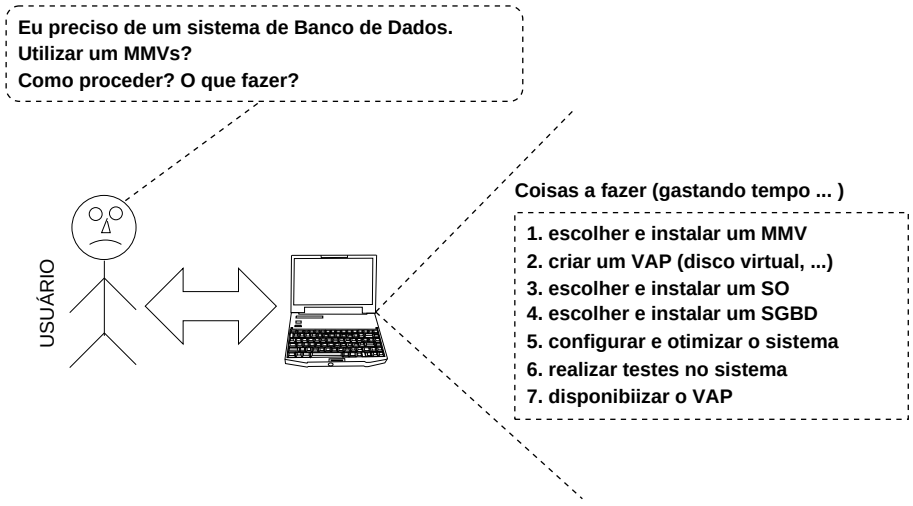

Figura 3.7: Usuário prepara aula de banco de dados

Uma das soluções passa pela utilização de *MVs*. Porém, o usuário não quer perder o seu tempo instalando um *SO*, preparando e configurando o sistema de banco de dados, o que pode ser relativamente simples ou mais complicado. Mesmo assim, em ambos os

casos o usuário terá que gastar algum tempo nessa atividade.

A saída para o usuário é utilizar um *VAP*, com *SO* e sistema de banco de dados já instalados, configurados e prontos para utilizar. Com esta opção, o usuário terá apenas o trabalho de obter e ativar o *VAP*. Isso, mais uma vez, pode, igualmente, ser trabalhoso se não houver um repositório de *VAPs* com alguns mecanismos de controle e ativação de *VAPs*, pois o usuário, em alguns casos, ainda precisará procurar pelo *VAP* e pelo *MMV* apropriado à execução do *VAP*.

Cenários como esse estão entre os objetos de aplicação da solução proposta. O sistema arquitetado e prototipado neste trabalho leva em conta a existência de repositórios de *VAPs*. Estes repositórios são gerenciados pelos administradores de redes e/ou sistemas. Sendo assim, as demandas dos usuários são encaminhadas até eles. A partir da demanda, os administradores preparam e disponibilizam, de forma pública ou privada (grupo ou indivíduo), um *VAP* capaz de suprir as demandas do usuário.

O usuário, como ilustrado na figura 3.8, precisa apenas utilizar o *deployment agent*, conforme apresentado na seção 3.2.4, para solicitar o *download* e a ativação do *VAP* desejado. Isso facilita e simplifica o trabalho do usuário, pois ele não precisa mais sair à procura de *SOs*, cópias de arquivos de instalação de bancos de dados, *VAPs*, *MMVs* e/ou realizar instalações, configurações e testes de ambientes. Com a solução proposta neste trabalho, o usuário pode gastar o seu tempo em atividades mais pertinentes a sua função, enquanto que a infraestrutura, a sistemática e o administrador do sistema de gerenciamento de *VAPs* provêem os recursos, as ferramentas e o trabalho necessário ao atendimento da demanda do usuário.

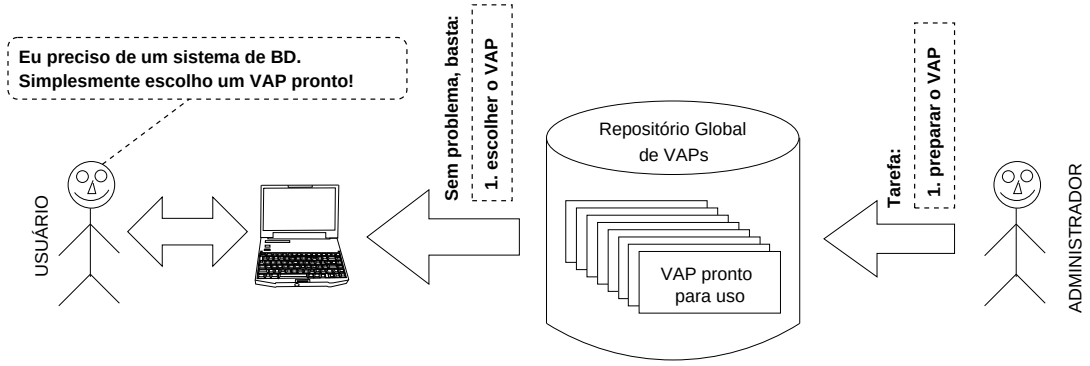

Figura 3.8: Usuário escolhe o *virtual appliance* desejado

O administrador do sistema possui todas as ferramentas e recursos necessários à rápida e correta instalação, configuração e preparação de novos *VAPs*. A instalação de um

sistema de banco de dados pode rapidamente ser realizada através de *templates* e *VAPs* pré-configuradas, ou seja, para um administrador de sistema é mais fácil, rápido e prático que para um usuário final.

3.3 Detalhamento da Arquitetura

3.3.1 *Virtual Appliance*

Um *VAP* é um pacote de dados. Ele contém um *SO* e aplicações específicas para diferentes finalidades. Um *VAP* consiste nos dados necessários à instanciação de uma *MV* sobre um determinado *MMV*.

3.3.1.1 Descrição

Um *VAP* é descrito por um conjunto de dados e informações básicas de uso, manutenção e aplicação. Essas informações incluem dados sobre o *SO* e as aplicações da *VAP*, bem como os *MMVs* nos quais ele pode ser instanciado.

O administrador do sistema, quando cria e disponibiliza o *VAP*, é responsável por incluir as informações básicas e essenciais sobre ele. Isso é feito, no modelo adotado, através de arquivos *XML*, como pode ser observado nas próximas seções.

3.3.1.2 Dependências

As dependências têm por objetivo especificar relações entre *VAPs*, no sentido de um *groupware*, ou seja, um conjunto de *VAPs* que, no conjunto, são capazes de criar um ambiente muitas vezes complexo. Um exemplo prático é um ambiente de desenvolvimento de sistemas. Supondo que estejamos tratando de ambientes Java, onde há as ferramentas de implementação, os servidores de aplicação, os sistemas de banco de dados e um sistema de versões concorrentes (*CVS*). A figura 3.9 ilustra esse cenário.

As ferramentas de desenvolvimento normalmente são instaladas e utilizadas nos *PCs* dos programadores. Já os servidores de aplicação, banco de dados e sistemas de controle de versionamento de código são normalmente compartilhados entre os desenvolvedores. Logo, caso o usuário (empresa, desenvolvedor, grupo de pesquisa, entre outros) desejasse criar o ambiente completo de desenvolvimento de sistemas, ele teria que instalar e configurar diferentes sistemas, o que pode ser trabalhoso e, inclusive, resultar em perdas financeiras indesejadas à corporação.

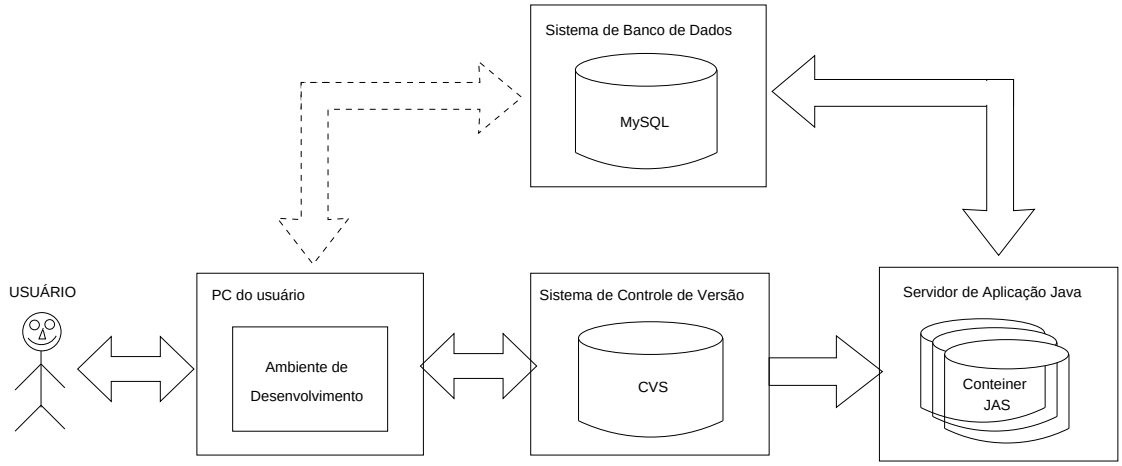

Figura 3.9: Ambiente de desenvolvimento

O uso de *VAPs* com especificação de dependências pode resolver o problema de uma maneira mais prática. O usuário, quando deseja criar um ambiente de desenvolvimento, simplesmente seleciona o *VAP* com as ferramentas desejadas, no caso do exemplo anterior, o *VAP* com Java e as demais ferramentas de implementação. O sistema irá detectar as dependências e proceder à ativação de todos os *VAPs* caso necessário/desejado pelo usuário.

3.3.2 Grupos de *VAPs*

A aplicação de grupos de *VAPs* é diferente do uso de relações de dependências. No caso dos grupos não existe necessariamente um vínculo de dependência.

Os grupos de *VAPs* são interessantes em contextos onde se deseja criar um ambiente qualquer, diversificado ou não, a critério do usuário. Nesses casos, os *VAPs* poderão ser ativados sem nenhuma seqüência e poderão ou não possuir relações de dependência. Um exemplo poderia ser a criação de um aglomerado de computador (*cluster*) virtual. O aglomerado pode ter um conjunto de nós variável, sendo que cada nó pode, inclusive, ser constituído por sistemas diversos. Não há necessariamente um vínculo direto entre os nós, pois eles simplesmente estarão ou não disponíveis às aplicações paralelas desenvolvidas para o ambiente em questão.

Na definição dos grupos de *VAPs* há uma associação com grupos de *hosts*. Estes correspondem aos locais aonde os *VAPs* serão instanciados.

3.3.3 Componente de *Cache*

O objetivo primário do componente de *cache*, localizado no *host* hospedeiro, é manter o máximo de dados dos *VAPs* localmente. Desta forma, em situações recorrentes, as solicitações dos usuários poderão ser atendidas com maior rapidez e menor consumo de rede.

A figura 3.10 ilustra o funcionamento e uso do componente de *cache*. Ele é especialmente útil em casos/cenários onde seria necessário o *download* de um mesmo *VAP* várias vezes em um mesmo *host* hospedeiro. Supondo um *VAP* de 500 *MB*, fazer o *download* por quatro vezes representaria um consumo de 2 *GB*, desnecessário, de banda da rede de dados.

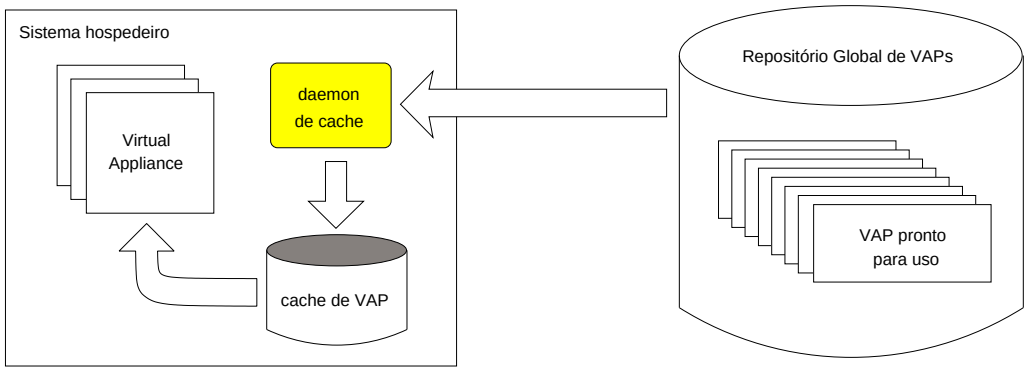

Figura 3.10: Componente de *Cache*

Em suma, a *cache* irá reduzir o consumo médio de banda da rede, pois cópias dos *VAPs* serão mantidas localmente. Apenas em casos de atualização dos *VAPs* será necessário baixá-los novamente para a *cache* local.

3.3.4 Componente de Instalação de *MMVs*

O componente de instalação de *MMVs* tem como função suprir casos de ausência de *MMV* apropriado para instanciação do *VAP*. Ele permite que o usuário instale um novo *MMV* que permite a ativação do *VAP*. A figura 3.11 ilustra o uso do componente de instalação de novos *MMVs*. Quando o usuário solicita a instalação de um novo *MMV* para suprir as exigências de um *VAP*, o componente de instalação busca o pacote de dados do novo *MMV* no repositório global. O pacote é então baixado e instalado no sistema local.

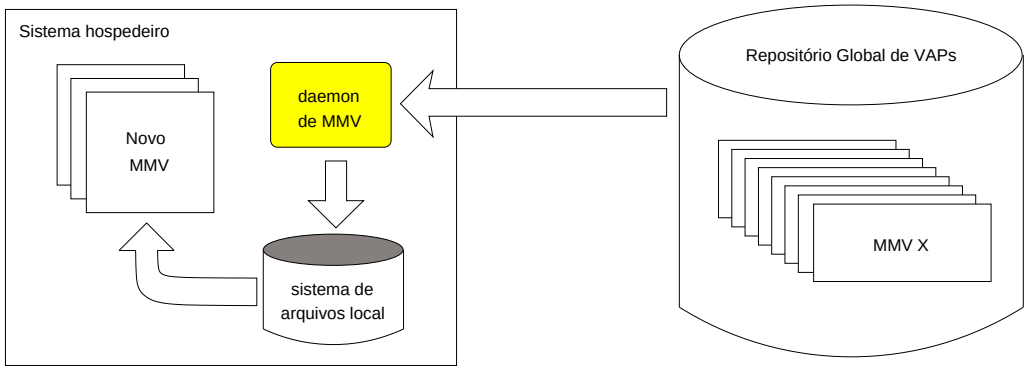

Figura 3.11: Componente de Instalação de *MMVs*

3.3.5 Interfaces em Nível de Usuário

A interface com o usuário é realizada através do componente de gerenciamento da utilização dos *VAPs*, também denominado de *deployment agent*, como ilustrado na figura 3.12. O usuário, através desse componente, faz a requisição do *VAP*. A interface do usuário, sempre que necessário, comunica-se com os componentes de *cache* e instalação de *MMVs*.

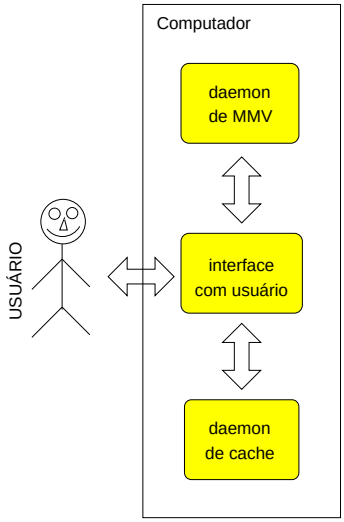

Figura 3.12: Interfaces em Nível de Usuário

O objetivo principal desse componente é fornecer uma interface simples para o usuário, proporcionando um uso prático de *VAPs*. O componente, antes de possibilitar e atender as requisições do usuário, irá também checar a autenticação do mesmo, utilizando-se dos mecanismos descritos na seção 3.3.6.

3.3.6 Autenticação, Controle e Acesso aos Dados

A autenticação e o controle de acesso são importantes em ambiente onde deseja-se ter diferentes tipos/grupos de usuários. O sistema de autenticação é localizado no repositório global e pode fazer uso de diferentes mecanismos de autenticação. Tradicionais mecanismos de autenticação, como *PAM*, *LDAP*, baseada em aplicação e autenticação local, podem ser utilizados como formas de autenticar os usuários.

A idéia é que os usuários possam também ser agrupados de acordo com os interesses. Com isso, poderão existir grupos de *VAPs* acessíveis apenas por determinados grupos de usuários.

3.4 Repositório Global - Exemplos de Configuração

3.4.1 Um Único Repositório Global

Um dos principais objetivos da arquitetura proposta é a concentração dos dados e mecanismos funcionais em um único repositório global. Isso facilita o controle, o gerenciamento e simplifica o sistema.

Como pode ser observado na figura 3.13 e na seção 3.1, o repositório global possui um conjunto de componentes, dados e informações que orquestram o funcionamento do sistema como um todo. O administrador do sistema precisa basicamente preocupar-se com o repositório global.

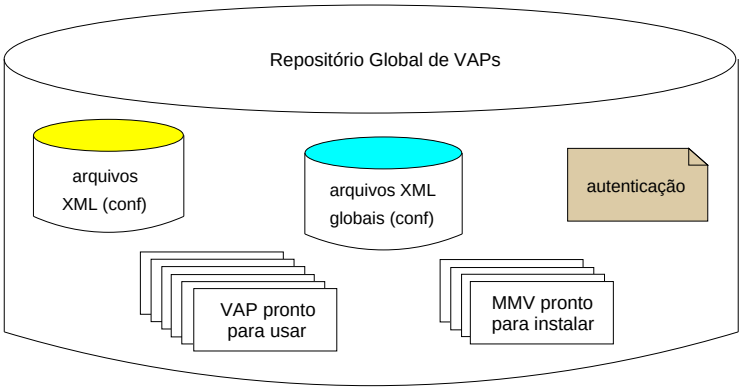

Figura 3.13: Repositório global

3.4.2 Repositório com Múltiplos Servidores

A arquitetura precisa também ser flexível o suficiente para tornar possível a distribuição das funcionalidades do repositório e ou a simples distribuição dos dados/pacotes,

como conjuntos de *VAPs* e *MMVs* prontos para serem instalados nos sistemas locais. A figura 3.14 ilustra graficamente um repositório com múltiplos servidores de dados. Como pode ser observado, o único componente que está presente em todos os servidores de dados é o de autenticação e controle de acesso dos usuários. Os demais componentes precisam estar presentes apenas em um dos servidores de dados.

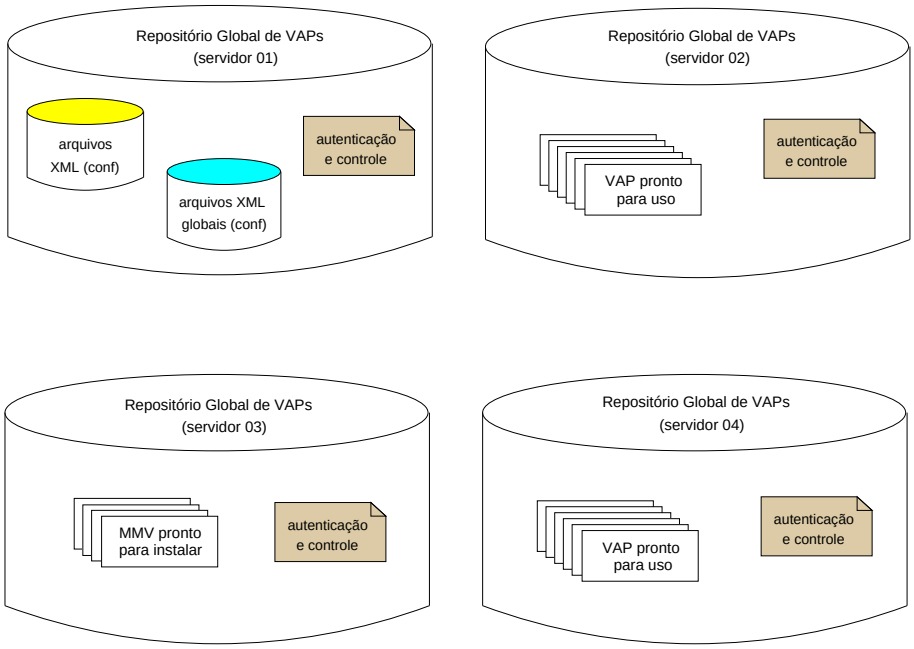

Figura 3.14: Múltiplos Servidores para o Repositório

Essa arquitetura permite separar, por exemplo, arquivos de configuração, *scripts* e arquivos de dados. Em termos práticos, há aspectos de segurança que podem ser reforçados em alguns servidores, como é o caso do servidor de arquivos de configuração.

O uso de múltiplos servidores de dados permite também a implementação de tolerância a falhas e distribuição de carga. Tudo isso é possível de forma simples e transparente para o usuário final.

Outro aspecto importante do uso de múltiplos servidores de dados é a acoplação fraca, ou seja, não há uma relação direta de dependência entre os diferentes servidores de dados. Essa característica permite a criação, distribuição e uso de múltiplos servidores de dados sem aumentar a complexidade do sistema em si.

3.4.3 Separação entre Dados, Configurações e Outros

A flexibilidade apresentada na seção 3.4.2 permite a separação entre dados, configurações, *scripts*, etc. Essa separação pode ser interessante por questões de segurança,

praticidade e gerenciamento. Cada ao administrador do sistema definir e aplicar a separação.

Um exemplo de separação de dados pode ser um ambiente onde diferentes administradores, ou mesmo usuários, auxiliam na manutenção do repositório global. Neste caso, poderia ser interessante separar os *VAPs* e as configurações por grupos, sendo que cada grupo ficaria em um determinado servidores de dados. Os administradores do sistema teriam acesso apenas aos respectivos servidores de dados. Um administrador, ou usuário, que contribui para manter o conjunto de *VAPs* públicos teria acesso apenas ao servidor de dados que mantém a parte pública do repositório global.

3.4.4 Réplicas Globais

Outra flexibilidade importante é a possibilidade de replicação do repositório global como um todo. Assim como pode-se utilizar diferentes servidores de dados para manter os diferentes conjuntos de dados e informações do repositório global, como apresentado na seção 3.4.2, também é possível replicar-se por completo o repositório global. Essa característica permite um maior grau de redundância, distribuição de carga e tolerância a falhas como um todo.

A figura 3.15 ilustra a replicação do repositório global. No caso, foram criadas duas réplicas que precisam ser mantidas atualizadas. Já os clientes poderão utilizar qualquer uma das réplicas do repositório. O uso desses repositórios globais pode ser configurado localmente ou nas configurações do repositório global primário.

3.5 Considerações

A proposta visa suprir demandas básicas para o gerenciamento de um repositório global de *VAPs*. A meta é oferecer de forma simples e transparente, aos usuários finais, *appliances* capazes de resolver os mais diversos tipos de problemas. Para isso, a proposta de sistema é aderente ao gerenciamento de máquinas virtuais heterogêneas, permitindo, por exemplo, a instalação automática e transparente de novos *MMVs* sempre que necessário. Como exemplo, um usuário traz consigo um *VAP* de fora, sua ou de outrem, e quer criar uma instância do mesmo em um sistema local. Neste caso, pode ser necessária a instalação de um novo *MMV*, mas o usuário pode não ter permissão para realizar essa operação, além de ser um trabalho a mais para ele. A solução é resolvida pelo sistema

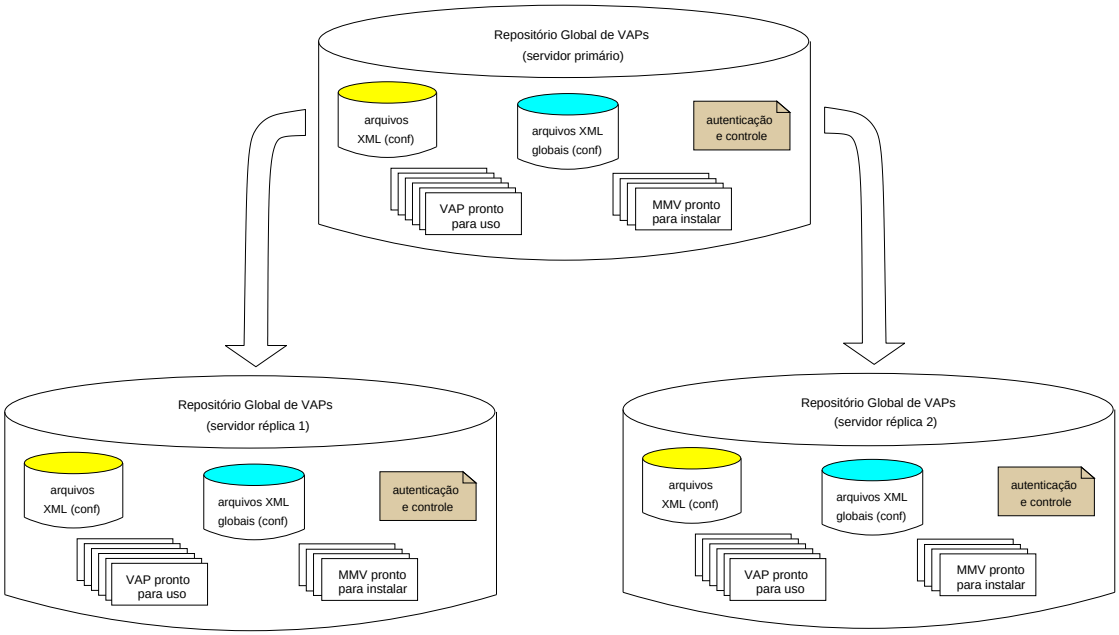

Figura 3.15: Réplicas globais

proposto, onde o *VMM component* permite que um usuário normal, com as devidas permissões no repositório global, instale de maneira automática e transparente um novo *MMV* no sistema local, resolvendo, desta forma, de maneira prática, o problema de gerenciar a heterogeneidade de máquinas virtuais.

4 DESENVOLVIMENTO E IMPLEMENTAÇÃO

4.1 Análise de Tecnologias de Desenvolvimento

4.1.1 Arquitetura

A solução proposta, seções 3.2.1, 3.1 e 3.2, baseia-se primariamente na existência de um repositório global de configurações, dados e informações sobre *VAPs*, *MMVs* e sistemas. Ela pressupõe a existência de componentes nos *hosts* hospedeiros que irão funcionar como interfaces entre as demandas dos usuários e os pacotes de dados disponíveis no repositório global.

A arquitetura proposta, seção 3.2, tem como principal objetivo proporcionar flexibilidade, simplicidade e eficiência no gerenciamento de infraestruturas computacionais virtuais. A idéia do uso de *VAPs*, uma tendência crescente na área da computação, busca concentrar os trabalhos de preparação e gerência de sistemas nos administradores de sistemas. Enquanto isso, os usuários podem dedicar integralmente os seus esforços nas atividades finais dos seus trabalhos, sem preocupar-se com infraestrutura. Mesmo para os administradores de sistemas, o uso de *VAPs* facilita e simplifica o trabalho, pois não há dependências de hardware e os processos de disponibilização de novos sistemas podem passar pelo simples uso de *templates* (modelos de *VAPs* pré-configurados) ou cópia de *VAPs* já existentes, com as devidas adaptações, por vezes pequenas.

Dentro desse contexto, uma das metas da implementação de um protótipo foi focar na simplicidade e flexibilidade da arquitetura proposta (seção 3.2). Complementarmente, utilizar ferramentas e soluções existentes, comumentemente utilizadas por administradores de sistemas, na composição da arquitetura, reduzindo potenciais trabalhos redundantes e, consequentemente, diminuindo também a curva de aprendizado dos administradores de sistemas.

4.1.2 Ambiente/Plataforma

Para o desenvolvimento do protótipo foi escolhido o sistema operacional *GNU/Linux* devido a sua flexibilidade e liberdade, principalmente por ser um sistema gratuito, de código aberto e amplamente suportado pela comunidade. Devido a isso, ele é uma das melhores alternativas para esse tipo de projeto, diminuindo implicações legais de desenvolvimento da solução e dependências de terceiros.

A família de sistemas *GNU/Linux* é interessante também pelo fato de suportar uma boa gama de *MMVs*, que são um dos objetos da proposta deste trabalho. Aspectos como simplicidade e flexibilidade, em termos de manutenção e automação de sistemas, são igualmente características importantes inerentes por natureza aos sistemas dessa família.

Complementarmente, a meta é utilizar o máximo possível de ferramentas e soluções, especialmente as gratuitas, na prototipação da arquitetura de sistema proposta neste trabalho (seção 3.2). Neste sentido, a plataforma *GNU/Linux* oferece uma grande variedade e quantidade de aplicativos, ferramentas e soluções tipicamente utilizadas na construção e manutenção de sistemas.

4.1.3 Ferramentas, Tecnologias e Aplicativos

Uma das preocupações da prototipação da solução foi relacionada às ferramentas e tecnologias. Como o foco do arquitetura de sistema proposta é administradores de sistemas (mantenedores) e usuários finais, a meta principal foi focar em ferramentas e tecnologias tradicionalmente conhecidas e utilizadas por administradores de sistemas.

Nessa linha de pensamento podem ser citadas algumas das escolhas realizadas na fase de prototipação. Todas tem como base de escolha parâmetros como o grau de conhecimento e utilização entre administradores de sistemas, a robustez, a maturidade da solução/linguagem, o uso em larga escala, a disponibilidade de documentação, o número e a atividade das comunidades pública, a produtividade, o tipo de licença (gratuito e de código aberto, preferencialmente), a disponibilidade (distribuição livre, preferencialmente) e a flexibilidade.

Linguagens de *scripting*: o uso de linguagens de *scripting*, tradicionais e conhecidas pelos administradores de sistemas de plataformas *GNU/Linux*. Exemplos de linguagens interpretadas: `Perl`, `Python`, `Bash` e `Tcsh`. Neste trabalho foram utilizadas as linguagens `Perl` e os recursos de *scripting* do `Bash`.

Servidores Web: uso de servidores Web, base do repositório global da solução proposta, para gerenciar e controlar o acesso aos conteúdos e informações sobre os pacotes de dados e configuração da solução proposta. Exemplo: Apache versão 2, um dos servidores Web mais robustos e utilizados no mundo.

Mecanismos de autenticação e controle de grupos de usuários: eles são importantes para o gerenciamento do acesso aos dados do repositório global, controlando usuários e grupos. Com esses mecanismos é possível manter-se diferentes grupos de *VAPs*, por exemplo, focados para diferentes contextos de aplicação, restringindo o acesso em nível de usuários e grupos. Exemplos de sistemas e tecnologias para o controle de usuários e grupos: *LDAP*, *NIS* e *PAM*. O *LDAP*, utilizado para gerenciar usuários e grupos junto ao `Apache2` na solução proposta, é um diretório *online* que tem sido utilizado amplamente na administração de credenciais em redes e sistemas.

Aplicativos de *download* de arquivos na Web: os aplicativos de *download* tem como finalidade básica proporcionar ao usuário o uso simples de protocolos como o *HTTP*. Eles disponibilizam interfaces práticas para a escolha e *download* de arquivos. Além disso, ele provêem também interface para os mecanismos de autenticação de servidores Web, promovendo níveis maiores de segurança sempre que necessário. Exemplos de aplicativos de *download* comuns em sistemas *GNU/Linux*: `links`, `lynx` e `wget`. Na implementação do protótipo foi utilizado o utilitário `wget`.

Aplicativos de sincronização de dados: o uso de aplicativos de sincronização de dados, geralmente arquivos, é especialmente interessante em ambientes onde deseja-se manter cópias dos cados, seja como réplica para disponibilidade, seja como *cache*. Na arquitetura proposta, seção 3.2, os aplicativos de sincronização de dados são, em particular, úteis para a manutenção das *caches* nos sistemas hospedeiros e *backup* e réplicas do repositório global. Exemplos de aplicativos de sincronização de arquivos e diretórios: `rsync`, `zsync`, `simba`, `csync2`, `unison` e `FreeFileSync`. Para sincronizar os dados entre o repositório global e os sistemas locais, a melhor opção, utilizada na implementação, é o `zsync`, visto que esta ferramenta realiza a sincronização de arquivos através do protocolo *HTTP* (Web). Como um dos objetivos do trabalho é focar em um protocolo de comunicação padrão, que normalmente

permite a transposição de sistemas de *firewall*, essa ferramenta foi a opção mais adequada, pois ela dispõe também de recursos similares ao `rsync`, permitindo, por exemplo, cópias diferenciais, ou seja, copiar apenas os blocos de dados do arquivo que sofreram alguma alteração, reduzindo a demanda de tráfego na rede.

Sistemas de arquivos com replicação nativa: os sistemas de arquivos com suporte a replicação nativa são úteis para a criação de cópias de alta disponibilidade do repositório global. Uma maior disponibilidade pode auxilar na distribuição de carga e conseqüente aumento da eficiência do sistema. Exemplos de sistemas de arquivos com suporte a replicação de dados: `InterMezzo`, `Lustre`, `MFS`, `Tahoe File System`, `rsyncfs` e `Starfish`. Um dos objetivos do trabalho, como meta futura, é avaliar alguns desses sistemas de arquivos para a manutenção de cópias do repositório global, aumentando a disponibilidade e a tolerância a falhas no sistema.

Linguagens de descrição e representação de dados: linguagens de descrição de dados são úteis em diferentes contextos. Elas tem sido aplicadas nas mais diversas áreas, seja para organização dos dados, seja para interoperabilidade entre sistemas, seja para padronização, uniformidade e flexibilidade. No âmbito deste trabalho, o objetivo é utilizar linguagens e ou formatos de representação de dados padrão de mercado, simples, inteligíveis, uniformes e flexíveis, padronizando formas de descrição e representação de dados. Ao mesmo tempo, a idéia é utilizar um padrão amplamente difundido e utilizado, tornando possível o uso de linguagens de programação e ferramentas diversas. Exemplos de linguagens e formatos de descrição e representação de dados: *XML*, *SGML* e *CSV*. A opção natural para os propósitos e as necessidades da solução proposta é o *XML* devido a sua simplicidade, uso em larga escala e grande variedade de ferramentas disponíveis para o desenvolvimento de *software* nas mais variadas linguagens de programação.

4.2 Implementação

4.2.1 Prototipação

Arquitetura. Alguns dos detalhes da implementação da arquitetura proposta, seção 3.2, podem ser vistos na figura 4.1. Como pode ser observado, a infraestrutura de dados é composta por diretórios gerenciados pelo `Apache2`.

Cada diretório contém conjuntos de *VAPs*, arquivos de configuração ou *scripts* de gerenciamento. O acesso aos diretórios é controlado pela `Apache2`, podendo ou não conter requisitos de autenticação.

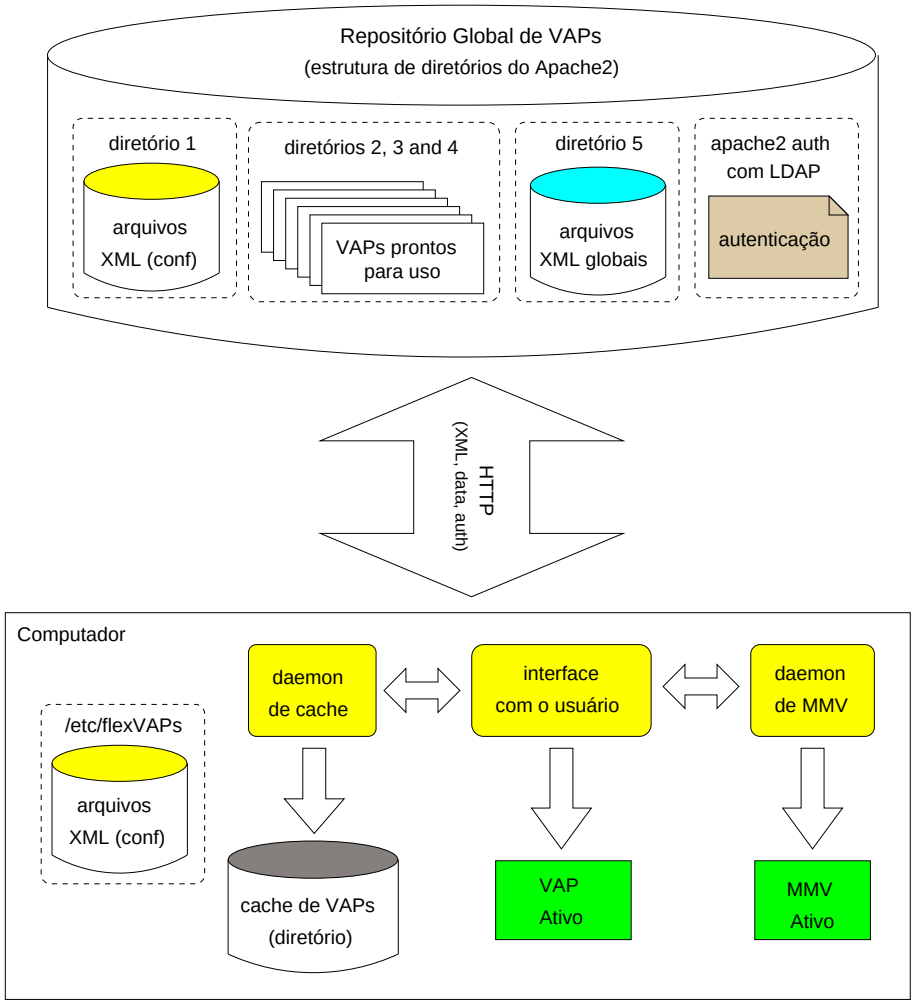

Figura 4.1: Arquitetura desenvolvida

Quando há requisitos de autenticação para o acesso a um diretório de *VAPs*, o controle é baseado em diretivas de configuração do `Apache2`. As diretivas utilizadas fazem uso da estrutura e organização de grupos de usuários do *LDAP*. Sendo assim, para cada diretório pode ser definido um determinado grupo de usuários autorizados a baixar os pacotes de dados e configurações nele contidos.

As configurações essenciais dos componentes de *cache*, *MMV* e interfaces disponíveis aos usuários possuem suas configurações locais no diretório `/etc/flexVAPs`. O endereço da *cache* local, diretório, partição ou unidade virtual de armazenamento, é definido nas configurações locais.

Aplicativos e utilitários. Na prototipação da solução foram utilizados o servidor Web

`Apache2`, aplicativos como `links`, `wget` e `zsync` e utilitários de gerenciamento de diretórios *online LDAP*, como o pacote `ldap-utils` do *GNU/Linux*. Com exceção do `ldap-utils`, necessário apenas entre o diretório *LDAP* e o servidor Web `Apache2`, todas as demais ferramentas são utilizadas exclusivamente sobre o protocolo *HTTP*, o que facilita a transposição de sistemas de segurança, como *firewalls*, além de padronizar a comunicação com os servidores do repositório global.

4.2.2 Repositório Global

O repositório global contém os pacotes de dados, as configurações e os *scripts* do sistema como um todo. Os *VAPs* são baixados do repositório global para as *caches* dos sistemas hospedeiros.

4.2.2.1 Armazenamento e Acesso aos Dados

O armazenamentos dos *VAPs*, *scripts*, configurações globais *XML*, pacotes de dados dos *MMVs*, entre outros dados, é realizado em uma estrutura de diretórios sob o domínio do `Apache2`.

O acesso aos dados do repositório é realizado exclusivamente através do protocolo *HTTP*. Isso facilita a transposição de sistemas de segurança, como *firewalls*, e padroniza o acesso aos dados. Ademais, há uma boa gama de aplicativos e ferramentas que podem ser utilizados para acesso aos dados, visto que é comum os utilitários dos sistemas operacionais possuírem suporte Web.

A figura 4.2 ilustra uma estrutura de diretório do `Apache2` para gerenciar o repositório global. No exemplo existem quatro níveis de diretórios, sendo eles o de armazenamento de *VAPs*, *MMVs* e configurações e o diretório de autenticação. Este último é utilizado pelo *daemon* de instalação de *MMV* do sistema hospedeiro para determinar se o usuário é ou não autorizado a instalar novos *MMVs* no sistema local.

A seguir, na figura 4.3, é apresentada a estrutura de diretórios no `Apache2` propriamente dito. Como pode ser observado, há agrupamentos de *VAPs* de acordo com os tipos de usuários e aplicações. Há *VAPs* de domínio público (`public`), de cursos/treinamento (`training`), de administração de sistemas (`admins`), de serviços de rede (`netservices`), de desenvolvimento (`development`), entre outros.

A estrutura de diretórios de autenticação (`auth`) é utilizada para autenticar usuários que tentam instalar novos *MMVs* nos *hosts* hospedeiros. Apenas usuários autorizados,

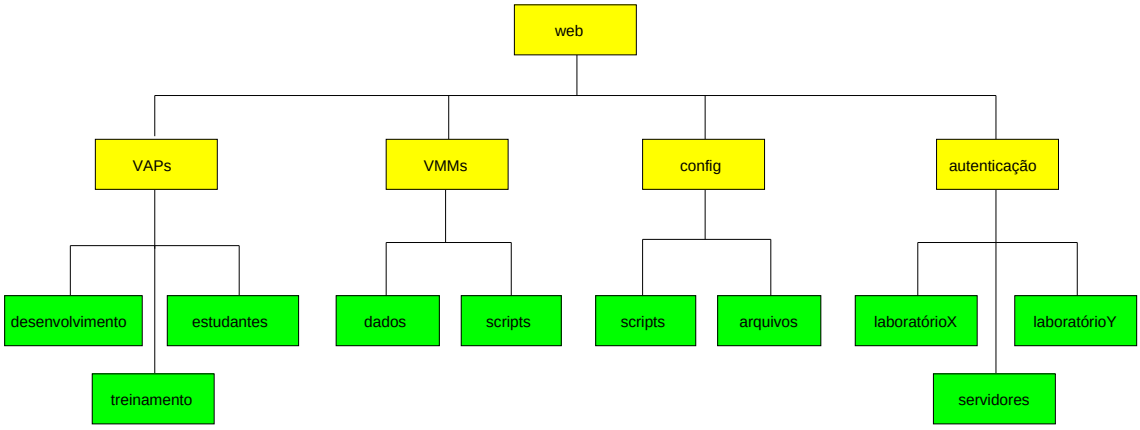

Figura 4.2: Diretórios `Apache2`

de acordo com um grupo de *hosts* em particular, podem fazê-lo. A exemplo, apenas um administrador de sistemas pode instalar novos *MMVs* em um servidor da rede. Já em um laboratório de ensino, alunos e professores poderiam também ser autorizados a instalar novos *MMVs* sempre que necessário. Esse controle é realizado pelos diretórios de autenticação. Estes estão vinculados, através do `Apache2`-*LDAP*, a grupos de usuários da rede.

A figura 4.4 apresenta um exemplo de configuração de um diretório no `Apache2`, sem a parte de controle/autenticação. Como pode ser observado, essencialmente, há apenas diretivas simples de configuração do `Apache2`.

4.2.2.2 Autenticação e Controle

Authenticação `Apache2`-*LDAP* . A figura 4.5 ilustra um exemplo de configuração de um diretório do `Apache2` com autenticação. No caso, o provedor da autenticação é o *LDAP*. Ele poderia ser qualquer outro mecanismos suportado pelo `Apache2`, como autenticação utilizando banco de dados (exemplo: `MySQL`) e autenticação em nível de aplicação, ou seja, do próprio servidor Web.

Um usuário somente conseguirá realizar o *download* de *VAPs* presentes no diretório `students` se ele pertencer ao grupo `students`, definido na linha 11 da configuração apresentada na figura 4.5. Sendo assim, o controle de acesso é realizado pelo `Apache2`, de acordo com os grupos e credenciais cadastrados no *LDAP*.

Grupos de usuários. A figura 4.6 ilustra a representação de um grupo no *LDAP*. A definição da entrada (`entry`) do grupo utiliza apenas esquemas padrões, disponíveis publicamente, ou seja, é um exemplo típico de definição de grupos em diretórios *online*

```
/var/www/flex
/var/www/flex/VAPs
/var/www/flex/VAPs/public
/var/www/flex/VAPs/students
/var/www/flex/VAPs/teachers
/var/www/flex/VAPs/admins
/var/www/flex/VAPs/netservices
/var/www/flex/VAPs/training
/var/www/flex/VAPs/development
/var/www/flex/VAPs/scripts
/var/www/flex/VMMs
/var/www/flex/VMMs/scripts
/var/www/flex/VMMs/data
/var/www/flex/config
/var/www/flex/config/scripts
/var/www/flex/config/files
/var/www/flex/auth
/var/www/flex/auth/laboratoryX
/var/www/flex/auth/laboratoryY
/var/www/flex/auth/servers
/var/www/flex/auth/development
```

Figura 4.3: Estrutura de diretórios no Apache2

```
1.  Alias /flex/VAPs/students "/var/www/flex/VAPs/public"
2.  <Directory /var/www/flex/VAPs/students/>
3.      Options -Indexes FollowSymLinks MultiViews
4.      AllowOverride all
5.      Order allow,deny
6.      allow from all
        ...
X.  </Directory>
```

Figura 4.4: Exemplo de configuração do Apache2

```
1.  Alias /flex/VAPs/students "/var/www/flex/VAPs/students"
2.  <Directory /var/www/flex/VAPs/students/>
3.    Options -Indexes FollowSymLinks MultiViews
4.    AllowOverride all
5.    Order allow,deny
6.    allow from all
7.    AuthType Basic
8.    AuthName  "FlexVAPs - auth required"
9.    AuthBasicProvider ldap
10.   AuthLDAPURL
        ldap://localhost/ou=users,dc=intranet??sub?(objectclass=*)
11.   require ldap-group cn=students,ou=groups,dc=intranet
12.   AuthLDAPBindDN cn=manager,dc=intranet
13.   AuthLDAPBindPassword flexvaps
14. </Directory>
```

Figura 4.5: Exemplo de autenticação de diretório no Apache2

```
1. # students, groups, intranet
2. dn: cn=students,ou=groups,dc=intranet
3. objectClass: posixGroup
4. objectClass: intranetGroup
5. objectClass: top
6. cn: students
7. gidNumber: 1000
8. uniqueMember: uid=foo1,ou=users,dc=intranet
9. uniqueMember: uid=foo2,ou=users,dc=intranet
```

Figura 4.6: Exemplo de grupo no *LDAP*

LDAP, comumentemente utilizados na administração de redes e sistemas diversos. As linhas 8 e 9 especificam os usuários que pertencem ao grupo. Sendo assim, essas linhas contém os identificadores únicos, também conhecidos como *DNs*[1], dos usuários.

4.2.3 *Daemon* de *Cache*

O *daemon* de *cache*, denominado `vapcached`, é o componente utilizado entre as interfaces em nível de usuário e o repositório global para o armazenamento temporário local dos *VAPs*. A solicitação do usuário é repassada ao *daemon* de *cache*, que avalia se o *VAP* pode ou não ser armazenado na *cache*, dependendo de sua configuração. Em caso positivo, o *VAP* é sincronizado do repositório global para a *cache* do sistema hospedeiro. Isso é realizado através da Web, utilizando o comando `zsync`.

A figura 4.7 apresenta um exemplo de configuração do *daemon* de *cache*. O tamanho da *cache* é especificado em *MB*. No exemplo a configuração compreende 10.000 *MB*, ou 10 *GB*.

O *daemon* de *cache* vai controlando também o nível de utilização da área de *cache* local. No momento em que um novo *VAP* precisa ser baixado do repositório global para a *cache* local, o *daemon* verifica o espaço disponível localmente. Caso o espaço seja insuficiente, ele verifica quais *VAPs* podem ser removidos da *cache* para dar espaço ao novo pacote de dados. A política padrão consiste em remover os *VAPs* mais antigos, ou menos recentemente atualizados, da *cache*. Nas configurações do *daemon* há um parâmetro, `max-vaps-rm-per-clean`, que define o número máximo de *VAPs* que poderão removidos em cada processo de limpeza. O ojetivo é simplesmente evitar que a *cache* tenha que ser totalmente limpa devido a um pacote de dados excessivamente grande. Neste

[1]Distinguished Names

```
<cache>
  <cachedir>/var/cache/flexVAPs</cachedir>
  <cachesize>10000</cachesize>
  <max-vaps-rm-per-clean>5</max-vaps-rm-per-clean>
  <daemon>
    <host>localhost</host>
    <port>7070</port>
    <protocol>tcp</protocol>
    <listen>5</listen>
    <reuse>1</reuse>
    <timeout>10</timeout>
    <number-of-tries>10</number-of-tries>
  </daemon>
</cache>
```

Figura 4.7: *XML* do *daemon* de *cache*

caso, a menos que o *VAP* seja realmente utilizado com freqüência, o pacote de dados poderia ser utilizado pelo usuário sem armazenamento local, ou seja, sem consumir espaço na *cache*.

Na configuração apresentada, o *daemon* estará aguardando conexões *TCP* na porta 7070. As requisições dos usuários são automaticamente encaminhadas pelos aplicativos em nível de usuário para os *daemons* de *cache*. Estes verificam a autorização dos usuários antes de proceder o *download* dos pacotes de dados para a *cache* local. As credenciais do usuário são apresentadas pelo `zsync` ao `Apache2`, liberando a sincronização do pacote de dados.

4.2.4 *Daemon* de Instalação de *MMVs*

O *daemon* de instalação de *MMVs*, denominado `vapvmminstd`, funciona de forma análoga ao *daemon* de *cache* (seção 4.2.3). A principal diferença é que o *daemon* de instalação recebe e processo requisições de instalação de novos *MMVs*. A figura 4.8 apresenta um exemplo de configuração do *daemon* de instalação de *MMVs*.

O primeiro parâmetro de configuração (`allow-install`) define se o sistema local permite ou não instalar novos *MMVs*. O segundo parâmetro (`install-auth-url`) especificação a *URL* de autenticação que deverá ser utilizada para checar a autorização, ou não, do usuário para a instalação do novo *MMV*.

Assim como no caso do *daemon* de *cache*, o *daemon* de instalação aguarda conexões *TCP* em uma porta específica, no caso 7071, definida na configuração. As requisições, encaminhas pelas interfaces dos usuários, são analisadas e processadas, quando autori-

```
<vmm-installer>
  <allow-install>yes</allow-install>
  <install-auth-url>
    http://auth.vap.intranet/VAPs/auth/labs
  </install-auth-url>
  <daemon>
    <host>localhost</host>
    <port>7071</port>
    <protocol>tcp</protocol>
    <listen>5</listen>
    <reuse>1</reuse>
    <timeout>10</timeout>
    <number-of-tries>10</number-of-tries>
  </daemon>
</vmm-installer>
```

Figura 4.8: *XML* do *daemon* de instalação de *MMVs*

zado (autenticação simples do usuário e verificação de sua autorização na *URL* específica de autenticação).

Uma vez confirmada a autorização do usuário, o processo de *download* e conseqüente instalação do novo *MMV* é iniciado. Os pacotes são descarregados do repositório global para o sistema local utilizando o aplicativo `wget`. No momento do *download*, novamente as credenciais do usuário são encaminhadas junto à requisição de *download* do pacote de dados.

4.2.5 Interfaces do Usuário

As interfaces do usuário final tem como finalidade prover meios para interagir com os demais componentes do sistema. O objetivo é atender as diferentes demandas do usuário, podendo englobar desde a simples requisição de um *VAP* até a ativação de um grupo de *VAPs*.

Uma das principais interfaces é o `vapdeploy`. Através dela o usuário pode listar os *VAPs* disponíveis no repositório global, escolher e solicitar a ativação do *VAP*. A interface permite também, por exemplo, ao usuário a requisição de instalação de um novo *MMV*, capaz de atender o *VAP* escolhido.

Uma segunda interface é o `vapgroup`, que possibilita a ativação de grupos de *VAPs*. Os grupos são aplicáveis em diferentes contextos, como laboratórios de pesquisa específicos.

Já o segundo tipo de usuário, o administrador do sistema, atua diretamente na infraes-

```
<vap-image>
  <name>student basic VAP </name>
  <description>Simple Raw GNU/Linux</description>
  <version>1.0</version>
  <url>
   http://repository.vap.intranet/flex/VAPs/students/studentVAP.tgz
  </url>
  <cacheable>yes</cacheable>
  <size>400</size>
  <timestamp>20090122111300</timestamp>
  <md5sum>b6d81b360a5672d80c27430f39153e2c</md5sum>
  <sha1sum>3b71f43ff30f4b15b5cd85dd9e95ebc7e84eb5a3</sha1sum>
  <compressed>yes</compressed>
  <config-file>
   http://config.vap.intranet/flex/VAPs/students/studentVAP.cfg
  </config-file>
  <optional-scripts>
    <firsttime-boot-script>
     http://scripts.vap.intranet/flex/scripts/labs/labX.fbs
    </firsttime-boot-script>
    <anytime-boot-script>
     http://scripts.vap.intranet/flex/scripts/labs/labX.abs
    </anytime-boot-script>
    <vap-startup-script>
     http://scripts.vap.intranet/flex/VAPs/students/studentVAP.vss
    </vap-startup-script>
  </optional-scripts>
</vap-image>
```

Figura 4.9: Exemplo de configuração de uma imagem de um *VAP*

trutura de diretórios do `Apache2`. Lá estão disponíveis conjuntos de *scripts* que auxiliam o processo de publicação dos *VAPs* e gerenciamento do controle de acesso. Os *templates*, para criação de novos *VAPs*, atendendo a demandas dos usuários finais, ficam a cargo do administrador do sistema. Bem como cabe ao administrador do sistema criar os *scripts* necessários para a ativação dos *VAPs* e ou a instalação de novos *MMVs*.

4.2.6 Ativação do *VAP*

A figura 4.9 ilustra um exemplo de configuração de um pacote de dados, um *VAP*. Esses dados são utilizados tanto pelos componentes de gerenciamento das *caches* locais, quanto pelos aplicativos de escolha e ativação dos *VAPs* em nível de usuário.

O pacote de dados é obtido do repositório global através da *URL* (`url`) definida no arquivo de configuração, como apresentado na figura 4.9. A ativação do *VAP* é feito através de um arquivo de configuração, como o `config-file`, que pode ser igualmente descarregado do repositório global para o sistema local, ou através de um *script*, denominado

de *VAP startup script* (`vap-startup-script`). Tanto o arquivo de configuração, dependente de *MMV*, devem ser preparados pelo administrador do sistema quando o *VAP* é criado.

Há também outros *scripts* opcionais, como os de primeiro (`firsttime-boot-script`) e qualquer (`anytime-boot-script`) *boot* do sistema. Ambos devem ser configurados pelo administrador do sistema nos sistemas operacionais dos *VAPs*. As funções desses *scripts* são diversas e altamente variáveis, dependendo das necessidades dos usuários para cada *VAP*. O *script* de primeiro *boot*, por exemplo, pode ser utilizado para configurar a rede do sistema na primeira instanciação da *MV*.

O uso de domínios distintos representa um dos pontos de flexibilidade da arquitetura. No exemplo da figura 4.9 há pelo menos três domínios, sendo eles `repository.vap.intranet`, `scripts.vap.intranet` e `config.vap.intranet`. Cada um pode representar um servidor distinto, sem comprometer absolutamente nada na infraestrutura do repositório global. Como apresentado na seção 3.4, um repositório global pode ser composto por múltiplos servidores de dados, aumentando em potencial características como flexibilidade, segurança e disponibilidade.

4.3 Considerações

O desenvolvimento do protótipo teve como um de seus objetivos utilizar recursos e ferramentas disponíveis e conhecidos por administradores de sistemas. Outro objetivo foi criar uma infraestrutura baseada em configurações *XML* inteligíveis e de fácil associação. Em nível de usuário final, as interfaces que permitem a interação com o sistema, possibilitando a utilização da infraestrutura provida pela solução.

A seção 6.2, do apêndice 6, contém exemplos de configurações de uso do sistema. Os exemplos ilustram configurações básicas de um *VAP* e um *MMV*, dois componentes essenciais e primordiais para o sistema.

5 AVALIAÇÃO

Este capítulo apresenta e discute os principais resultados do trabalho. A primeira seção (5.1) apresenta uma avaliação comparativa entre os sistemas de gerenciamento de *VAPs* Collective, `Scripts` IBM e *FlexVAPs*. As seções 5.2 e 5.3 apresentam alguns testes realizados com um repositório de *VAPs*, com o intuito de quantificar algumas características do sistema e verificar seu funcionamento. Na seção seguinte, 5.4, são apresentados alguns cenários práticos de aplicação da solução. Por fim, na última seção do capítulo são feitas algumas considerações sobre a avaliação do trabalho.

5.1 Análise Qualitativa

5.1.1 Comparativo de Características

As soluções que lidam com *virtual appliances* geralmente são focadas, ou até restritas, a um determinado tipo de aplicação. Esse é o caso de soluções como a da IBM, voltada para alta disponibilidade em *data centers*, e o Collective, voltado para usuários finais, com restrição de *MMV*.

A seguir serão analisadas algumas características consideradas importantes para o gerenciamento de *virtual appliances* para máquinas virtuais heterogêneas. Essas características se referem a qualidades dificilmente mensuráveis, mas que podem mesmo assim serem discutidas e comparadas quando presentes nos diferentes sistemas. Assim, propõe-se para cada característica uma escala classificatória, com vistas à avaliação de *FlexVAPs* frente a outros sistemas correlatos.

Adaptabilidade. É caracterizado como a capacidade de a solução ser facilmente adaptada a diferentes cenários e contextos de aplicação. Um sistema adaptável prevê em sua arquitetura componentes fracamente acoplados, ou seja, independentes. Com isso, novos cenários podem ser rapidamente gerenciados através da criação de componentes

específicos e independentes. Um sistema com adaptabilidade baixa é aquele que possui uma arquitetura mais rígida e única, ou seja, projetada sem a previsão de componentes simples e independentes. Por outro lado, um sistema com adaptabilidade média é aquele que possui uma arquitetura mista, parcialmente composta por componentes independentes. Já um sistema altamente adaptável é aquele projetado e composto por componentes simples e independentes, ou seja, fracamente acoplados.

`Scripts` IBM: adaptabilidade **média**. Há uma grande dependência dos *scripts* específicos de configuração e instanciação dos *virtual appliances* e existe uma falta de mecanismos automatizados para realizar a acoplagem automática dos disquetes virtuais que contém os dados de configuração e ativação. A versão disponível contempla *scripts* para Xen e Vmware. Contudo, os *scripts* podem ser considerados componentes fracamente acoplados e podem ser alterados por administradores de sistemas para satisfazer as suas necessidades.

Collective: adaptabilidade **baixa**. A sua principal finalidade consiste em prover soluções virtualizadas para os usuários finais sobre o monitor de máquinas virtuais VMware. Além disso, a arquitetura não prevê a existência de componentes fracamente acoplados, que poderiam ser criados pelos administradores de sistemas de acordo com suas demandas.

FlexVAPs: adaptabilidade **alta**. Uma de suas características é a flexibilidade advinda da estrutura de componentes fracamente acoplados, o que permite a fácil e rápida criação e adaptação de novos componentes, para cenários e contextos diversos. Além disso, não há restrição ou dependência alguma em relação a monitores de máquinas virtuais.

Heterogeneidade. Um sistema que preza pela heterogeneidade de máquinas virtuais deve ser capaz de lidar com os diferentes monitores de máquinas virtuais disponíveis no mercado. Os *VAPs*, que podem ser específicos aos *MMVs*, devem poder ser utilizados pelos usuários finais sem maiores necessidades de intervenção dos administradores de sistemas. Exemplo: um professor traz um *VAP* e disponibiliza ele em um repositório de *VAPs* da rede local. Os usuários, alunos, irão querer utilizar os sistemas disponibilizados pelo professor nesse *virtual appliance*. Porém, o laboratório de informática, além dos próprios *PCs* dos usuários, não contém o *MMVs* necessário para a instanciação do *VAP*. Neste caso, o sistema deve prover recursos que permitam atender de forma rápida e transparente esse demanda dos usuários. Heterogeneidade baixa significa que o sistema

foi projetado e concebido para determinados tipos de ambientes, como o *MMV*Vmware. Um sistema com heterogeneidade média é aquele concebido para alguns monitores de máquinas virtuais e que pode ser adaptado pelos usuários finais, como administradores de sistemas, para outros *MMVs*. Já um sistema com heterogeneidade alta é aquele concebido com heterogeneidade em mente, ou seja, oferece suporte nativo a diferentes *MMVs*.

`Scripts` IBM: heterogeneidade **média**. O sistema de *scripts* permite trabalhar com diferentes monitores de máquinas virtuais. Porém, para cada um terá que ser criado um conjunto de *scripts* específicos, ou seja, trabalhar com heterogeneidade aparentemente não é o foco da solução, pois ela considera basicamente o uso de VMware e Xen, ambas plataformas de virtualização voltadas para servidores de rede.

Collective: heterogeneidade **baixa**. O objetivo do Collective é disponibilizar *PCs* virtuais aos usuários finais sobre a plataforma do VMware. Há uma amarração forte entre a solução do Collective e o monitor de máquinas virtuais da VMware, sendo assim, *virtual appliances* diversas, que dependam de outros *MMVs*, não poderão ser utilizadas sobre sua infraestrutura.

FlexVAPs: heterogeneidade **alta**. A concepção e criação do *FlexVAPs* teve como objetivo primário a possibilidade de gerenciar *virtual appliances* para máquinas virtuais heterogêneas. A solução proposta permite a publicação e distribuição de *VAPs* que dependam dos mais variados tipos de monitores de máquinas virtuais.

Extensibilidade. Esta característica pressupõe que o sistema permite a sua adequação a outras situações, imprevistas na sua concepção. Ou mesmo o uso de outras tecnologias. Nesse sentido, uma arquitetura fracamente acoplada, composta por componentes especializados, contribui para o uso de outras tecnologias e ou adaptações futuras a cenários diversos. Um sistema com extensibilidade baixa é aquele projetado para um determinado cenário, provendo somente interfaces pré-definidas para administradores do sistema e usuários finais. Extensibilidade média significa que o sistema suporta algum tipo de adequação para novos cenários e ou contextos. Por outro lado, um sistema com extensibilidade alta é aquele concebido para atender cenários e contextos diversos, composto por componentes fracamente acoplados que podem ser adaptados e ou criados conforme as necessidades de cada ambiente e ou caso de aplicação.

`Scripts` IBM: extensibilidade **média**. Extender a solução para outros ambientes é um processo manual, que depende da criação de novos *scripts*, específicos cada *MMV*. A

solução é desprovida de uma arquitetura capaz de permitir a inclusão de novos recursos e novos ambientes.

Collective: extensibilidade **baixa**. A solução proposta e implementada pelo Collective é destinada a distribuição de *PCs* virtuais aos usuários finais. A sua concepção não leva em consideração ambientes diversos, com potenciais características e demandas particulares.

FlexVAPs: extensibilidade **alta**. No projeto do sistema existiu uma preocupação especial com relação a possibilidade de extensão e uso da solução em diferentes contextos e ambientes. Os usuários podem rapidamente criar e ou adaptar componentes para atender os mais variados tipos de demandas e particularidades da infraestrutura computacional local.

Escalabilidade teórica. A solução proposta pelo sistema deve levar em consideração questões de escalabilidade, onde o aumento progressivo de usuários e equipamentos que utilizam a solução não implique em problemas de desempenho ou funcionamento do sistema. O uso de infraestruturas distribuídas, escaláveis, é uma alternativa prática que ameniza o impacto negativo da escalabilidade teórica de sistemas desse gênero. Exemplo: considerando que o sistema funcione a partir de repositórios de dados. Neste caso, a estrutura de repositórios pode ser replicável e ou distribuível de maneira a evitar problemas com o aumento do número de usuários que necessitam fazer uso dos dados disponíveis nos repositórios. Um sistema com escalabilidade teórica baixa tem como objetivo de aplicação primária um sistema local, ou seja, ele é desprovido de recursos que permitam sua aplicação em escalas maiores (redes locais e ou de longa distância). A escalabilidade teórica média parte do princípio que o sistema possui e ou permite a utilização de algum recurso que escale a sua aplicação em redes locais com tecnologias iguais ou superiores a Fast e Gigabit Ethernet. Um sistema com escalabilidade teórica alta, por sua vez, parte do princípio que seu uso pode ser estendido além das redes locais, mesmo que isso impacte em mecanismos complementares para suporte de replicação e sincronização de dados.

`Scripts` IBM: escalabilidade teórica **baixa**. A solução não provê uma infraestrutura para a distribuição e ativação de *VAPs* em larga escala. O conjunto de ferramenta necessita de algumas intervenções manuais dos administradores de sistemas para instanciar as máquinas virtuais nos sistemas hospedeiros.

Collective: escalabilidade teórica **média/alta**. A arquitetura do Collective foi proje-

tada para atender usuários finais a partir de duas estruturas de dados centrais, o repositório de *virtual appliances* e o repositório de dados dos usuários. Esses dois repositórios acabam sendo os pontos limitantes da escalabilidade do sistema, considerando casos de maior abrangência. Porém, o sistema eficiente de *cache* prevê o uso do sistema pela Internet, disponibilizando *PCs* virtuais para os usuários, não importando aonde eles estejam, dede que estejam conectados.

FlexVAPs: escalabilidade teórica **média**. A concepção do *FlexVAPs* pressupõe a existência de um repositório global. Mas, esse repositório pode ser distribuído e ou replicado em diferentes *hosts*. Essa possibilidade leva o sistema a uma escalabilidade teórica boa. Na prática, algumas questões técnicas e tecnológicas poderão influenciar essa escalabilidade, porém, mesmo assim, a concepção do sistema propicia uma disponibilidade alta e natural, especialmente em redes locais. O principal custo da escalabilidade estará atrelado a replicação dos dados do repositório. O trabalho [75] responde as principais questões inerentes a criação e manutenção de réplicas para disponibilidade de serviços.

Gerenciamento. Administrador e usuários do sistema devem tem condições de realizar suas atividades, como publicação (administradores) e utilização (usuários) de *VAPs*, de forma rápida e prática. Os administradores do sistema devem ser capazes de publicar *virtual appliances* através da disponibilização e configuração auxiliar de um pacote de dados (arquivo do *VAP*). Já o usuário deve ter como única tarefa a escolha do *VAP* a ser utilizado. Gerenciamento **alto** significa que o administrador e ou usuário possui controle total sobre o sistema, podendo fazer o que quiser. Isso inclui poder modificar o sistema para atender as suas necessidades mais particulares. Flexibilidade gerencial alta significa que o usuário final pode realizar fáceis e rápidas modificações nos recursos e ferramentas disponíveis no sistema. Para isso, ele deve ser aberto e concebido em algum tipo de linguagem e ou tecnologia conhecida pelos usuários. Uma flexibilidade gerencial média implica em um sistema que permite alguma intervenção menor por parte dos usuários. Já um sistema de flexibilidade gerencial baixa permite que os usuários utilizem somente as suas funcionalidades e facilidades pré-definidas.

Gerenciamento (administradores)

`Scripts` IBM: flexibilidade gerencial **alta**. Os administradores do sistema tem o domínio completo sobre o sistema de maneira simples e prática, pois ele é constituído de *scripts*, algo do dia-a-dia desses usuários.

Collective: flexibilidade gerencial **baixa/média**. O sistema, dada a complexidade e as tecnologias utilizadas, não é muito flexível e ou simples aos administradores do sistemas. Estes, basicamente, apenas controlam a publicação dos *virtual appliances*. No caso de potenciais modificações e ou intervenções no sistema, os administradores deverão conhecer e lidar com tecnologias as vezes pouco conhecidas e ou de uma curva de aprendizado maior, como é o caso do `Java`.

FlexVAPs: flexibilidade gerencial **alta**. A arquitetura do sistema é composta por componentes simples, especializados, desenvolvidos em linguagens comuns aos administradores de sistemas, como `Perl` e `Bash` *Scripting*. Além disso, todas as ferramentas e servidores, como `Apache2`, `wget` e `zsync`, fazem parte do dia-a-dia de administradores de sistemas.

Gerenciamento (usuários)

IBM *Scripts*: flexibilidade gerencial **alta**. No caso, os usuários são os próprios administradores de sistemas, pois são eles quem irão gerenciar os *virtual appliances* destinados a serviços e soluções de rede.

Collective: flexibilidade gerencial **baixa**. Os usuários finais meramente irão utilizar a interface do sistema e escolher os *VAPs* desejados. O usuário não tem praticamente nenhuma ação ou possibilidade de gerenciamento no sistema.

FlexVAPs: flexibilidade gerencial **alta**. Os usuários finais podem, além de simplesmente utilizar as interfaces disponíveis, podem gerenciar as configurações e ou alterar propriamente dito os componentes dos *hosts* hospedeiros. Além disso, eles podem também criar novos componentes capazes de tirar proveito da infraestrutura disponível, atendendo, eventualmente, a particularidades individuais. Em alguns desses casos, o usuário terá que ter também permissões administrativas no sistema local.

A tabela 5.1 sumariza o comparativo das características a pouco discriminadas entre as três soluções de sistemas para o gerenciamento de *virtual appliance*. Como pode ser observado, há algumas características que são similares entre os sistemas. Em especial, existem três itens onde o *FlexVAPs* se destaca, sendo eles a adaptabilidade, heterogeneidade e extensibilidade. Isso deve principalmente ao fato de ele ser constituído por um conjunto de componentes especializados e fracamente acoplados, o que permite destacar-se nessas características.

Característica/Sistema	`Scripts` IBM	Collective	*FlexVAPs*
Adaptabilidade	**Média**	**Baixa**	**Alta**
Heterogeneidade	**Média**	**Baixa**	**Alta**
Extensibilidade	**Média**	**Baixa**	**Alta**
Escalabilidade	**Baixa**	**Média/Alta**	**Alta**
Flexibilidade gerencial (admins)	**Alta**	**Baixa/Média**	**Alta**
Flexibilidade gerencial (usuários)	**Alta**	**Baixa**	**Alta**

Tabela 5.1: Sumário do comparativo de características entre os sistemas

Monitores de Máquinas Virtuais. O suporte à diversidade de monitores de máquinas virtuais é atualmente importante. Os diferentes *VAPs*, geralmente específicos aos respectivos *MMVs*, precisam ser atendidos pelo sistema de forma transparente aos usuários finais. Nesse sentido, é imprescindível que o sistema tenha um bom suporte a inclusão e distribuição de *VAPs* para máquinas virtuais heterogêneas (que dependem de diferentes *MMVs*, como Xen, VMware, VirtualBox, KVM, Qemu, entre outros). A tabela 5.2 sumariza os monitores de máquinas virtuais suportados pelos sistemas. No caso do `Scripts` IBM, diferentemente do *FlexVAPs*, é necessário criar novos *scripts* de gerenciamento para cada novo *MMV*.

MMV/Sistema	`Scripts` IBM	Collective	*FlexVAPs*
Xen	**Sim**	**Não**	**Sim**
VMware	**Sim**	**Sim**	**Sim**
KVM	**Não**	**Não**	**Sim**
Qemu	**Não**	**Não**	**Sim**
VirtualBox	**Não**	**Não**	**Sim**
Outros	**Não**	**Não**	**Sim**

Tabela 5.2: Monitores de Máquinas Virtuais suportados pelos sistemas

A principal vantagem do *FlexVAPs* é sua arquitetura, que permite trabalhar de forma transparente ao usuários final com diferentes monitores de máquinas virtuais. Isso é especialmente interessante devido a diversidade de soluções de virtualização existentes e as demandas bastante variáveis dos usuários finais.

A seguir são relacionadas algumas limitações e outras características. A tabela 5.3 apresenta um sumário comparativo entre os três sistemas.

Nível dos usuários administradores: grau de conhecimento requerido aos usuários ad-

ministradores do sistema.

Nível dos usuários finais: grau de conhecimento necessário aos usuários finais do sistema.

Tempo até a publicação do *VAP*: tempo necessário desde a solicitação do *VAP* até a publicação efetiva no repositório. No caso do Collective, esse tempo é baixo principalmente pelo fato de ser uma solução baseada em um único monitor de máquinas virtuais, o VMware. Com isso é possível reduzir o tempo e mais facilmente automatizar algumas etapas do processo que vai desde a solicitação até a publicação do *VAP*.

Mecanismo de *cache* de *VAPs*: consiste no armazenamento local temporário ou permanente do *VAP*, sem a necessidade de realizar o *download* do mesmo do repositório central a cada uso.

Uso de repositório central: uso de um sistema de armazenamento central, único, para os *virtual appliances* gerenciados pelo sistema.

Dependência da rede: define o quanto o sistema depende da rede para operação.

Dependência de outros dispositivos: alguns sistemas podem depender de dispositivos virtuais, como disquetes virtuais, como fonte de entrada de dados.

Solução aberta e livre: é a característica de a solução ser pública, podendo qualquer um utilizá-la, adaptá-la ou incrementá-la de acordo com as suas necessidades. Essa é uma característica importante para muitos usuários e casos de aplicação.

Como pode ser observado nas tabelas 5.1, 5.2 e 5.3, há vantagens e desvantagens em cada um dos sistemas. A escolha entre um ou outro depende das demandas do ambiente e ou dos usuários.

5.1.2 Comparativo de Procedimentos

Criação e publicação de *VAPs*

A seguir são relacionadas algumas das etapas, algumas vezes necessárias, entre a solicitação e a publicação do *virtual appliance*. A tabela 5.4 sumariza essas etapas, relacionando-as aos sistemas em avaliação.

Característica/Sistema	`Scripts` IBM	Collective	*FlexVAPs*
Nível dos usuários administradores	**Alto**	**Baixo**	**Médio**
Nível dos usuários finais	**Alto**	**Baixo**	**Baixo/Médio**
Tempo até a publicação do *VAP*	**Médio**	**Baixo**	**Médio**
Mecanismo de *cache* de *VAPs*	**Não**	**Sim**	**Sim**
Uso de repositório central	**Não**	**Sim**	**Sim**
Dependência da rede	**Não**	**Sim**	**Sim**
Dependência de outros dispositivos	**Sim**	**Não**	**Não**
Solução aberta/livre	**Sim**	**Não**	**Sim**

Tabela 5.3: Limitações e outras características dos sistemas

1. Criação. Em todos os três sistemas essa fase existe e é obrigatória.

2. Configuração. Em todos os três sistemas essa fase existe e é obrigatória.

3. Preparação dos *scripts* de ativação. Esta etapa inexiste no Collective. Já no *FlexVAPs* ela é opcional, podendo ou não ser utilizada.

4. Publicação em repositório. A solução dos `Scripts` IBM permite o uso de várias alternativas, como diretórios `NFS`, Web, locais (acesso via *FTP* ou *SSH*, por exemplo) ou mesmo mídias removíveis.

5. *Download* para o sistema local. No caso do `Scripts` IBM, essa etapa pode representar um simples processo de cópia a partir de uma mídia móvel, como um disco USB.

6. Ativação. É a última etapa do processo. Ela é parte de todos os sistemas.

Etapa/Sistema	`Scripts` IBM	Collective	*FlexVAPs*
1 - Criação	**Obrigatória**	**Obrigatória**	**Obrigatória**
2 - Configuração	**Obrigatória**	**Obrigatória**	**Obrigatória**
3 - Preparação	**Obrigatória**	**Inexistente**	**Opcional**
4 - Publicação	**Inexistente**	**Obrigatória**	**Obrigatória**
5 - *Download*	**Opcional**	**Obrigatória**	**Obrigatória**
6 - Ativação	**Obrigatória**	**Obrigatória**	**Obrigatória**

Tabela 5.4: Etapas entre a solicitação e a publicação do *virtual appliance*

Cenário convencional *versus* proposto

A seguir são apresentados os passos que seriam necessários em um ambiente convencional para a criação e utilização de *VAPs*. O tempo gasto, na média, será na ordem de algumas horas, visto que compete ao usuário final a execução de todas as etapas relacionadas a seguir.

1. escolher, instalar e testar um *MMV*;
2. criar, configurar e instalar um *VAP*;
   (a) criar e configurar uma *MV*;
   (b) escolher, instalar e testar um *SO*;
   (c) instalar e testar as aplicações;
3. copiar o *VAP* para o(s) *host* (s);
4. utilizar o ambiente virtual.

Por outro lado, a proposto deste trabalho minimiza os esforços e as demandas das etapas 1, 2 e 3. Estas serão atendidas pelo administrador do sistema. Como ele possui experiência com sistemas e modelos (*templates*) prontos, pré-configurados, para os mais variados tipos de demanda, atender as necessidades do usuário será algo tranqüilo e relativamente rápido (na média, na ordem de minutos). Neste contexto, cabe ao usuário final apenas usufruir da etapa 4, ou seja, utilizar o ambiente virtual, economizando tempo e esforços que seriam necessários em um ambiente convencional.

O gráfico da figura 5.1 ilustra uma escala de tempo necessário nos dois cenário, convencional e proposto. As medidas são em unidades de tempo. No caso, é considerado que um administrador de sistemas gasta apenas 50% do tempo que um usuário normal gastaria na preparação de um ambiente virtual. Na prática, essa proporção pode variar para mais ou para menos, sendo altamente influenciada pela experiência e agilidade do usuário final e do administrador do sistema.

Na figura 5.1 são apresentados também dois hipotéticos cenários convencionais 2 e 3. Eles representam as dificuldades que podem advir da etapa 3. A cópia do *VAP* para os *hosts* destino pode ser algo relativamente simples, algo que demanda um certo trabalho, bem como algo complexo, pois depende muito dos recursos disponíveis no ambiente de computação. É bastante comum a inexistência de meios e mecanismos, para os usuários

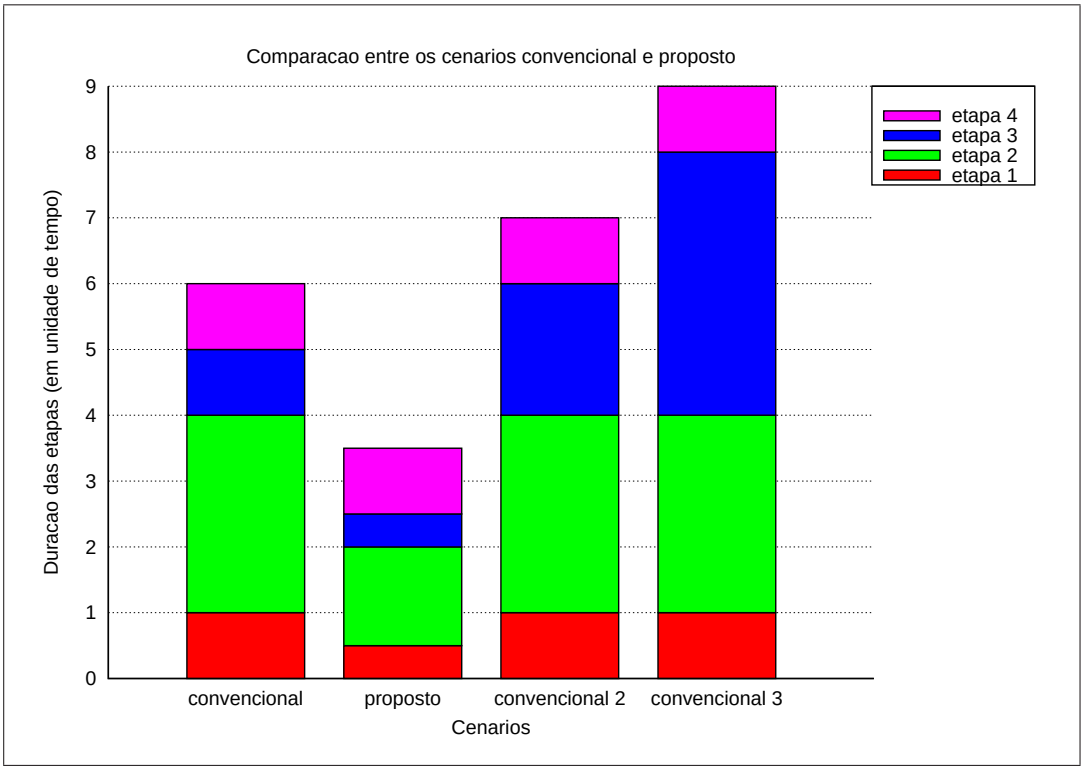

Figura 5.1: Cenário convencional *versus* cenário proposto)

finais, de cópia em massa de arquivos para múltiplos destinos (*PCs*). Sendo assim, o tempo da etapa 3, num cenário convencional qualquer, pode variar de forma significativa.

Utilização de *VAPs*: processo manual *versus* sistema

O objetivo desta seção é estabelecer uma comparação entre os procedimentos manual e automatizado de ativação de *VAPs*. A comparação toma como base apenas as tarefas do usuário final. É considerada a existência de um repositório global de *virtual appliances*, conforme previsto na proposta do trabalho.

O repositório global contém *VAPs* para diferentes fins, como aulas, administração de redes, desenvolvimento de *software* e ambientes de pesquisa. O acesso a alguns conjuntos de *VAPs* (diretórios do repositório global) é restrito a determinados grupos de usuários. Nestes casos, é necessário um processo prévio de autenticação para acesso aos *virtual appliances* propriamente ditos.

Caso 1. Considerando um *VAP* público, os procedimentos manuais são os sequintes:

1. abrir em um navegador a *URL* do repositório global;

2. dentre os arquivos (*VAPs*) listados no diretório Web, escolher o mais adequado a necessidade do usuário;

3. identificar os comandos e ou baixar os *scripts* de inicialização do *VAP*;

4. verificar se o sistema hospedeiro está preparado para instanciar a *MV* (em caso negativo, trocar de *VAP* ou preparar o sistema local);

5. executar os comandos e ou *scripts* de ativação do *VAP*.

O primeiro passo requer que o usuário conheça a *URL* do repositório de *VAPs*. Caso estejam disponíveis vários servidores de *virtual appliances* na rede, o usuário teria que conhecer todas as respectivas *URLs* para descobrir se a sua demanda pode ser suprida por alguma das partes (servidores) do repositório global.

O segundo passo exige um conhecimento prévio do usuário acerca do conteúdo de cada arquivo (*VAP*) e ou a abertura manual dos respectivos arquivos de descrição. Isso pode ser um processo relativamente complicado para o usuário final.

A terceira etapa do processo consiste em identificar os comandos e ou *scripts* de inicialização do *VAP*. Cabe ao usuário certificar-se da utilização dos recursos corretos para a instanciação do *virtual appliance*.

Os dois últimos passos correspondem, respectivamente, a verificação do sistema local (ver se o *MMV* correto está disponível) e a execução do comando e ou *script* de ativação propriamente dito.

Caso 2. Considerando um *VAP* de acesso restrito, os procedimentos são similares ao caso 1. A diferença está entre o primeiro e o segundo passos, onde há um procedimento intermediário de autenticação (usuário e senha) e autorização (verificação das permissões para virtualizar os *VAPs* do respectivo diretório.

Tanto no caso 1, quanto no caso 2, os mesmos procedimentos via interface do sistema são simplificados. O usuário final precisa apenas utilizar o comando e escolher o *virtual appliance*. No caso de conjuntos restritos de *VAPs*, o usuário precisará passar por parâmetro ou entrada padrão os dados (usuário e senha) para verificação de autenticação e autorização.

A figura 5.2 ilustra a diferença de tempo e esforço entre o processo manual e o proposto (sistema). Há quatro casos representados: 1) manual sem autenticação; 2) manual com autenticação; 3) sistema sem autenticação; e 4) sistema com autenticação. A diferença entre com e sem autenticação, tanto num caso quanto noutro, é de uma unidade de tempo. A maior diferença está entre o processo manual e o sistema, perfazendo 4 unida-

des de tempo. Em síntese, o processo via sistema necessita apenas 1 quinto do esforço por parte do usuário em relação ao processo manual.

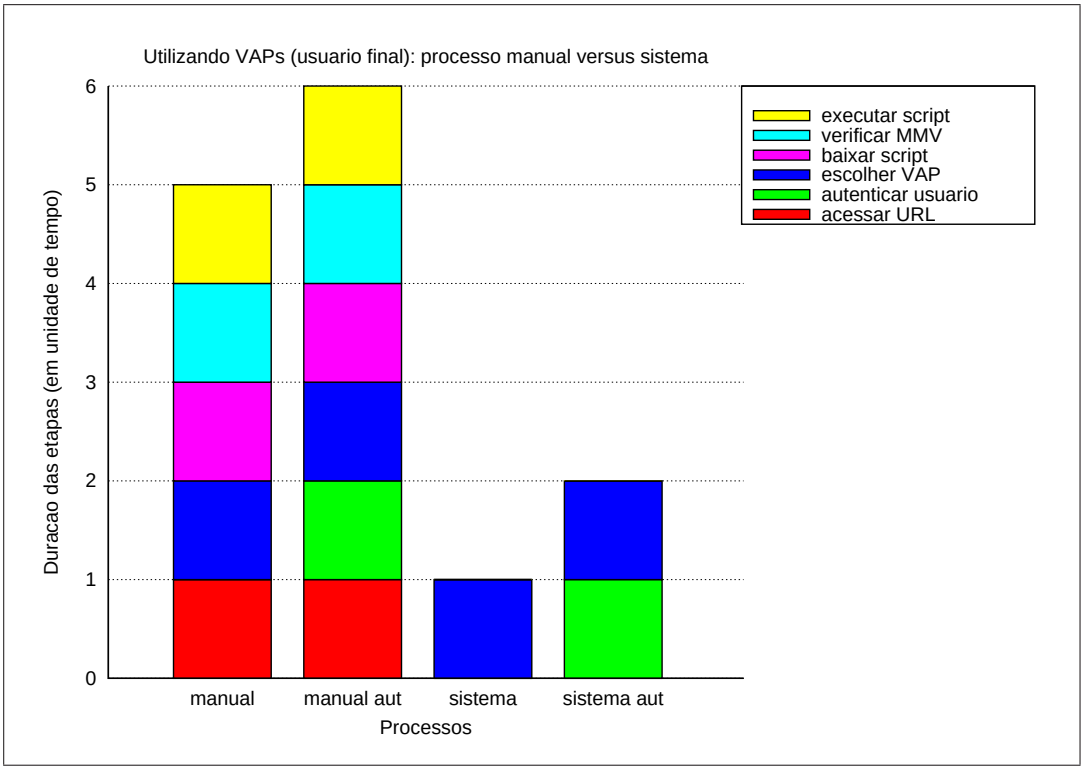

Figura 5.2: Utilização de *VAPs*: processo manual *versus* sistema

5.2 Análise Quantitativa

A seguir são apresentados alguns dados quantitativos de utilização prática da solução proposta. A seção 5.2.1 apresenta alguns dados sobre o repositório de *VAPs* utilizado nos testes do protótipo da solução proposta. Na seqüência, a seção 5.2.2 apresenta alguns resultados numéricos da utilização de ferramentas como a interface do usuário (`vapdeploy`).

5.2.1 Repositório de *VAPs*

O repositório atual de *VAPs* contém mais de 20 *GB* de dados. Ele é composto por uma estrutura de 12 diretórios. Alguns são meramente de uso experimental, enquanto outros armazenam *VAPs* prontas para serem utilizadas na prática.

```
clients   inacessible  private   students
develop   labmov       public    teachers
dsd       network      servers   templates
```

Cada diretório possui um controle de acesso por grupos de um diretório *LDAP*. A seguir é apresentado um exemplo de grupo utilizado para controlar o acesso aos dados do diretório `private`. Como pode ser observado, apenas um único usuário, `dlk`, tem permissão para acessar as *VAPs* desse diretório.

```
# private, groups, intranet
dn: cn=private,ou=groups,dc=intranet
userPassword:: e2NyeXB0fSo=
objectClass: posixGroup
objectClass: intranetGroup
objectClass: top
cn: private
gidNumber: 10008
uniqueMember: uid=dlk,ou=users,dc=intranet
```

A seguir é apresentada a respectiva configuração do `Apache2` para o controle de acesso ao diretório `private`. Pode-se observar que o controle é realizado pelo grupo *LDAP* (`ldap-group`).

```
Alias /flex/VAPs/private "/www/flex/VAPs/private"
<Directory /www/flex/VAPs/private/>
    Options Indexes FollowSymLinks MultiViews
    AllowOverride all
    Order allow,deny
    allow from all
    AuthType Basic
    AuthName  "FlexVAPs - auth required"
    AuthBasicProvider ldap
    AuthLDAPURL ldap://ldap.vap.intranet/...
    require ldap-group cn=private,ou=groups,dc=intranet
</Directory>
```

Algumas estatísticas do repositório:

Tamanho: 23.880 *MB*.

Diretórios: 12.

Número de *VAPs*: 19.

Número de *VAPs* meramente experimentais: 13.

Total de *VAPs*: 32.

Maior *VAP*: 2.001 *MB*.

Menor *VAP*: 296 *MB* (compactada).

Tamanho médio das *VAPs*: 1.256 *MB*.

As *VAPs* disponíveis são para os *MMVs* Xen, KVM e VMware. Elas são específicas, ou seja, para um dos respectivos monitores de máquinas virtuais. Sendo assim, há *VAPs* que irão demanda o Xen, o KVM ou o VMware. E, elas poderão ser instanciadas somente em sistemas que possuírem o *MMV* necessário.

Exemplo de *VAPs* disponíveis:

```
network/jaunty_apache2.ext3.xml
network/jaunty_ldap.ext3.xml
network/jaunty_firewall.ext3.xml
network/jaunty_wordpress.ext3.xml
network/jaunty_mysql.ext3.xml
templates/jaunty_base.img.xml
templates/VMware_VM_Debian_Template.zip.xml
templates/VMware_VM_Ubuntu_Desktop_Template.zip.xml
templates/VMware_VM_Ubuntu_Server_Template.zip.xml
public/x_jaunty_base.ext3.gz.xml
public/x_jaunty_base.ext3.xml
students/jaunty_net.ext3.xml
students/jaunty_sniffers.ext3.xml
develop/jaunty_tomcat.ext3.xml
develop/jaunty_java.ext3.xml
develop/jaunty_cplusplus.ext3.xml
```

Aquivos `.xml` (*XML*) representam os arquivos de configuração das *VAPs*. Eles contém as informações básicas sobre o sistema da *VAP* e outras coisas, como *MMVs* requeridos/suportados, *scripts* de inicialização, *scripts* de configuração durante o processo de ativação, *scripts* de instalação dos *MMVs*, *timestamp* de *cache*, *timestamp* de sistema de arquivos local e comandos para decompressão. Com excessão do *script* de inicialização e dos *timestamps*, os demais dados citados são opcionais. O *timestamp* de *cache* é utilizado para determinar qual é o período de tempo que a *VAP* será utilizada diretamente da *cache* sem verificação com o repositório central. Isso deve-se ao fato de, na média, as *VAPs* serem bastante grande e o tempo de verificação de consistência ser relativamente oneroso, tampo tem termos de tempo quanto de computação, pois hoje ele é baseado em algoritmos de *hash MD5* e *SHA1*. O *timestamp* de sistema de arquivos local é utilizado para determinar quando a *VAP* deve ser instanciada de uma cópia nova, da *cache*. Esse *timestamp* pode ser reduzido ou prolongado, dependendo das necessidades do usuário (solicitante da *VAP*) e ou tipo de aplicação/finalidade.

O uso de *VAPs* compactadas é interessante para otimizar o uso da rede e da *cache* do sistema hospedeiro. Um exemplo simples é a *VAP* `x_jaunty_base`, disponível no

diretório público do repositório. Ela está disponível tanto na forma original (disco virtual provido pelo usuário requisitante da *virtual appliance*), quanto na forma de um arquivo compactado com `gzip`. Na forma original o disco virtual corresponde a 2001 *MB* de dados a serem transmitidos e armazenados. Já na forma compactada o tamanho cai para 296 *MB*, ou seja, uma diferença significativa, de praticamente 75%. Essa diferença é ilustrada na figura 5.3. Dependendo do uso e do conteúdo do disco virtual essa diferença poderá variar, para mais ou para menos. Contudo, no geral, teremos uma redução de tamanho e conseqüente espaço em disco que poderá ser útil, desejável, para otimizar tanto o espaço do repositório global, quanto da *cache* local, pois ambos tendem a crescer de forma vertiginosa com a disponibilização crescente de *VAPs*, já que os arquivos, na média, tendem a ser grandes.

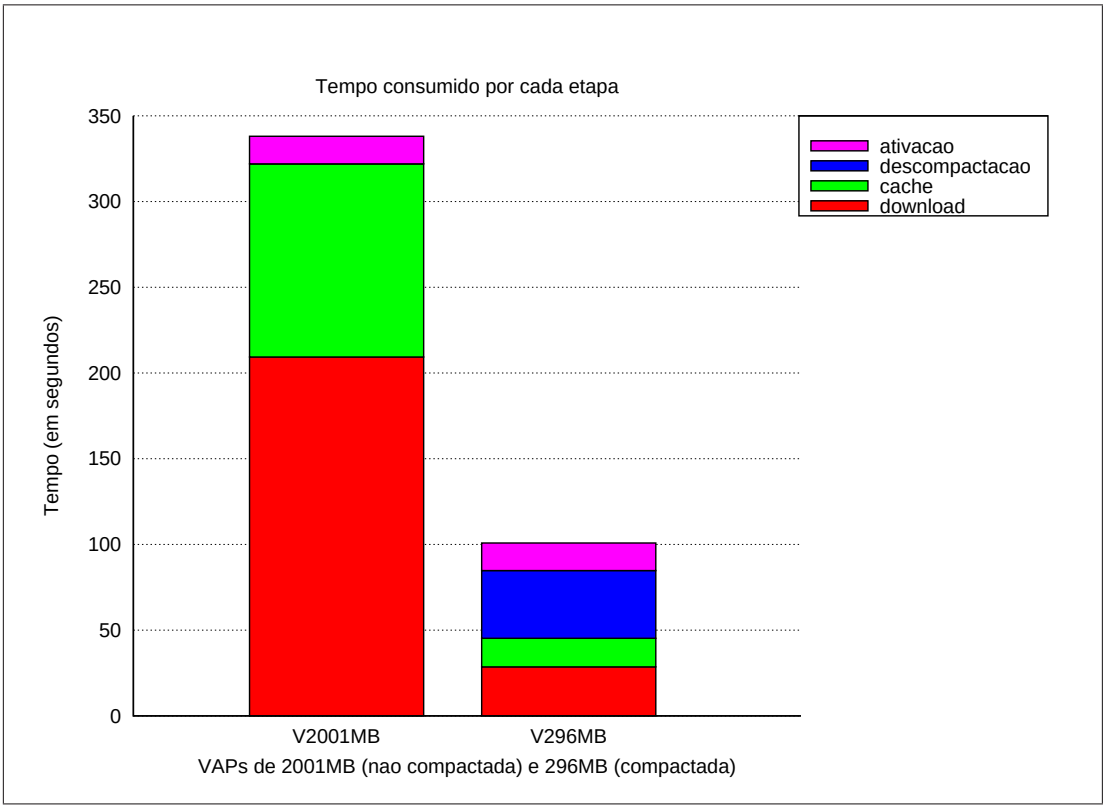

Figura 5.3: Tamanho de um *VAP* (não compactado e compactado)

5.2.2 *Download* e Ativação de *VAPs*

O ambiente de testes dos dados apresentados a seguir é constituído por uma rede Fast Ethernet, um *PC* servidor e *PCs hosts*, onde as *VAPs* são instanciadas. Tanto o servidor quanto os *hosts* rodam Ubuntu Desktop na versão 9.04.

O tempo de *download* e ativação de um *VAP* (`jaunty_base.img`[1]) de 2001 *MB* fica em torno de 322 segundos. O tempo foi obtido através da média de 5 *downloads* consecutivos do *VAP* (tempo de execução da interface do usuário). A maior parte do tempo é devido a transferência dos dados pela rede (em torno de 60%), . Em segundo lugar, vem a cópia da *cache* para o local de ativação (em torno de 35%). Por fim, o menor tempo é o da ativação propriamente dita (em torno de 5%).

O tempo de *download* de um *VAP* normal, sem compactação, e sua versão compactada pode diferir de forma significativa. A exemplo, um Ubuntu Server com um disco virtual de 2 *GB* leva 322 segundos entre o *download* para a *cache* local e a ativação propriamente dita. Já a versão compactada da *VAP*, agora com apenas 296 *MB*, leva 101 segundos para completar o mesmo processo. Sendo assim, pode-se concluir que, em alguns casos, poderá ser muito proveitoso em termos de eficiência compactar as *VAPs*, reduzindo o tráfego da rede e o tempo de *download* e armazenamento local. Além disso, isso também aumenta o número de *VAPs* comportadas no mesmo espaço de *cache*.

A figura 5.4 apresenta um gráfico da diferença de tempo para cada etapa de um mesmo *VAP*. No tamanho de 2001 *MB* ele está descompactado. Já no tamanho de 296 *MB* ele está compactado. Como pode ser observado, neste caso há um tempo considerável gasto na descompactação do *VAP*. Contudo, mesmo assim a diferença de tempo, considerando todas as etapas, fica em torno de 65%.

Uma das constatações é que o tamanho das *VAPs*, estejam elas compactadas ou não, deve ser o menor possível, o mais otimizado possível, contendo apenas e exatamente o necessário. Esse tamanho tem impacto desde a escolha do sistema operacionais. Um exemplo simples, considerando a instalação de um sistema básico, utilizando o `deboostrap`, de duas distribuições distintas: uma *VAP* Debian, para VMware, possui com um tamanho em torno de 300 *MB*, enquanto que a mesma *VAP*, com Ubuntu Server, atinge um tamanho próximo a 606 *MB*. Neste caso, temos uma diferença, ilustrada na figura 5.5, no tamanho de duas vezes, considerando apenas o sistema operacional básico.

Durante a fase de testes do sistema foram utilizadas *VAPs* para os *MMVs* Xen, KVM e VMware. O processo de *download* e ativação das *virtual appliances* é o mesmo, independente do monitor de máquinas virtuais. O que muda é o *script* e ou comando de inicialização da *MV*. Para o usuário final, tudo isso é transparente.

[1] Sistema básico de um Ubuntu Server

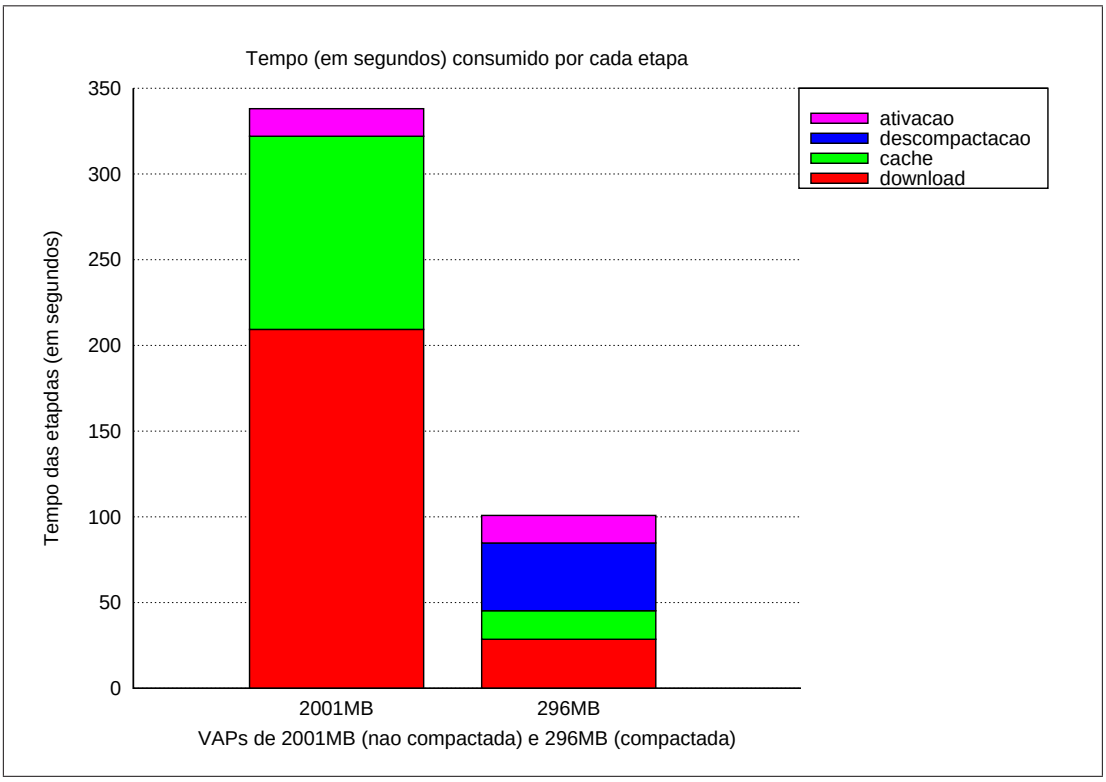

Figura 5.4: Tempo de cada etapa para um *VAP* (não compactado e compactado)

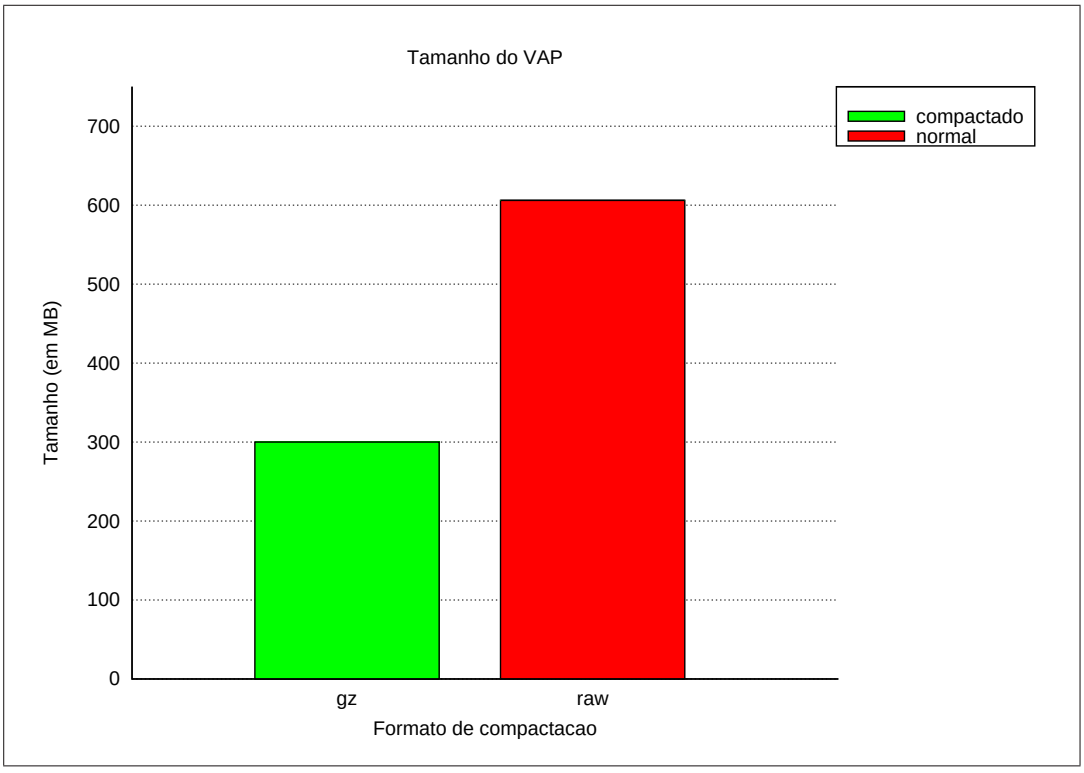

Figura 5.5: Tamanho de um *VAP* (compactado e não compactado)

Também foram realizados testes de ativação de *VAPs* com instalação de *MMVs*. Esta é realizada através de *scripts* preparados e testados pelo administrador do sistema. Nas

instalações experimentais de monitores de máquinas virtuais, o principal problema identificado foi o problema de compatibilidade. Um exemplo de incompatibilidade é o KVM e o Xen, já que eles utilizam versões de *kernel* do Linux distintas. Algo similar ocorre entre o FluxBox e o KVM, pois o primeiro não funciona com as `flags VMX` habilitadas no *kernel*. Em ambos os exemplos é necessário re-iniciar o sistema, além de instalar o *MMVs*. Outros casos, como o VMware, demandam que o sistema hospedeiro tenha instalado os fontes do *kernel*. Enfim, estes exemplos demonstram que em alguns casos o processo de instalação dos *MMVs* pode exigir etapas, preparativos e testes prévios.

5.3 Verificações Básicas

As seções 5.3.1, 5.3.2, 5.3.3 e 5.3.4 discriminam algumas verificações básicas realizadas no sistema prototipado. Elas representam os testes básicos das funcionalidades mais essenciais da solução proposta.

5.3.1 Ativação de um *VAP*

Considerando a arquitetura e os recursos disponibilizados pela solução arquitetura e prototipada neste trabalho, a ativação de um *VAP* é um dos processos mais simples e prático. Utilizando a interface do `vapdeploy`, o usuário consegue listar e escolher um dos *VAPs*, disponíveis no seu respectivo nível de permissão, do repositório global.

Etapas do processo, utilizando o `vapdeploy`:

1. listagem da relação de *VAPs* disponíveis ao usuário;
2. escolha do *VAP* desejado;
3. acionamento do *daemon* de *cache* para *download* do *VAP*;
4. cópia do *VAP* da *cache* para o diretório de trabalho do usuário, ou diretório temporário do sistema;
5. inicialização do *VAP*.

O tempo de *download* e inicialização do *VAP* irá depender de diferentes fatores, como capacidade da rede e da máquina hospedeira. Considerando um caso simples, utilizando uma rede cabeada de 100 *Mbps*, duas máquinas X (cliente) e Y (repositório) com *GNU/Linux* e um *VAP* de 500 *MB*, o tempo de *download*, utilizando o `wget`, fica

em torno de 81 segundos. Já o tempo de ativação, considerando o Xen, fica em torno de 17 segundos, perfazendo um tempo total aproximadamente 100 segundos, conforme ilustrado graficamente na figura 5.6.

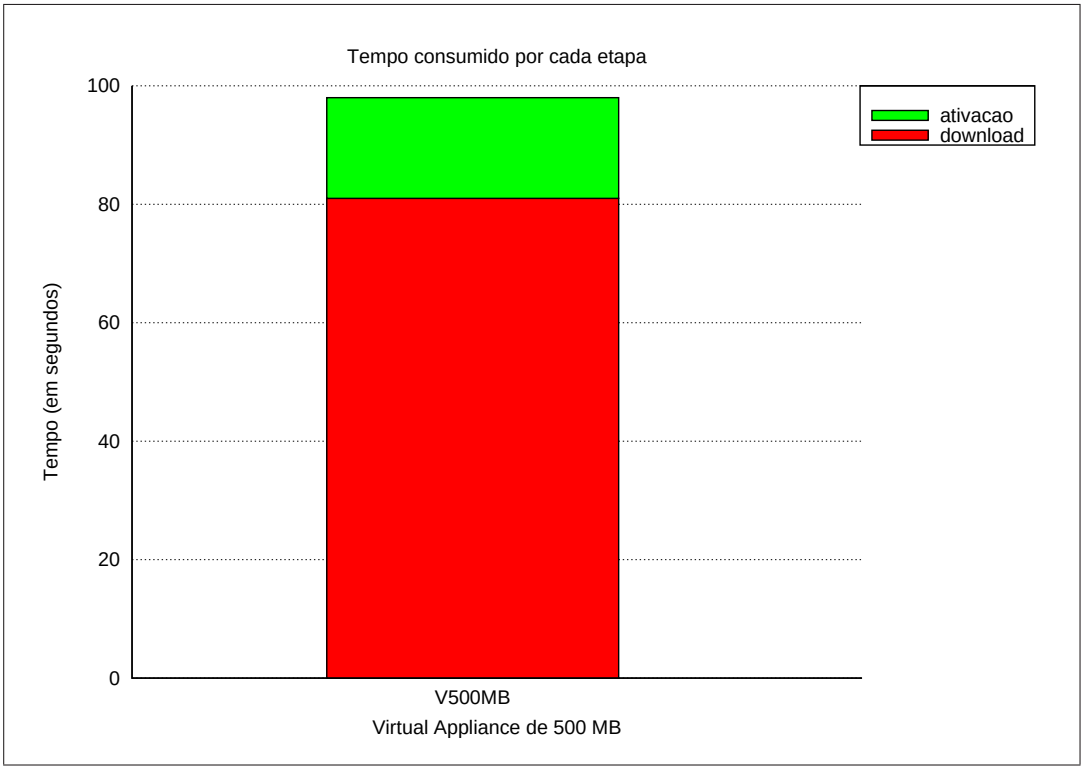

Figura 5.6: Tempo de *download* e ativação de um *VAP* de 500 *MB*

No caso do Collective, o processo é idêntico. Por outro lado, no caso do `Scripts` IBM há algumas intervenções manuais que são necessárias. Os itens 1, 2 e 3 não existem nessa solução. E o item 4 pode envolver comandos e ações manuais, pois o sistema não prevê o uso de um repositório central, por exemplo.

5.3.2 Ativação de *VAPs* com Instalação de *MMVs*

A ativação de *VAPs* com instalação de *MMV* é semelhante à ativação de um *VAP*, como apresentado na seção 5.3.1. Há apenas dois passos intermediários, como destacado a seguir.

1. listagem da relação de *VAPs* disponíveis ao usuário;
2. escolha do *VAP* desejado;
3. acionamento do *daemon* de *cache* para *download* do *VAP*;

4. cópia do *VAP* da *cache* para o diretório de trabalho do usuário, ou diretório temporário do sistema;

5. **o usuário escolhe o *MMV* a ser instalado para satisfazer as exigências do *VAP*** ;

6. **o instalador de *MMVs* é acionado, procedendo a instalação, caso o usuário seja autorizado, do novo *MMV*** ;

7. inicialização do *VAP*.

A instalação de um novo *MMV* no sistema hospedeiro depende de permissões que são atribuídas pelos administradores do sistema aos usuários. Considerando um usuário autorizado, o *daemon* de instalação de *MMVs* irá realizar o *download* do pacote de dados e do *script* de instalação do novo *MMV*. Caso tudo ocorra sem problemas, o *VAP* será instanciado sem maiores problemas.

Com relação a esse exemplo, nem o Collective e nem os `Scripts` IBM contemplam a instalação automática de novos monitores de máquinas virtuais. A base do Collective é sobre o VMware, enquanto que os `Scripts` IBM foram projetados para VMware e Xen, sendo necessário um novo conjunto de *scripts* específicos para suportar novos *MMVs*.

5.3.3 Ativação de *VAPs* com Satisfação de Dependências

A instalação de *VAPs* com satisfação de dependências agrega mais uma funcionalidade aos processos ilustrados nas seções 5.3.1 e 5.3.2. Além de verificar os dados necessários para a ativação do *VAP* selecionado pelo usuário, o `vapdeploy` também avalia as possíveis dependências do pacote. Caso hajam dependências, o usuário poderá escolher entre ativá-las ou não.

Caso o usuário optar por ativar também as dependências, o `vapdeploy` irá realizar o *download* e a ativação ordenada das dependências, conforme especificado no arquivo de configuração do *VAP*. Cada *VAP* presente na lista de dependências será ativada no sistema local. Ao final, a *VAP* original, escolhida pelo usuário, fechará o processo aberto pela requisição inicial do usuário.

Como podem haver vários níveis de dependência, o administrador do sistema pode configurar o nível máximo de análise. Sendo assim, o `vapdeploy` irá ir até o nível máximo definido no sistema.

O Collective suporta a configuração de dependência nos *VAPs*, de forma análoga ao *FlexVAPs*. O sistema da IBM, por outro lado, é desprovido de recursos para trabalhar com dependências de serviços, criando relações entre os *virtual appliances*.

5.3.4 Ativação de Grupos de *VAPs*

A ativação de grupos de *VAPs* é um processo diferenciado, visto que demanda a comunicação com grupos de *hosts* aonde serão ativados os *VAPs*. Um grupo de *VAPs* pode ser caracterizado como um conjunto de sistemas que serão inicializados para criar um determinado ambiente, normalmente com um fim definido. A exemplo, um grupo de *VAPs* pode criar um conjunto de pacotes, entre servidores e nós, destinados à criação de um *cluster*, ou um ambiente de computação distribuída.

Ao grupo de *VAPs* deve estar associado/definido um grupo de *hosts*. Os *VAPs* do grupo serão ativados nos respectivos *hosts*. Nesses termos, podem haver, por exemplo, *VAPs* servidores e clientes. Da mesma forma, podem existir *hosts* caracterizados como servidores, clientes ou quaisquer. *VAPs* servidores serão ativados em *hosts* servidores. *VAPs* clientes serão instanciados em *hosts* clientes. No caso dos *hosts* quaisquer, tanto *VAPs* servidores quanto clientes poderão ser instanciados.

Nem o Collective nem os `Scripts` IBM trabalham com grupos de *VAPs*. Esse recurso não fora previsto nesses sistemas.

5.4 Cenários de Aplicação

O objetivo desta seção é apresentar alguns cenários de aplicação da solução proposta. Áreas como desenvolvimento de software, manutenção de laboratórios de ensino, administração de redes e ambientes de pesquisa em geral podem beneficiar-se da solução proposta.

5.4.1 Desenvolvimento de Software

Os exemplos de aplicação podem ser variados, indo desde simples demandas de uso pessoal até complicados contextos de computação, como sistemas distribuídos. O objetivo é apresentar apenas alguns exemplos simples de como as *VAPs* podem contribuir no dia-a-dia dos usuários e administradores de sistemas.

Um primeiro exemplo são as ferramentas e demandas para o desenvolvimento de software. É relativamente comum diferentes ambientes possuírem peculiaridades de configu-

ração e utilização, o que dificulta a manutenção de diferentes versões e ou ferramentas em um mesmo sistema. Com o conceito e protótipo apresentados neste trabalho, torna-se possível criar e manter *VAPs* para cada versão e ou tipo de ferramenta de uma forma prática. A cada momento, o usuário pode selecionar a *virtual appliance* mais adequada ao seu trabalho e ou experimento. Neste caso, em uma disciplina como Laboratório de Desenvolvimento de Software para Dispositivos Móveis, podem ser criadas e disponibilizadas *VAPs* específicas, com diferentes versões de *SDKs* e ou plataformas de desenvolvimento de *software*, como Android, SymbianOS e JME. Algumas plataformas de desenvolvimento possuem, inclusive, restrições de sistemas operacionais, o que pode ser um problema com abordagens convencionais, pois pode ser necessário manter-se vários sistemas operacionais em uma mesma máquina física, o que gera trabalho e demandas administrativas periódicas. Por outro lado, com a utilização de *VAPs* o problema torna-se potencialmente menos impactantes, pois um único sistema operacional nas máquinas hospedeiras pode ser o suficiente para vários tipos de sistemas operacionais clientes, rodando sobre máquinas virtuais isoladas.

5.4.2 Laboratórios de Ensino

Laboratórios de ensino podem ser considerados um cenário típico de aplicação de soluções como a proposta neste trabalho. Um laboratório de ensino, na área da computação, tradicionalmente demandas de uma variedade de sistemas. Gerenciar essa variedade geralmente é um dos grandes desafios dos administradores de rede.

Os sistemas de gerenciamento de *virtual appliances* podem contribuir de forma significativa no gerenciamento de laboratórios de ensino. As demandas dos usuários, sejam eles professores, alunos ou mesmo visitantes eventuais, podem ser atendidas através de *VAPs* específicos. Com isso, os sistemas hospedeiros sempre estão livres de entulhos e lixos gerados pela instalação de programas e dados dos usuários. Estes passam a utilizar apenas máquinas virtuais, que podem ser inicializadas a partir de uma imagem original toda vez que alguém for utilizá-las. Isso evita uma série de problemas gerenciais de manutenção e problemas desses ambientes.

Exemplo de uso prático em disciplina da graduação. Suponha-se que o professor da disciplina de banco de dados esteja pesquisando material na Web para uma aula nova, diferente. Durante a busca ele encontra disponível na Internet um *virtual appliance* de

um sistema novo, interessante. Então, o professor decide utilizar esse *VAP* para a sua aula. Considerando uma infraestrutura de publicação e distribuição de *virtual appliances*, o professor precisaria apenas encaminhar o pacote de dados ao administrador do sistema, para publicação. Uma vez publicado, o *VAP* poderá ser selecionado e utilizado pelos alunos da disciplina, de forma simples e transparente.

5.4.3 Administração de Redes

O gerenciamento de redes é uma tarefa que muitas vezes exige um maior nível de conhecimento e experiência dos administradores de sistemas. Um dos problemas é o grande número de serviços e inter-dependências entre eles. Neste sentido, o uso de *virtual appliances* com possibilidade de estabelecer co-relações (dependências) entre eles pode ser uma alternativa para amenizar o problema.

A configuração das *VAPs* que serão instanciados nos sistemas hospedeiros podem incluir informações sobre dependências de sistemas. Exemplo: para criar uma rede local convencional podem ser necessários serviços como *DNS*, *DHCP*, *LDAP* e *SMTP*. Supondo que cada serviço esteja configurado em uma máquina virtual separada, a instanciação do *virtual appliance SMTP* pode implicar também na instanciação automática dos *VAPs DNS*, *DHCP* e *LDAP*. Com isso, o administrador do sistema não precisa conhecer os detalhes da inter-dependência dos serviços, além da ativação ser transparente e automática.

5.4.4 Ambientes de Pesquisa

Os ambientes de pesquisa podem ser bastante variados, indo desde um simples sistema até um conjunto de milhares de sistemas que compõe uma infraestrutura distribuída e complexa. Gerenciar essa variedade de ambientes é tradicionalmente um problema.

Uma forma de amenizar esse problema é através das infraestruturas de gerenciamento de *virtual appliances*. O simples fato de utilizar máquinas virtuais já é um fator que contribui para reduzir o problema de gerenciamento dos ambientes de pesquisa. Indo um pouco além, estabelecer grupos de *VAPs* e relações de dependência entre os *virtual appliances* pode ser uma alternativa para reduzir ainda mais o problema. Exemplo prático: considerando um ambiente distribuído composto de dez tipos distintos de sistemas. O usuário deseja criar de duas a oito instância de cada sistema, criando um ambiente distribuído diversificado.

5.4.5 Cenários Diversos e Considerações

As aplicações dos sistemas de compartilhamento e distribuição de *virtual appliances* vão bastante além dos exemplos apresentados nas seções anteriores. Há muitas aplicações que podem beneficiar-se dos *VAPs*. Exemplos incluem demandas de ambientes de contextos de cursos e treinamentos de informática, desenvolvimento e teste de *software* e mesmo os *PCs* dos usuários finais, cujo objetivo pode ser um simples teste e ou experimentação de novos ambientes e ou aplicações.

Todos os casos comentados estão em fase de experimentação prática. O maior trabalho tem residido na preparação das *VAPs* e no ajuste do *MMVs* nos sistemas hospedeiros, uma vez que a instalação nem sempre é simples e nem sempre funciona. Durante os testes, alguns *MMVs*, inclusive, apresentam problemas de compatibilidade. Por exemplo: KVM e FluxBox não conseguem co-existir, uma vez que o FluxBox requer um kernel com `VMX` desabilitado, enquanto que o KVM exige o oposto. Outro exemplo é o Xen, que exige um *kernel* modificado, gerando complicações para o uso simultâneo de outros *MMVs*, como é o caso do KVM. O sistema hospedeiro poderia ficar com dois ou mais *kernels* disponíveis e os *scripts* de ativação das *VAPs* identificar e indicar ao usuário a eventual necessidade de re-inicialização do sistema.

Com relação às *VAPs*, uma vez prontas, configuradas e disponibilizadas no repositório, o uso torna-se simples, podendo ser através das ferramentas do *FlexVAPs* ou por meio de um simples *download* e ativação manual. Para o usuário final o principal fator, que pode eventualmente pesar de forma negativa, é o tempo de *download*, seja para a *cache* seja para o sistema de arquivos aonde a *MV* será ativada.

5.5 Considerações

Na seção 5.1.1, em especial na tabela 5.2, pode ser observado que a pesquisa atingiu o objetivo proposto, ou seja, a proposição e verificação de um sistema capaz de trabalhar com o gerenciamento de *virtual appliances* para máquinas virtuais heterogêneas. Adicionalmente, o capítulo apresentou outros dados sobre a solução prototipada e cenários de aplicação.

Para fins de verificações essenciais da solução proposta, foi criado e experimentado um primeiro repositório de *VAPs*. Ele contém um conjunto de *VAPs* para fins de teste e utilização prática experimental.

Durante os experimentos práticos, os principais problemas identificados foram relacionados ao tempo necessário para o *download* (e *cache*) da *VAP* e conseqüente cópia para um diretório do sistema local (ativação). Quando a *VAP* está compactada, otimizando o uso da *cache*, ela é somente descompactada quando instanciada no sistema de arquivos local, ou seja, quando o usuário solicita a sua ativação.

Além do mais, é importante observar que o próprio processo de criação das *VAPs*, ou *templates*, com a escolha do sistema operacional, pode ter um impacto significativo no sistema, gerando mais ou menos tráfego na rede e tempo de espera por parte do usuário final. Logo, é importante reduzir-se ao máximo o tamanho das *VAPs* criadas e disponibilizadas no repositório.

6 CONCLUSÃO

Os monitores de máquinas virtuais representam uma das grandes tendências atuais da computação[20, 42]. Eles surgiram na década de 60 e 70[42, 74], porém, apenas agora estão causando um grande impacto nos sistemas de computação de um modo geral. Esse fato está atrelado principalmente as grandes evoluções de hardware que ocorreram durante as últimas décadas.

Com a evolução das ferramentas de virtualização emergiram novas aplicações, sendo que a maioria delas está sendo empacotada na forma de *virtual appliances*. As aplicações são as mais variadas possíveis, indo desde ambientes de ensino até sistemas de linha de produção. Essa vasta gama de aplicações, brevemente exploradas no decorrer do texto, demonstra ainda mais a tendência e o grande potencial por três dos monitores de máquinas virtuais. Em diferentes cenários reais eles são capazes de resolver desde problemas de segurança básicos até problemas de desempenho e escalabilidade. Um dos principais problemas que surge com essa variedade de cenários e aplicações é o gerenciamento das *VAPs* propriamente ditas. É neste ponto que concentrou-se a proposta deste trabalho, ou seja, desenhar e avaliar uma solução para o gerenciamento de máquinas virtuais heterogêneas.

A proposta de gerenciamento de *VAPs* para *MVs* heterogêneas parte do pressuposto da existência de um repositório global, centralizado ou distribuído, de *virtual appliances*. Os usuários finais, através de uma interface de interação com o sistema, podem visualizar as *VAPs* disponíveis e escolher uma delas para instanciação no sistema local. O *download* para a *cache* local é iniciado. Logo em seguida a *VAP* é copiada da *cache* para o sistemas de arquivos local (diretório do usuário, por exemplo) onde a máquina virtual é instanciada.

Os testes apresentados demonstram que a proposta desenvolvida neste trabalho é uma alternativa viável e aplicável para o gerenciamento de *virtual appliances* para máquinas

virtuais heterogêneas. Os resultados também permitem concluir que é necessário um certo nível de cuidado por parte dos administradores de sistemas, ou mesmo usuários, na preparação das *VAPs*. Quanto mais enxuto estiver e ou for o sistema melhor. Algumas vezes, a própria escolha do sistema operacional pode ter um impacto significativo no tamanho da *VAP* e consequentemente no tempo necessário para o *download* e posterior ativação.

Por fim, a concepção da solução, a implantação do protótipo e os resultados atingidos abriram muitas novas possibilidades de trabalhos futuros. Na lista podem ser incluídos: testes mais exaustivos com uma variedade maior de *MMVs*; fechamento da implementação da ativação dos grupos de *VAPs*; inclusão de mecanismos de programação automática da instanciação de *virtual appliances* (sem necessidade de intervenção humana); inclusão de ferramentas e mecanismos para identificação e cruzamento de demandas dos usuários com a disponibilidade de sistemas (*VAPs*) já pré-configurados; avaliação e inclusão de mecanismos de distribuição de carga entre servidores do repositório global; utilização de mecanismos de tolerância a falhas no sistema; investigação de soluções para aumentar a segurança e a eficiência do sistema; entre outras coisas.

REFERÊNCIAS

[1] Keith Adams and Ole Agesen. A comparison of software and hardware techniques for x86 virtualization. In *Twelth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XII)*, pages 2–13, San Jose, CA, USA, October 2000. ACM Press.

[2] Agren. Teaching computer concepts using virtual machines. *SIGCSEB: SIGCSE Bulletin (ACM Special Interest Group on Computer Science Education)*, 31, 1999.

[3] Bowen Alpern, Joshua Auerbach, Vasanth Bala, Thomas Frauenhofer, Todd Mummert, and Michael Pigott. Pds: a virtual execution environment for software deployment. In *VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, pages 175–185, New York, NY, USA, 2005. ACM.

[4] Charles Babcock. Mokafive virtual desktops: A flexible leash?, 2008. `http://www.informationweek.com/blog/main/archives/2008/05/mokafive_virtua.html`.

[5] Gaurav Banga, Peter Druschel, and Jeffrey C. Mogul. Resource containers: A new facility for resource management in server systems. In *OSDI*, pages 45–58, 1999.

[6] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, and Rolf Neugebauer. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, pages 164–177, Bolton Landing, NY, USA, October 2003. ACM.

[7] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In

SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles, pages 164–177, New York, NY, USA, 2003. ACM Press.

[8] Fabrice Bellard. QEMU, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46. USENIX, 2005.

[9] R. Bradshaw, N. Desai, T. Freeman, and K. Keahey. A scalable approach to deploying and managing appliances. In *TeraGrid 2007 Conference*, page 6, 2007. `http://workspace.globus.org/papers/Scalable_Approach_To_Deploying_And_Managing_Appliances.pdf`.

[10] Ceedo Technologies, Ltd. . Ceedo - flexible computing through virtualization, 2008. `http://www.ceedo.com/`.

[11] Ramesh Chandra, Nickolai Zeldovich, Constantine Sapuntzakis, and Monica S. Lam. The collective: A cache-based system management architecture. In *In Proc. 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, pages 259–272, 2005.

[12] Changhua, Le He, Qingbo Wang, and Ruth Willenborg. Simplifying service deployment with virtual appliances. *Services Computing, 2008. SCC '08. IEEE International Conference on*, 2:265–272, July 2008.

[13] Z. B. Daho and N. Simoni. Towards dynamic virtual private service networks: Design and self-management. In *10th IEEE/IFIP Network Operations and Management Symposium. NOMS 2006.*, pages 1–4, 2006.

[14] Jeff Dike. A User-Mode port of the linux kernel. In *Proceedings of the 4th Annual Showcase and Conference (LINUX-00)*, pages 63–72, Berkeley, CA, October 10–14 2000. The USENIX Association.

[15] Yaozu Dong. Xen and Intel virtualization technology for IA-64. In Anonymous, editor, *Proceedings of Gelato ICE: Itanium Conference and Expo: Spotlighting Linux on Itanium-based Platforms, October 1-4, 2006, Biopolis, Singapore*, 2006.

[16] R. Figueiredo, P. A. Dinda, and J. Fortes. Resource virtualization renaissance. *Computer*, 38:28–31, may 2005.

[17] David Frascone. Debugging kernel modules with user-mode Linux. *Linux Journal*, 97:76, 78, 80–81, May 2002.

[18] Tim Freeman. Virtualization and grid computing, 2008. `http://www.gridvm.org/`.

[19] F. Galan, D. Fernandez, J. Ruiz, O. Walid, and T. de Miguel. Use of virtualization tools in computer network laboratories. In *Proceedings of the FIfth International Conference on Information Technology Based Higher Education and Training. ITHET 2004.*, pages 209–214, 2004.

[20] G. Goth. Virtualization: Old technology offers huge new potential. *IEEE Distributed Systems Online*, 8:3–7, Feb 2007.

[21] Sachin Goyal and John Carter. A lightweight secure cyber foraging infrastructure for resource-constrained devices. In *WMCSA*, pages 186–195. IEEE Computer Society, 2004.

[22] Andreas Grau, Harald Weinschrott, and Christopher Schwarzer. Untersuchung der skalierbarkeit virtueller maschinen für den einsatz in rechnernetzemulation (evaluating the scalability of virtual machines for use in computer network emulation). Student report softwaretechnology, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, October 9, 2006.

[23] Le He, Shawn Smith, Ruth Willenborg, and Qingbo Wang. Automating deployment and activation of virtual images. *IBM WebSphere Developer Technical Journal*, 10(7), 2007. `http://www.ibm.com/developerworks/websphere/techjournal/0708_he/0708_he.html`.

[24] High Performance On Demand Solutions and VMware. Using vmware esx server with ibm websphere application server, 2006. `http://www.vmware.com/vmtn/resources/527`.

[25] Paco Hope. Using jails in freebsd for fun and profit. In *;login: The Magazine of USENIX & SAGE; June 2002 Volume 27, Number 3*. USENIX, 2002.

[26] Matthew E. Hoskins. User-mode linux. *Linux Journal*, 2006(145), May 2006.

[27] InstallFree Inc. Installfree, 2008. `http://www.installfree.com/`.

[28] Diego Kreutz. Conceitos e aplicações dos monitores de máquinas virtuais. Technical report, Programa de Pós Graduação em Informática (PPGI), Universidade Federal de Santa Maria (UFSM), Santa Maria, RS, 2007.

[29] Monica Lam, Mendel Rosenblum, Dan Boneh, Ramesh Chandra, Jim Chow, Tal Garfinkel, Jim Norris, Ben Pfaff, Joel Sandin, Constantine Sapuntzakis, Hovav Shacham, and Nickolai Zeldovich. The collective - a virtual appliance computing infrastructure, 2005. `http://suif.stanford.edu/collective/`.

[30] Peter S. Magnusson. The virtual test lab. *IEEE Computer*, 38(5):95–97, 2005.

[31] S. Mastrianni, D. F. Bantz, K. A. Beaty, T. Chefalas, S. Jalan, G. Kar, A. Kochut, D. J. Lan, L. O'Connell, A. Sailer, G. Wang, Q. B. Wang, and D. G. Shea. It autopilot: a flexible it service management and delivery platform for small and medium business. *IBM Syst. J.*, 46(3):609–624, 2007.

[32] Kirk McKusick. The jail facility in FreeBSD 5.2. *;login: the USENIX Association newsletter*, 29(4), August 2004.

[33] D. A. Menasce and M. N. Bennani. Autonomic virtualized environments. *Autonomic and Autonomous Systems, 2006. ICAS '06. 2006 International Conference on*, pages 28–28, July 2006.

[34] Daniel A. Menascé and Mohamed N. Bennani. Autonomic virtualized environments. In *ICAS*, page 28. IEEE Computer Society, 2006.

[35] Jason Nieh and Chris Vaill. Experiences teaching operating systems using virtual platforms and linux. *SIGOPS Oper. Syst. Rev.*, 40(2):100–104, 2006.

[36] Iain Oliver, Kristoffer Getchell, Alan Miller, and Colin Allison. Using disruptive technology for explorative learning. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 96–100, New York, NY, USA, 2007. ACM Press.

[37] Pradeep Padala, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kenneth Salem. Adaptive control of virtualized resour-

ces in utility computing environments. In *EuroSys '07: Proceedings of the 2007 conference on EuroSys*, pages 289–302, New York, NY, USA, 2007. ACM Press.

[38] PANORAMA. Panorama unveils world's first bi virtual appliance. *DATABASE Trends and Applications*, 21(6):1, 2007. `http://www.panoramasoftware.com/documents/dbta-article-on-panorama-virtual-appliance.pdf`, `http://www.dbta.com`.

[39] Positive Software Corporation. Free virtual private server solution, 2007. `http://www.freevps.com/`.

[40] Rare Ideas, LLC et al. Portableapps.com - your digital life anywhere, 2008. `http://portableapps.com/`.

[41] Dirk Riehle, Steven Fraleigh, Dirk Bucka-Lassen, and Nosa Omorogbe. The architecture of a UML virtual machine. In *OOPSLA*, pages 327–341, 2001.

[42] Mendel Rosenblum and Tal Garfinkel. Virtual machine monitors: Current technology and future trends. *IEEE Computer*, 38(5):39–47, 2005.

[43] Paul Ruth, Xuxian Jiang, Dongyan Xu, and Sebastien Goasguen. Virtual distributed environments in a shared infrastructure. *IEEEC*, 38(5):63–69, May 2005.

[44] Paul Ruth, Phil McGachey, Xuxian Jiang, and Dongyan Xu. Viocluster: Virtualization for dynamic computational domains. In *IEEE International Conference on Cluster Computing (Cluster 2005)*, 2005.

[45] Constantine Sapuntzakis, David Brumley, Ramesh Chandra, Nickolai Zeldovich, Jim Chow, Monica S. Lam, and Mendel Rosenblum. Virtual appliances for deploying and maintaining software. In *Proceedings of the Seventeenth Large Installation Systems Administration Conference (LISA 2003)*, October 2003.

[46] Constantine Sapuntzakis and Monica S. Lam. Virtual appliances in the collective: a road to hassle-free computing. In *HOTOS'03: Proceedings of the 9th conference on Hot Topics in Operating Systems*, pages 10–10, Berkeley, CA, USA, 2003. USENIX Association.

[47] Evan Sarmiento. Securing FreeBSD using Jail. *Sys Admin: The Journal for UNIX Systems Administrators*, 10(5):31, 32, 34, 36–37, May 2001.

[48] A. Shoykhet, J. Lange, and P. Dinda. Virtuoso: A system for virtual machine marketplaces, August 08 2004.

[49] James E. Smith and Ravi Nair. The architecture of virtual machines. *IEEE Computer*, 38(5):32–38, 2005.

[50] Stephen Soltesz, Herbert Poetzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. In *Proceedings of the 2nd EuroSys Conference*, Lisboa, Portugal, March 2007.

[51] SWsoft. Virtuozzo server virtualization, 2007. `http://www.swsoft.com/en/products/virtuozzo`.

[52] Linux-VServer Team. Linux virtual server, 2007. `http://linux-vserver.org/`.

[53] U3 LLC. U3, 2008. `http://www.u3.com/`.

[54] Geoffroy Vallee, Thomas Naughton, and Stephen L. Scott. System management software for virtual environments. In *CF '07: Proceedings of the 4th international conference on Computing frontiers*, pages 153–160, New York, NY, USA, 2007. ACM Press.

[55] Leendert van Doorn. Hardware virtualization trends. In ACM, editor, *VEE 2006: proceedings of the Second International Conference on Virtual Execution Environments, June 14-16, 2006, Ottawa, Ontario, Canada*, pages 45–45, pub-ACM:adr, 2006. ACM Press.

[56] Steven J. Vaughan-Nichols. New approach to virtualization is a lightweight. *IEEE Computer*, 39(11):12–14, 2006.

[57] Virtual Appliances. Virtual appliances, 2007. `http://www.virtualappliances.net/`.

[58] VMware, Inc. Deliver desktops from the data center, 2008. http://www.vmware.com/products/vdi/.

[59] VMware, Inc. Virtual appliance marketplace, 2008. http://www.vmware.com/appliances/.

[60] VMware, Inc. Vmware virtual desktop infrastructure, 2008. http://www.vmware.com/files/pdf/vdi_datasheet.pdf.

[61] Xiaoying Wang, Zhihui Du, Yinong Chen, Sanli Li, Dongjun Lan, Gang Wang, and Ying Chen. An autonomic provisioning framework for outsourcing data center based on virtual appliances. *Cluster Computing*, 11(3):229–245, 2008.

[62] Xiaoying Wang, Zhihui Du, Sanli Li, and Yinong Chen. Modeling and simulation of virtualized autonomic service centers. *SIMULATION*, 84(4):119–136, 2008. http://sim.sagepub.com/cgi/content/abstract/84/4/119.

[63] XiaoYing Wang, DongJun Lan, Xing Fang, Meng Ye, and Ying Che. A resource management framework for multi-tier service delivery in autonomic virtualized environments. *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pages 310–316, April 2008.

[64] XiaoYing Wang, DongJun Lan, Gang Wang, Xing Fang, Meng Ye, Ying Chen, and QingBo Wang. Appliance-based autonomic provisioning framework for virtualized outsourcing data center. *Autonomic Computing, 2007. ICAC '07. Fourth International Conference on*, pages 29–29, June 2007.

[65] Robert N. M. Watson. Jails: Confining the omnipotent root, 2000. phk.freebsd.dk/pubs/sane2000-jail.pdf.

[66] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. Denali: A scalable isolation kernel, July 10 2002.

[67] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. Scale and performance in the Denali isolation kernel. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation*, December 2002.

[68] Wikipedia. Comparison of virtual machines, 2007. http://en.wikipedia.org/wiki/Comparison_of_virtual_machines.

[69] Wikipedia. Freevps, 2007. http://en.wikipedia.org/wiki/FreeVPS.

[70] Wikipedia. Kernel-based virtual machine, 2007. http://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine.

[71] Wikipedia. Parallels workstation, 2007. http://en.wikipedia.org/wiki/Parallels_Workstation.

[72] Wikipedia, the free encyclopedia. Mojopac, 2008. http://en.wikipedia.org/wiki/Mojopac.

[73] D.I. Wolinsky and R.J. Figueiredo. Simplifying resource sharing in voluntary grid computing with the grid appliance. *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8, April 2008.

[74] Carl J. Young. Extended architecture and hypervisor performance. In *Proceedings of the workshop on virtual computer systems*, pages 177–183, New York, NY, USA, 1973. ACM Press.

[75] Haifeng Yu and Amin Vahdat. Minimal replication cost for availability. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 98–107, New York, NY, USA, 2002. ACM.

[76] Yang Yu, Fanglu Guo, Susanta Nanda, Lap-Chung Lam, and Tzi cker Chiueh. A feather-weight virtual machine for windows applications. In Hans-Juergen Boehm and David Grove, editors, *Proceedings of the 2nd International Conference on Virtual Execution Environments, VEE 2006, Ottawa, Ontario, Canada, June 14-16, 2006*, pages 24–34. ACM, 2006.

[77] Yuedong Zhang, Zhenhua Song, Dingju Zhu, Zhuan Chen, and Yuzhong Sun. Redar: A remote desktop architecture for the distributed virtual personal computing. *Grid and Cooperative Computing, 2006. GCC 2006. Fifth International Conference*, pages 1–8, Oct. 2006.

APÊNDICE A - EXEMPLOS DE CONFIGURAÇÃO

Este apêndice apresenta alguns exemplos de configuração do sistema. O objetivo é mostrar como se dá na prática a descrição de *virtual appliances*, grupos, entre outras coisas.

6.1 *Virtual Appliances*

6.1.1 Informações Básicas

Há dois tipos de informações básicas. O primeiro consiste em um conjunto de dados genéricos, independentes do tipo de *MMV*. A seguir é apresentada a relação de dados gerais em formato *XML*.

Como pode ser observado na figura 6.1, os dados genéricos incluem o número de *CPUs* virtuais (`vcpus`), a quantidade recomendada de memória (`mem`), o número de interfaces virtuais de rede (`nics`), o tamanho do disco virtual (`disk-size` - em *MB*), o nome do arquivo do disco virtual (`disk-file`) e o tipo de *MAC*, sendo fixo ou não (`fixed-mac`). Esses dados podem ser utilizados para preparar, configurar e ativar os *VAPs* na maioria dos *MMV*, pois constituem informações gerais, comuns aos *MMVs*.

O segundo conjunto de informações, representado na figura 6.2, é mais específico, relativa à imagem do *VAP* (`vap-image`), como ilustrado no *XML* apresentado a seguir.

```
<vcpus> N </vcpus>
<mem> N </mem>
<nics> N </nics>
<disk-size> N </disk-size>
<disk-file> file name </disk-file>
<fixed-mac> yes or no </fixed-mac>
```

Figura 6.1: Informações básicas de *hardware* de um *VAP*

```
<vap-image>
  <name>apache2</name>
  <description>Apache 2, MySQL Server, PHP</description>
  <version>1.0</version>
  <url>http://repository.vap.intranet/VAPs/public/apache2.zip</url>
  <cacheable>yes</cacheable>
  <size>200</size>
  <timestamp>20090128115100</timestamp>
  <md5sum>c9443ab54420621173c7d8074c076ea0</md5sum>
  <sha1sum>24d28ff5c1ac12e754e8896295dd5c43d038bdf1</sha1sum>
  <config-file>apache2.vmx</config-file>
</vap-image>
```

Figura 6.2: Informações básicas de descrição de um *VAP*

Nesse grupo estão inclusos dados como o nome do *VAP* (`name`), uma descrição do conteúdo do seu conteúdo `description`), um identificador de versão (`version`), um identificar único do tipo *URI* (`url`), o tamanho do arquivo/pacote (`size`), um carimbo de tempo (`timestamp`), os valores *MD5* (`md5sum`) e *SHA1* (`sha1sum`) e a definição de um arquivo de configuração (`config-file`). Dados como `timestamp`, `md5sum` e `sha1sum` servem para o controle de versão e integridade do pacote do *VAP*.

A `url` identifica o endereço para o *download* do *VAP*. Já parâmetros como `cacheable` determinam se o pacote do *VAP* pode ou não ser mantido nas *caches* locais dos sistemas hospedeiros.

6.1.2 Dependências

A figura 6.3 ilustra a configuração das dependências de um *VAP*. No caso, há três dependências registradas no arquivo de configuração do *appliance*.

Como pode ser observado no exemplo da figura 6.3, há informações como ordem (`order`), nome (`name`) e *URL* (`url`). A ordem define a seqüência lógica de ativação dos *VAPs*. O nome e a *URL* são informações básicas para buscar as informações e os dados dos *VAPs* que serão ativados.

O exemplo apresentado, para a criação de um ambiente de desenvolvimento de sistemas, ilustra a aplicação de dependências. Outros exemplos poderiam ser utilizados, indo desde áreas como administração de redes e laboratórios de informática até a criação e manutenção de grandes e complexos ambientes virtuais de pesquisa, como grades computacionais.

```
<vap-dependencies>
  <vap-dependency>
    <order>1</order>
    <name>Java Application Server (JAS)</name>
    <url>http://repository.vap.intranet/VAPs/development/JAS.xml</url>
  </vap-dependency>
  <vap-dependency>
    <order>2</order>
    <name>Database System</name>
    <url>http://repository.vap.intranet/VAPs/development/DBS.xml</url>
  </vap-dependency>
  <vap-dependency>
    <order>3</order>
    <name>Control Versioning System (CVS)</name>
    <url>http://repository.vap.intranet/VAPs/development/CVS.xml</url>
  </vap-dependency>
</vap-dependencies>
```

Figura 6.3: Dependências de um *VAP*

```
<cache>
  <cachedir>/var/cache/flexVAPs</cachedir>
  <cachesize>10000</cachesize>
  <max-vaps-rm-per-clean>5</max-vaps-rm-per-clean>
  <daemon>
    <host>localhost</host>
    <port>7070</port>
    <protocol>tcp</protocol>
  </daemon>
</cache>
```

Figura 6.4: Configuração básica do componente de *cache*

6.1.3 Configuração Básica do Componente de *Cache*

A configuração do componente de *cache* passa pela definição do diretório de *cache* (`cachedir`), do tamanho máximo de armazenamento de *VAPs* (`cachesize`), o número máximo de *VAPs* que podem ser removidos em cada "limpeza de *cache* "[1].

As configurações de *cache* incluem também os dados de conexão do componente de *cache*, também denominado de *daemon* (`daemon`). Por padrão, o componente irá abrir uma porta TCP/IP, aguardando por conexões dos clientes.

[1]Limpeza de *cache* consiste no processo de remover os *VAPs* mais antigos, liberando espaço para o armazenamento local de novos *VAPs*.

```
<vmm-installer>
  <allow-install>yes</allow-install>
  <daemon>
    <host>localhost</host>
    <port>7071</port>
    <protocol>tcp</protocol>
  </daemon>
</vmm-installer>
```

Figura 6.5: Configuração básica do componente de instalação de *VMMs*

```
<vap-config>
  <vcpus>1</vcpus>
  <mem>512</mem>
  <nics>1</nics>
  <disk-size>4096</disk-size>
  <disk-file>devVAP.vmdk</disk-file>
  <fixed-mac>no</fixed-mac>
  ... (VAP image, VMMs, dependencies, ...)
</vap-config>
```

Figura 6.6: *VAP* basic config example

6.1.4 Configuração Básica do Componente de *MMV*

O componente de instalação é similar ao componente de *cache*, porém, com funções diferentes. A configuração do *daemon* de instalação de *MMVs*, no sistema hospedeiro, é simples, como ilustrado a seguir. Em primeiro lugar, é definido se o sistema local permite ou não instalação de novos *MMVs*. Ademais, são definidos os dados básicos de conexão do componente de instalação, que pode também ser ativado remotamente.

6.2 Exemplos de Configurações de Uso do Sistema

Esta seção apresenta exemplos de configuração dos principais componentes e pacotes do sistema. As configurações representam dados reais, aplicáveis ao sistema.

6.2.1 *Virtual Appliance*

As figuras 6.6, 6.7, 6.9, e 6.11 formam um exemplo de configuração de um pacote de dados, um *VAP*. Como pode ser observado, há configurações básicas/genéricas, específicas, de *MMVs* que são capazes de suportar a instanciação do *VAP* e de dependências em relação a outros *VAPs*.

As configurações básicas incluem o número de *CPUs* virtuais (1), a quantidade mí-

```
<vap-image>
  <name>Development VAP</name>
  <description>
   GNU/Linux with Java development environment.
   Including Eclipse, SUN JDK 1.6 and gcj 4.4.
  </description>
  <version>1.0</version>
  <url>
   http://repository.vap.intranet/flex/VAPs/devs/devVAP.tgz
  </url>
  <cacheable>yes</cacheable>
  <size>2000</size>
  <timestamp>20090122111300</timestamp>
  <md5sum>b6d81b360a5672d80c27430f39153e2c</md5sum>
  <sha1sum>3b71f43ff30f4b15b5cd85dd9e95ebc7e84eb5a3</sha1sum>
  <compressed>yes</compressed>
  <config-file>
   http://config.vap.intranet/flex/VAPs/devs/devVAP.cfg
  </config-file>
</vap-image>
```

Figura 6.7: Exemplo de configuração de uma imagem de um *VAP*

nima de memória (512 *MB*), o número de interfaces de redes virtuais (1), o tamanho (4 *GB*) e o nome do arquivo de dados. Essas informações são gerais, aplicáveis a maioria dos *MMVs*.

As figuras 6.7 e 6.8 apresentam a configuração da imagem (pacote de dados) do *VAP*. A figura 6.7 apresenta os detalhes do pacote. Como pode ser observado, é um sistema com *GNU/Linux* e ferramentas para o desenvolvimento de aplicações `Java`. O pacote tem tamanho de 2 *GB* e está compactado (`compressed = yes`). Além disso, ele possui um arquivo de configuração, para inicialização do *VAP*, denominado `devVAP.cfg`.

Na figura 6.8 são apresentadas configurações opcionais de *scripts*. Os quatro tipos de *scripts* atualmente suportados são: primeiro e qualquer *boot*[2], configuração (`vap--setup-script`) e inicialização (`vap-startup-script`) do *VAP*. O *script* de configuração do *VAP* tem como principal finalidade realizar ajustes no sistema local e ou nas configurações do *MMV* antes de criar as instâncias da *MV* do *VAP*. Já o *script* de inicialização tem como simples função inicializar/instanciar a *MV* do *VAP*.

[2]Ambos os *scripts* são configurados no sistema do *VAP*.

```
<optional-scripts>
  <firsttime-boot-script>
   http://scripts.vap.intranet/scripts/labs/labX.fbs
  </firsttime-boot-script>
  <anytime-boot-script>
   http://scripts.vap.intranet/scripts/labs/labX.abs
  </anytime-boot-script>
  <vap-startup-script>
   http://scripts.vap.intranet/flex/VAPs/devs/devVAP.vss
  </vap-startup-script>
  <vap-setup-script>
   http://scripts.vap.intranet/flex/VAPs/scripts/devVAP.sh
  </vap-setup-script>
</optional-scripts>
```

Figura 6.8: Exemplo de configuração dos *scripts* opcionais do *VAP*

6.2.2 *MMV*

Um *VAP* suporta a configuração de *MMVs*. A figura 6.9 apresenta um exemplo de configuração de *MMVs* para o *VAP* cuja descrição é apresentada na seção 6.2.1. A idéia é que esses *MMVs*, quando ausentes no sistema hospedeiro, possam ser instalados de forma automática e transparente.

As informações básicas, essenciais, do *MMV* incluem a *URL* (`url`) com a descrição do pacote de dados a ser instalado no sistema local, o endereço do arquivo de configuração para inicialização do *VAP* com o respectivo *MMV* (`vap-basic-config`) e o *script* de instanciação do *VAP* no sistema hospedeiro com o novo *MMV* (`vmm-vap-startup--script`).

A figura 6.10 apresenta um exemplo de configuração de um pacote de dados para instalação de um novo *MMV* no sistema hospedeiro. As duas principais informações correspondem a *URL* (`url`) do pacote de dados a ser instalado e o *script* de instalação do pacote. O componente de instalação de *MMVs* no sistema hospedeiro utiliza essas informações e dados para proceder a instalação do novo *MMV*.

6.3 Dependências

A figura 6.11 apresenta as dependências do *VAP* descrito na seção 6.2.1. As dependências podem ser recomendadas ou obrigatórias. No exemplo em questão, as dependências são apenas recomendadas (`strictly-needed` igual a no). Neste caso, a ativação das dependências é opcional. Caso contrário, a ativação da *VAP* dependeria da ativação das

```
<vmms>
  <vmm>
    <name>vmware</name>
    <version>1.2</version>
    <url>
     http://vmms.vap.intranet/flex/VMMs/files/vmware1.2.xml
    </url>
    <vap-basic-config>
     http://config.vap.intranet/flex/VMMs/devs/devVAP.vmx
    </vap-basic-config>
    <vmm-vap-startup-script>
     http://scripts.vap.intranet/flex/VMMs/scripts/devVAP-vmware.sh
    </vmm-vap-startup-script>
  </vmm>
  <vmm>
    <name>fluxbox</name>
    <version>1.1</version>
    <url>
     http://vmms.vap.intranet/VAPs/VMMs/fluxbox1.1.xml
    </url>
    <vap-basic-config>
     http://config.vap.intranet/flex/VMMs/devs/devVAP.flx
    </vap-basic-config>
    <vmm-vap-startup-script>
     http://scripts.vap.intranet/flex/VMMs/scripts/devVAP-fluxbox.sh
    </vmm-vap-startup-script>
  </vmm>
</vmms>
```

Figura 6.9: Exemplo de descrição de um *VMMs*

```
<vmm>
  <name>vmware</name>
  <version>1.2</version>
  <description>Vmware Player</description>
  <host-os>GNU/Linux</host-os>
  <vmm-image>
    <url>
     http://vmms.vap.intranet/flex/VMMs/files/vmware1.2.tgz
    </url>
    <size>12</size>
    <md5sum>4dde7cfbdb096c7eced160658b7117</md5sum>
    <sha1sum>afb3246750e909dd4fd2023e33ac5e</sha1sum>
  </vmm-image>
  <install-script>
   http://vmms.vap.intranet/flex/VMMs/scripts/vmware1.2.sh
  </install-script>
</vmm>
```

Figura 6.10: Exemplo de descrição do Vmware 1.2 *XML* (`vmware1.2.xml`)

```
<dependencies>
  <vap>
    <order>1</order>
    <name>MySQL</name>
    <url>
     http://repository.vap.intranet/flex/VAPs/public/MySQL.xml
    </url>
    <strictly-needed>no</strictly-needed>
  </vap>
  <vap>
    <order>2</order>
    <name>JAS</name>
    <url>
     http://repository.vap.intranet/flex/VAPs/public/JAS.xml
    </url>
    <strictly-needed>no</strictly-needed>
  </vap>
</dependencies>
```

Figura 6.11: Exemplo de descrição de dependências

suas respectivas dependências.

As dependências são úteis em diferentes contextos. Um deles é o exemplo apresentado, pois normalmente utiliza-se um banco de dados e um servidor de aplicação para o desenvolvimento de *software* `Java`, como é o caso. O servidor de banco de dados serve para organizar e armazenar os dados, enquanto que o servidor de aplicação serve para testar as soluções desenvolvidas.

6.4 Comandos

Os comandos são úteis e necessários em diferentes contextos. Um deles é o caso da descompressão, necessária a alguns *VAPs*. A figura 6.12 ilustra a configuração de comandos no *host* hospedeiro. Essa configuração, por padrão disponível no arquivo `/etc/flexVAPs/commands.xml`, é particular a cada sistema hospedeiro. Essa característica oferece flexibilidade adicional ao sistema, utilizando recursos existentes em sistemas heterogêneos sem a necessidade de modificações no sistema em si.

Cada comando, como ilustrado na figura 6.12, possui um nome, um caminho do executável (arquivo necessário para a execução do comando) e os parâmetros do comando. Um comando de descompressão, como o `unrar`, precisa de alguns parâmetros de execução. No caso do `unrar`, o parâmetro obrigatório é o `x` e o parâmetro opcional é o `-o+`.

```
<sys-commands>
  <command>
    <name>wget</name>
    <path>/usr/bin/wget</path>
    <parameters>-a /tmp/flexVAPs.log --tries=10</parameters>
  </command>
  <command>
    <name>zsync</name>
    <path>/usr/bin/zsync</path>
    <parameters>-s</parameters>
  </command>
  <command>
    <name>rsync</name>
    <path>/usr/bin/rsync</path>
    <parameters>-q</parameters>
  </command>
  <command>
    <name>gunzip</name>
    <path>/usr/bin/gunzip</path>
    <parameters>-fq</parameters>
  </command>
  <command>
    <name>unrar</name>
    <path>/usr/local/uncompress/bin/unrar</path>
    <parameters>x -o+</parameters>
  </command>
</sys-commands>
```

Figura 6.12: Exemplo de descrição de comandos

```
<compress-types>
  <type>
    <name>gz</name>
    <compress>gzip</compress>
    <compress-pars>-f</compress-pars>
    <uncompress>gunzip</uncompress>
    <uncompress-pars>-fq</uncompress-pars>
  </type>
  <type>
    <name>zip</name>
    <compress>zip</compress>
    <compress-pars>-r</compress-pars>
    <uncompress>unzip</uncompress>
    <uncompress-pars>-u</uncompress-pars>
  </type>
  <type>
    <name>tgz</name>
    <compress>tar</compress>
    <compress-pars>czf</compress-pars>
    <uncompress>tar</uncompress>
    <uncompress-pars>xzf</uncompress-pars>
  </type>
  <type>
    <name>rar</name>
    <compress>rar</compress>
    <compress-pars></compress-pars>
    <uncompress>unrar</uncompress>
    <uncompress-pars>x</uncompress-pars>
  </type>
</compress-types>
```

Figura 6.13: Exemplo de descrição de compactadores e descompactadores de dados

Descompressão de dados.

A figura 6.13 apresenta um exemplo de tipos de compressão de dados. Essa descrição é mantida no repositório global e utilizada toda vez que há um *VAP* comprimido que precisa ser descomprimido. Porém, os comandos e parâmetros são substituídos pelos respectivos comandos locais, quando configurados no sistema hospedeiro, conforme ilustrado na figura 6.12.

Essa informação, de tipos de compressão de dados, é mais um nível de flexibilidade da arquitetura do sistema. Novos tipos de compressão podem ser incluídos, de forma simples e prática, a qualquer momento, sem necessidade de modificação alguma do sistema. Isso permite trabalhar-se com diferentes sistemas operacionais, contextos e tipos de dados.

```
#!/bin/sh
[ $1 ] || { echo "Usage: $0 file|dir"; exit; }
FILE=$1
[ -f $FILE ] || [ FILE=`find $1 -name \*.vmx`; }
vmplayer $FILE
```

Figura 6.14: Exemplo simples de um *script* (`devVAP.vss`)

```
#!/bin/sh
comando 1
comando 2
...
comando n
```

Figura 6.15: Ilustração de um *script* de configuração (`devVAP.sh`)

6.5 *Scripts*

Os *scripts*, conforme apresentado na seção 6.2.1 e na figura 6.8, possuem funções diversas. A principal vantagem dessa abordagem é a flexibilidade, pois um *script* pode teoricamente conter qualquer código, capaz de resolver os mais diversos tipos de problemas.

Um *script* pode ser algo bastante simples, como ilustrado na figura 6.14, que serve apenas para inicializar um *VAP*. Mas, um *script* pode também ser algo mais complexo, como ilustrado na figura 6.15, utilizado para preparar o ambiente do *MMV* para receber o novo *VAP*.

No caso de *scripts* de inicialização de *VAP*, figura 6.14, existe a necessidade de definir-se alguns detalhes básicos para fins de padronização e uniformidade. A linguagem utilizada é livre, podendo ser `Bash` *Scripting*, `Perl`, `Python`, entre outras. Mas, parâmetros como o arquivo de configuração, ou diretório de dados, do *VAP* devem tratados pelo *script*, que é o que acontece na figura 6.14, onde o primeiro parâmetro de entrada é analisado.

Um *script* pode ter de *uma* a *N* linhas de código. Tudo depende da aplicação e complexidade necessárias. Cabe ao usuário, desenvolvedor ou administrador do sistema codificar e, consequentemente, definir as funcionalidades e a complexidade do *script*.

```
<vap-defaults>
  <user>anonymous</user>
  <password>anonymous</password>
  <repository>
   http://repository.vap.intranet/flex/VAPs
  </repository>
  <list-of-dirs>init/list-of-directories.xml</list-of-dirs>
  <cacheable>yes</cacheable>
  <networking>yes</networking>
  <max-dependency-level>1</max-dependency-level>
  <global-config-url>
   http://config.vap.intranet/flex/VAPs/config/global.xml
  </global-config-url>
  <global-config-allowed>yes</global-config-allowed>
</vap-defaults>

<logs>
  <info>yes</info>
  <warn>yes</warn>
  <debug>yes</debug>
  <nullfile>/dev/null</nullfile>
  <logfile>/var/log/flexVAPs.log</logfile>
</logs>

<temporary-data>
  <tmp-dir>/tmp</tmp-dir>
  <tmp-file-template>tmp-XXXXXXX</tmp-file-template>
</temporary-data>

<host-online-config>
 http://config.vap.intranet/flex/VAPs/config/family-X.xml
</host-online-config>
```

Figura 6.16: Exemplo de configuração complementar para um *host* hospedeiro

6.6 Configuração do Sistema Hospedeiro

No sistema hospedeiro há um conjunto de informações de configuração necessárias aos componentes de *cache* (seção 4.2.3), aos componentes de instalação de *MMV* (seção 4.2.4), às interfaces em nível de usuário (seção 4.2.5) e dados gerais aos sistemas locais. A figura 6.16 ilustra configurações complementares também presentes no sistema local (`/etc/flexVAPs`). Elas são utilizadas pelos componentes do sistema para os mais diversos fins.

Como pode ser observado na figura 6.16, há quatro conjunto de dados definidos. Cada conjunto possui um objetivo finalístico distinto.

Configurações padrão dos *VAPs* : especificam dados básicos como *URL* padrão do repositório global e *URL* padrão das configurações globais. Parâmetros como usuário

(`user`) e senha (`password`) definem os valores padrões que serão utilizados pelo *daemons* e aplicativos nos sistemas hospedeiros.

Sistema de *logs* : os parâmetros de configuração dos *logs* tem diversas finalidades. O arquivo de *mylog* (`logfile`) especifica o arquivo de registro geral das atividades dos componentes ativos no sistema hospedeiro. O arquivo nulo (`nullfile`) especifica o arquivo a ser utilizado para saídas de dados que não serão registradas em local algum. Já os parâmetros de nível de *mylog*, que podem ser informacionais (`info`), de alerta (`warn`), ou de depuração (`debug`), definem a quantidade e o tipo de informações que os componentes do sistema irão gerar durante a execução.

Armazenamento temporário de dados : os parâmetros de diretório (`tmp-dir`) e modelo de nome para arquivos temporários (`tmp-file-template`) são utilizados para o armazenamento e ou criação de arquivos temporários durante a execução dos componentes do sistema. A exemplo, um arquivo *XML* do repositório global, como o das configurações globais (ver seção 6.7) é descarregado para o sistema de arquivos local como um arquivo temporário, que será utilizado apenas para atender a requisição atual.

Configurações *online* : as configurações *online*, definidas pelo parâmetro `host-online-config`, são úteis para dar maior flexibilidade e dinamicidade ao sistema. Em cada *boot*, ou em períodos agendados gerenciador de tarefas, o sistema hospedeiro checa as configurações *online*, atualizando o arquivo de configuração local sempre que necessário. Essas configurações *online* são caracterizadas por famílias de *hosts* hospedeiros. Exemplo: um determinado laboratório de informática, ou conjunto qualquer de máquinas, pode ser caracterizado como "*família X*". Sendo assim, todas as máquinas desse conjunto possuem um arquivo de configuração, *online*, em comum, ou seja, as atualizações de configuração podem ser realizadas simultaneamente em grupos de máquinas, de maneira simples e prática.

Atualização da Configuração. A atualização da configuração do sistema hospedeiro, como descrito resumidamente na seção 6.6, pode ser realizada de forma automática e *online*. O repositório global contém um arquivo *XML* semelhante ao arquivo de configuração do sistema local. No exemplo da seção 6.6 esse arquivo está disponível na *URL* `http://config.vap.intranet/flex/VAPs/config/family-X.xml`.

```
<global-config>
  <vap-repositories>
    <repository>
      <name>Rep1</name>
      <url>http://vapstore.vap.intranet/flex/VAPs</url>
      <list-of-dirs>init/list-of-directories.xml</list-of-dirs>
      <active>yes</active>
    </repository>
    <repository>
      <name>Rep2</name>
      <url>http://getvaps.vap.intranet/flex/VAPs</url>
      <list-of-dirs>init/list-of-directories.xml</list-of-dirs>
      <active>no</active>
    </repository>
    <repository>
      <name>Rep3</name>
      <url>http://public.vap.intranet/flex/VAPs</url>
      <list-of-dirs>init/list-of-directories.xml</list-of-dirs>
      <active>no</active>
    </repository>
  </vap-repositories>
  <compress-types-url>
   http://config.vap.intranet/flex/config/compress-types.xml
  </compress-types-url>
</global-config>
```

Figura 6.17: Exemplo de configuração global

O processo de atualização das configurações do sistema hospedeiro pode ocorrer basicamente em dois momento, na hora da inicialização do sistema (*boot*) ou em período programados no agendador de tarefas do sistema hospedeiro. No primeiro caso o processo de atualização é simples, sendo o suficiente apenas copiar o arquivo de configuração *online* para o sistema local. Já no segundo caso é necessário re-iniciar os serviços de manutenção da *cache* e instalação de *MMVs*, para que estes leiam as novas configurações do sistema local.

6.7 Configuração Global

As configurações globais, que podem ou não serem utilizadas pelos componentes do sistema de acordo com as configurações locais, são particularmente úteis quando há vários servidores de dados agregados no repositório global. Neste caso, a configuração global, como ilustrado na figura 6.17, irá indicar o endereço dos servidores de pacotes de dados. Quando utilizadas, essas configurações fazem com que as ferramentas em nível de usuário busquem informações e dados nos diferentes endereços, servidores.

As configurações globais incluem também informações como o endereço do arquivo *online* de descrição dos tipos de compactação de dados utilizados no sistema, conforme discriminado na seção 6.4. Essa é mais uma característica que oferece flexibilidade e dinamicidade ao sistema, pois o arquivo de descrição de tipos de compressão de dados pode ser rápida e facilmente atualizado e propagada para todos os componentes do sistema localizados nos *hosts* hospedeiros.

6.8 Grupos de *VAPs*

Os grupos de *VAPs*, como proposto na arquitetura do sistema (ver seção 3.3.2), é uma forma de agregar *VAPs* que irão formar um ambiente para um determinado fim. Diferentemente de dependências, os grupos de *VAPs* podem ser arranjados das mais diversas formas, sem ordem de precedência ou dependência estrita em eles.

A figura 6.18 apresenta a configuração simples de um grupo de *VAPs*. A definição do grupo é simples, incluindo apenas dados como o nome (`name`), a *URL* (`url`) e o número de instâncias do *VAP* (`instances`). No caso do *nodeX*, está definido que serão alocadas 10 instâncias do *nodeX.xml*. Tanto as instâncias do *serverX*, quanto as do *nodeX* serão criadas nos sistemas hospedeiros do grupo de *hosts clusterX*, cuja descrição é apresentada na figura 6.19.

Como pode ser observado, o grupo de *hosts* (`host-group`) é formado por quatro unidades. O primeiro *host*, *server1*, é do tipo servidor (`<type> server </type>`) e aceita apenas uma instância de um *VAP* qualquer. O segundo e o terceiro *host*, *node1* e *node2*, são do tipo cliente (`<type> client </type>`) e comportam até duas instâncias de um *VAP* qualquer. Já o quarto *host*, *nodeX*, é do tipo genérico (`<type> any </type>`), ou seja, pode ser tanto servidor quanto cliente, e comporta até quatro instâncias de *VAPs* quaisquer.

Com os recursos de agrupamentos de *VAPs* e associação de grupos de *hosts* é possível criar-se ambientes diversos, para os mais diversos fins. Um exemplo é um *cluster* dinâmico, utilizado por diferentes tipos de usuários com diferentes demandas de sistemas. Neste contexto, cada usuário pode instanciar os nós do *cluster* de forma dinâmica e automática, instanciando o seu conjunto de *VAPs* sempre que necessário.

```
<vap-group>
  <vap>
    <name>serverX</name>
    <url>
     http://groups.vap.intranet/flex/VAPs/servers/serverX.xml
    </url>
    <instances>1</instances>
  </vap>
  <vap>
    <name>nodeX</name>
    <url>
     http://groups.vap.intranet/flex/VAPs/clients/nodeX.xml
    </url>
    <instances>4</instances>
  </vap>
  <hosts>
    <name>clusterX</name>
    <url>
     http://config.vap.intranet/flex/config/hosts/clusterX.xml
    </url>
  </hosts>
  <daemon-connection>
    <port>7771</port>
    <protocol>tcp</protocol>
  </daemon-connection>
</vap-group>
```

Figura 6.18: Exemplo de descrição de um grupo de *VAPs*

```
<host-group name=cluster1>
  <host>
    <name>server1</name>
    <ip>10.0.0.1</ip>
    <max-instances>1</max-instances>
    <type>server</type>
  </host>
  <host>
    <name>node1</name>
    <ip>10.0.0.11</ip>
    <max-instances>2</max-instances>
    <type>client</type>
  </host>
  <host>
    <name>node2</name>
    <ip>10.0.0.12</ip>
    <max-instances>2</max-instances>
    <type>client</type>
  </host>
  <host>
    <name>nodeX</name>
    <ip>10.0.0.100</ip>
    <max-instances>4</max-instances>
    <type>any</type>
  </host>
</host-group>
```

Figura 6.19: Exemplo de descrição de um grupo de *hosts*

APÊNDICE B - PUBLICAÇÕES GERADAS

Durante a realização deste trabalho, foi desenvolvido e publicado um artigo internacional. O artigo descreve e avalia a proposta do sistema. O mini-curso e o trabalho individual apresentam conceitos, aplicações, mercado e prática de virtualização, incluindo *virtual appliances*.

Artigo: *FlexVAPs: a system for managing virtual appliances for heterogeneous virtualized environments*, aprovado, apresentado e publicado nos anais *LANOMS 2009* (`http://www.lanoms.org/2009/`).

Mini-curso: *Virtualização: Conceitos, Aplicações, Mercado e Prática*, aprovado, apresentado e publicado nos anais *ERRC 2009* (`http://www.setrem.com.br/faculdade/tr/7errc/data.html`).

Trabalho individual: *Conceitos e Aplicações dos Monitores de Máquinas Virtuais*, aprovado e apresentado ao *PPGI/UFSM* em *2007*. (`http://www.inf.ufsm.br/index/pos-graduacao`)