Automatização da administração de sistemas Unix

Carlos Alberto da Silva

Tese de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Automatização da administração de sistemas Unix

Carlos Alberto da Silva

Dezembro de 2001.

Banca Examinadora:

- Prof. Dr. Paulo Lício de Geus (Orientador)
- Prof. Dr. Fabio Queda Bueno da Silva
  Depto. de Informática, Universidade Federal de Pernambuco
- Prof. Dr. Célio Cardoso Guimarães
  Instituto de Computação, Unicamp

Automatização da administração de sistemas Unix

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Carlos Alberto da Silva e aprovada pela Banca Examinadora.

Campinas, 10 de Dezembro de 2001.

Prof. Dr. Paulo Lício de Geus
(Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Silva, Carlos Alberto da
Si38a Automatização da administração de sistema UNIX / Carlos Alberto da Silva -- Campinas, [S.P. :s.n.], 2001.

Orientador : Paulo Lício de Geus
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. UNIX (Sistema operacional de computador). 2. Redes de computação – Administração. 3. Software. I. Geus, Paulo Lício de. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 10 de dezembro de 2001, pela Banca Examinadora composta pelos Professores Doutores:

Prof. Dr. Fábio Queda Bueno da Silva
UFPe

Prof. Dr. Edmundo Roberto Mauro Madeira
UNICAMP

Prof. Dr. Célio Cardoso Guimarães
IC – UNICAMP

Prof. Dr. Paulo Lício de Geus
IC – UNICAMP

A meus filhos, Felipe e Éloa, por todo amor e compreensão que sempre me deram.

Agradecimentos

Ao Prof. Dr. Paulo Lício de Geus, pela orientação, apoio, amizade e pela confiança em min depositada.

Aos vários professores do Instituto de Computação que colaboraram para minha formação. Um agradecimento especial para o Prof. Dr. Cid Carvalho e Profª. Dra. Claudia Bauzer pelo apoio recebido.

Aos funcionários do Instituto de Computação, por colaborarem para um ambiente de trabalho agradável. Um agradecimento em especial ao funcionário Luís (in memoriam), pelo apoio no inicio desta jornada.

A CPqD e FUFMS, pelo apoio financeiro.

À minha mãe, pelo carinho e dedicação constantes.

A meu pai, por ter me ensinado a vencer.

A meu filho Felipe, pelo poder de concentração.

À minha filha, pelo espirito de luta.

Aos meus amigos, por afastarem de min o fantasma da solidão.

Prefácio

A administração de redes de computadores vem se tornando, com o passar dos anos, uma tarefa complexa e lenta, devido ao aumento significativo das redes, ao número de softwares instalados nelas e ao seu grande número de usuários. Hoje, quando se fabrica um computador, o mesmo já prevê sua utilização em rede, assim como, a maioria dos softwares é desenvolvida para serem utilizados em ambiente de rede.

Nesta dissertação, a tarefa de administrar a manutenção de software em uma rede Unix é estudada em detalhes, tratando os softwares em duas categorias: o sistema operacional unix e os aplicativos. No assunto de manutenção de sistema operacional serão analisadas as formas de instalação, configuração, atualização e remoção (*patch*), estudando as várias metodologias de abordar tais assuntos. Em manutenção de aplicativos serão analisados os mesmos problemas de sistemas operacionais, além der ser apresentado métodos de resolver os problemas de: espaço de nomes (hierarquia de diretório) do sistema de arquivos, controle de versões de software, e o monitoramento do uso de software, inerentes a aplicativos.

Enquanto grande parte da literatura trata o assunto de administrar software com desdém, abrangendo apenas a tarefa de como configurar e manter os softwares funcionando, neste trabalho a preocupação é planejar as atividades de manutenção de software, e acompanhar todo o ciclo de vida de um software. Neste ciclo, distinguem-se 4 ações básicas: instalação, configuração, atualização e remoção. Serão apresentados alguns métodos que visam aumentar a produtividade dos administradores de redes na área de manutenção de softwares em ambiente Unix, com o enfoque de manter um, ou vários, ambientes computacionais produtivos, de fácil administração, e de forma planejada e documentada.

Lista de Figuras

Lista de tabelas

Conteúdo

Capítulo 1
Introdução

Atualmente, devido ao aumento no tamanho das redes de computadores, e consequentemente no número de estações de trabalho nelas, a tarefa de administrá-las tornou-se mais complexa e lenta, exigindo dos administradores de rede soluções em menor tempo e de melhor qualidade.

A necessidade de um usuário, quando começa a trabalhar em uma estação de trabalho é utilizar todos os recursos computacionais de forma transparente e objetiva, não querendo saber se um recurso está local ou remoto, ou se está configurado ou não para sua estação. Assim, os administradores de rede devem produzir ambientes de trabalhos estáveis, confiáveis e disponíveis, independentes de localização, distância e tempo.

O administrador de rede tem sobre sua responsabilidade direta a administração de três grupos distintos e inseparáveis, conforme especificação da [26], a saber: *software*, *hardware* e *peopleware*[1].

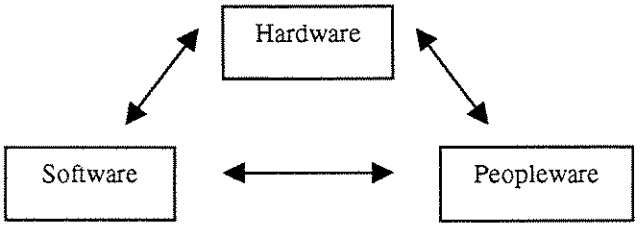

Figura 1.1: Grupos interagindo.

O grupo de *Hardware* é composto de todo o equipamento de rede, como estações, servidores, hubs, switches, roteadores, cabeamento, impressoras, scanners etc. O administrador de rede tem total domínio sobre as rotinas de manutenção, como instalar, configurar e remover um hardware; Tem o registro de todas as

[1] Em algumas literaturas podem ser chamadas de usuários ou grupos de usuários.

informações deste hardware, sobre sua funcionalidade e sua manutenção, e como são suas interações com os outros grupos.

No grupo de *Peopleware* têm-se todos os usuários ou grupos de usuários que utilizam os serviços disponíveis da rede, como *internet*, *e-mail*, *ftp*, impressão etc., para produzirem suas atividades diárias. O administrador de rede deve ter o registro de todos os usuários, a que grupo ou subgrupo pertence, assim como as suas necessidades e forma que interage com os demais grupos, orientado-o para utilizar a rede da melhor forma possível.

No grupo de *Software* têm-se todos os programas, sistemas e aplicativos que estão instalados e configurados para serem utilizados pelo grupo de *peopleware* através do grupo de *hardware*. O administrador deve registrar todas informações possíveis sobre um software específico, do tipo: quando foi instalado, quando foi configurado, quando foi atualizado, quem o utiliza, com que freqüência, quanto durou a sessão e outras informações.

Em resumo, na visão do usuário que interage diretamente com sua estação de trabalho, mas compreende o ambiente de rede ao qual sua estação está conectada e configurada, seu conhecimento se restringe em conhecer os serviços disponíveis para ele (usuário) utilizar.

Atualmente, o administrador de rede deve atender o grupo de *peopleware* primeiro para analisar suas necessidades e funcionalidade para determinar o grupo de *software* que melhor solucionará estas necessidades, e por conseqüência determinará o grupo de *hardware*. Por exemplo, uma empresa financeira deseja criar um departamento de cobrança, o administrador de rede analisa as necessidades do grupo *peopleware* que vai trabalhar neste departamento, especificando quais softwares serão necessários, e por conseqüência especificará quais estações de trabalho e servidores serão necessários para atender a demanda deste novo departamento.

Na visão do administrador de rede, ele é responsável pelas rotinas de controle, administração, e gerência dos três grupos, bem como suas interações. Como em qualquer tarefa de administração da rede, descritas em [19], o administrador da rede deve ser responsável por sua organização e documentação, registrando todas as rotinas executadas diariamente, tendo total controle sobre todos os procedimentos de manutenção de cada grupo, garantindo que estes procedimentos deverão ser iniciados, e que terminarão num prazo finito.

Devemos ressaltar que o administrador de rede não consegue quantificar o tempo a ser gasto em procedimento de manutenção de software devido a influencia direta de valores variáveis, do tipo: trafego na rede, tempo de leitura e escrita no disco rígido, velocidade do barramento da estação, e outras. Devido as estas variáveis, os tempos são estimados média aritmética.

Esta dissertação tem por objetivo descrever apenas o grupo de *Software*, propondo novas metodologias para as rotinas de manutenção de software, criadas e executadas pelos administradores de rede no ambiente Unix Solaris, da Sun Microsystems. Neste contexto, a palavra procedimento ou rotina é definido

como uma seqüência de passos (atos) para executar uma atividade de manutenção de software no ambiente da rede e executada pelo administrador da rede em tempo finito.

Dentro do grupo de *software*, os softwares que executam em uma estação de trabalho ou servidor, podem ser classificados em duas classes:

- ***Classe Sistema Operacional (OS)***: um conjunto de softwares (programas) que: controla todos os recursos computacionais da estação de trabalho (memória, disco, *floppy*, cd-rom etc.), coordena a execução de aplicativos nesta estação, gerencia a comunicação com a rede, e outras funcionalidades.
- ***Classe Aplicativos***: são pacotes de software que executam funções específicas para atender as necessidades do usuário, por exemplo: editor de texto, *browser* de Internet, programa de receber e enviar *e-mail*, compilador da linguagem C++ e outros programas/sistemas. Basicamente, pode-se defini-los com todos os softwares que um usuário final utiliza em seu dia a dia para trabalhar.

O administrador de rede deve ter total conhecimento sobre as rotinas de manutenção de software sobre estas duas classes, conhecendo sua organização, funcionalidade e manutenção, de forma independente e inter-relacionadas. Assim como, este grupo de *software* interage com os outros grupos (*hardware* e *peopleware*).

A tarefa de administrar a Classe sistema operacional Unix pode ser dividida nos seguintes procedimentos:

- **Instalação**: este procedimento pode ser visto como instalar o sistema operacional em uma estação ou servidor que não o tem, como também a sobreposição de um sistema operacional já instalado, mas ambos os procedimentos serão tratados como instalação. Deverá ser planejado o que será instalado (quais pacotes de software), como será instalado (via rede, cd-rom, ou disquete), de que forma será instalado (interativa ou automática), como será configurado para interagir com os demais softwares, como será configurado para interagir com o grupo de *hardware*, e como será divulgado/disponibilizado para o grupo *peopleware*.
- **Configuração**: o procedimento de configurar o sistema operacional será visto como alterar a configuração de um serviço (OS) para melhorar seu desempenho, ou mesmo para disponibilizar novos recursos ou novos serviços ao grupo de *peopleware*. Exemplos: configurar o sistema operacional Unix para trabalhar em rede com uma nova placa de rede da 3Com, como conseqüência, configurar o sistema operacional para disponibilizar e/ou utilizar os serviços de NFS [SUN92a], DNS e NIS [43] do ambiente disponível na rede local através deste novo recurso.
- **Atualização** este procedimento é definido como inserir ou substituir um programa ou conjunto de programas que executará ou corrigirá novas funcionalidades ao sistema operacional. Exemplo: aplicar o *patch* #10567890, no sentido de substituir ou mesmo inserir se não tiver, para atualizar os drivers de impressão da impressora HP 4MV.

O administrador de rede deve ter controle total sobre as rotinas que compõem cada grupo de procedimentos relacionados à manutenção de software na ***classe sistema operacional***, registrando quando tais rotinas foram criadas, para que grupo ou subgrupo de *hardware* servirá ou para que grupo de *peopleware* servirá, ou seja, sua funcionalidade definida dentro dos grupos, assim como: registrar quem criou tal rotina, quem solicitou tal rotina, quando foi executada, quem a executou, e se esta rotina possui outras versões para a mesma função.

O procedimento de remoção na classe de sistemas operacionais não ocorre como um todo, ou seja, o sistema operacional nunca é removido, mas apenas um serviço do sistema operacional é removido, por exemplo: remover o patch #4556787 de uma estação especifica que controla o serviço de NFS [29].

Temos a motivação de encontrar soluções, como em [27], que ao executar um script produziu a instalação do sistema operacional Unix em 1500 estações e servidores conectados a rede. Este resultado foi obtido devido a uma política de administração de rede bem definida, um banco de dados para registrar as informações de cada estação e aos scripts que automatizam estes procedimentos de manutenção de software.

Na tarefa de administrar a ***Classe de Aplicativos*** para o ambiente Unix serão descritos os seguintes procedimentos que são executados pelo administrador de rede:

- **Instalação**: o procedimento de instalar um aplicativo pode ser visto como disponibilizar o mesmo em uma estação de trabalho ou servidor que não o tem, como também a instalação de uma nova versão, mas mantendo uma versão anterior. Quando isto acontece o administrador não deve deixar que a nova versão sobreponha os arquivos da versão anterior, este problema é conhecido como ***problema de espaço de nomes*** e será melhor descrito no Capítulo 4. Exemplo: instalar o editor de texto `emacs` versão 20.34 e manter a versão 20.30.
- **Configuração**: este procedimento de configurar o aplicativo será tratado como alterar a configuração de um aplicativo para melhorar seu desempenho ou mesmo para corrigir a sua forma de operar. Exemplos: configurar o `emacs` versão 19.38 para ser executado do servidor `Parana`.
- **Atualização**: o procedimento de atualizar será visto como inserir ou substituir um programa ou conjunto de programas que garante uma nova funcionalidade ao aplicativo. Exemplo: atualizar o programa `emacs` da versão 20.34, inserindo novas fontes da letra `Courier`, ou substituindo as já existentes.
- **Remoção**: este procedimento de remover um aplicativo será tratado como a retirada de uma versão do aplicativo da estação de trabalho ou servidor, sem que afete os outros aplicativos instalados ou versões do mesmo aplicativo. Exemplo: remover o `emacs` versão 19.13, mantendo as versões 19.25 e 19.38 no mesmo ambiente. Este procedimento está relacionado ao problema de espaço de nomes e controle de versões, que serão descritos e tratados nos Capítulos 4.

O principal objetivo a ser alcançados nos procedimentos de manutenção do *grupo software*, para a classe *de sistema operacional* e de *aplicativos*, é sempre manter o ambiente de trabalho do *grupo peopleware* estável (não

ocorrerão erros na execução), ou seja, se ocorrer um procedimento de instalação, configuração ou atualização do sistema operacional todos os aplicativos que executavam antes deste procedimento devem manter a sua funcionalidade preservada.

Para a classe de aplicativos, se uma versão de aplicativo for instalada, configurada, atualizada ou removida, não ocorrera perda de funcionalidade para os demais aplicativos.

Devido as características destes procedimentos, a falta de padronização e a falta de documentação, muitas das soluções encontradas resolvem apenas um tipo de procedimento. Diante disto resolvemos encontrar soluções, que combinadas, atendam a todos os procedimentos de manutenção de software.

A presente dissertação tem por objetivo descrever os problemas relacionados à manutenção de software em ambiente Unix, para os procedimentos dentro das ***classes de sistemas operacionais e aplicativos***.

1.1 Organização da dissertação

Nesta seção, delineamos a estrutura da dissertação e apresentamos algumas decisões acerca da apresentação do texto.

Optamos por manter alguns termos e siglas em inglês. Priorizamos a uniformidade com a literatura da área em detrimento do uso exclusivo da língua portuguesa.

No Capítulo 2 relacionaremos as atividades que o administrador de rede deve desempenhar, assim como descreveremos os problemas e dificuldades encontradas na execução destas atividades. Será feita uma descrição de cada procedimentos de manutenção de software no ambiente Unix [28], ressaltando as diferenças entre as classes de sistemas operacionais e a classe de aplicativos.

Para cada procedimento de manutenção da classe de sistema operacional serão descritos os problemas relacionados, como a falta de padronização, de documentação e etc.

Abordaremos as duas formas de distribuição dos aplicativos, *script* e *package*., e consequentemente suas vantagens e desvantagens. Serão descritos os procedimentos de manutenção da classe de aplicativos e os problemas relacionados a estes procedimentos.

No Capítulo 3 é explicada especificamente a manutenção de software na classe de sistemas operacionais, será feita a descrição dos métodos: Tradicional, IC-Unicamp e Custom JumpStart, usados no ambiente Solaris. Na apresentação do método Tradicional, assim chamado por ser utilizado na maioria das empresas pesquisadas, descreveremos as suas particularidades, vantagens e desvantagens. O método IC-Unicamp para manutenção do sistema operacional Unix, que utiliza a ferramenta Jumpstart do ambiente

Solaris da Sun Microsystems de forma interativa e incompleta será descrito em detalhes. Descreveremos o método Custom Jumpstart, que utiliza a mesmas ferramentas do Solaris da Sun Microsystems, de forma automática e completa, a ser implantado no ambiente do IC-Unicamp, ressaltando as suas vantagens em relação ao atualmente usado.

O Capítulo 4 descreverá os procedimentos de manutenção da classe de aplicativos e as soluções encontradas para os Problemas de Espaço de Nomes, Controle de Versões e Monitoramento do uso de Software.

Para o Problema de Espaço de Nomes serão descritas as soluções encontradas: o método Tradicional que é usado na maioria das empresas pesquisadas, inclusive no IC-Unicamp, o método Andrew, usando no sistema operacional Andrew, que é compatível com todos os Unix disponíveis, e o método Stow da Gnu.

Para o Problema de Controle de Versões serão apresentados: o método Tradicional e o método Wrappers da Sun Microsystems.

Para o Problema de Monitoramento do uso de Software serão apresentados: o método Tradicional e o método Solaris-Sun da Sun Microsystems.

Na seção de "Soluções Comerciais" serão descritas as soluções (sistemas) para o ambiente Unix – Solaris (Tivoli da IBM e Unicenter da Computer Associates) e para o ambiente Linux (Volution da Caldera).

No Capítulo 5 serão discutidos os resultados obtidos, as dificuldades encontradas e futuras extensões sobre o assunto de manutenção de software de forma automatizada.

A presente dissertação abordará o problema de executar os procedimentos de manutenção de software no ambiente Unix, descrevendo as soluções encontradas, assim como as vantagens e desvantagens de cada solução (estudo comparativo). Tendo como objetivo a definição de regras para uma política de manutenção de software, visando a implementação e implantação destas soluções automatizadas que reduzem o tempo de execução de cada procedimento, de forma padronizada e documentada.

Este estudo comparativo destas soluções apresentadas demonstra a crescente evolução destas soluções para simplificar e automatizar os procedimentos de manutenção de software no tempo.

Chamaremos qualquer computador de estação de trabalho ou simplesmente de estação, independente de sua funcionalidade como servidor. Definimos como ambiente Unix (compatível com o Unix BSD), independente das plataformas Intel e Sparc [44] (base de operação do IC-Unicamp).

Capítulo 2

Tarefas de um administrador de redes

Neste capítulo serão descritos as tarefas de um administrador de rede quanto a sua complexidade, tendo como base o Instituto de Computação da Unicamp, ou IC-Unicamp. Este conjunto de tarefas definia a política de manutenção de software do IC-Unicamp, à época desta dissertação.

Classificação das tarefas

A qualidade que um administrador de rede possui é diretamente proporcional à quantidade de conhecimento que ele possui sobre os três grupos que compõe a rede: *Software*, *Hardware* e *peopleware*.

Os melhores administradores são os generalistas que atuam nos três grupos, conforme pesquisa realizada pela "Job Descriptions for SysAdmins" [26], que classifica os profissionais dentro dos Estados Unidos em Novice, Junior, Intermediate/Advanced e Senior.

Os objetivos a serem alcançados quando se administra os procedimentos de manutenção de software em uma rede de computadores são:

- **Planejamento**: as tarefas a serem executadas devem ser definidas de forma objetiva, sem redundância ou conflito, especificando o que, como, onde, quem e quanto tempo (estimativa) [06];
- **Padronização**: as tarefas devem ser padronizadas para garantir a qualidade destas (utilizar uma única linguagem de script) e reutilização de procedimentos já definidos;
- **Documentação**: as tarefas a serem executadas devem ser registradas, incluindo dados de quando foram iniciadas, quem executou, quando terminaram e quais etapas foram executadas e quais não;
- **Acompanhamento**: permitir que todas as tarefas a serem executadas possam ser monitoradas em modo interativo e/ou remoto;
- **Procedimentos automáticos**: as tarefas devem ser executadas em modo automático, sem intervenção do administrador de rede, sendo opcional a sua execução em modo interativo;

Muitos administradores de rede alcançam estes objetivos de modo empírico ou simplesmente nem alcançam.

Padronização de Peopleware

O administrador de rede deve classificar o grupo de *peopleware* em subgrupos para facilitar o trabalho de administrar o grupo de *software* que compõe a rede. Será explicado o algoritmo Base_Organizacional para geração destes subgrupos.

O algoritmo Base_Organizacional gera automaticamente estes subgrupos baseados na estrutura organizacional de uma empresa, neste caso, do IC-Unicamp:

```
* Algoritmo Base_Organizacional
Ler a Estrutura Organizacional da Empresa
Para cada Unidade da Empresa (Departamento, secretaria, etc)
        Criar um Subgrupo com Nome.
        Definir o Subgrupo: Nome, responsável, etc
        Definir os softwares necessários (grupo software)
        Definir os cursos necessários (grupo Peopleware)
        Definir os hardwares necessários (grupo hardware)
Registrar este grupo na rede.
* Fim algoritmo Base_Organizacional
```

Figura 2.1: Algoritmo Base_Organizacional.

Este algoritmo definirá, baseado na estrutura organizacional da empresa, as suas unidades e necessidades que serão administradas. Em necessidades de software, os aplicativos que o usuário utilizara no seu dia a dia, assim como os cursos que os usuários deverão fazer em função de sua necessidades.

Por exemplo, dentro da Unicamp o subgrupo IC-Unicamp será criado, e com base na estrutura organizacional do IC-Unicamp serão criados os seus subgrupos

Subgrupo IC-Unicamp		**Subgrupos a serem criados**
1.	IC-Unicamp	IC-Unicamp
1.1	Diretoria IC-Unicamp	Diretores
1.2	Secretaria IC-Unicamp	Secretaria
1.3	Secretaria Graduação IC-Unicamp	Secretaria-Graduação
1.4	Secretaria Pós-Graduação IC-Unicamp	Secretaria-Pós-graduação
1.5	Divisão de Laboratórios	Laboratórios

Esta documentação do grupo *peopleware* se faz necessária a fim de atingir os objetivos das tarefas do administrador de rede, que poderá ser usada no planejamento de futuras atividades de administração, por espelhar a realidade da empresa.

Os grupos de usuários serão criados automaticamente baseados neste algoritmo e com base nestes grupos, os usuários do IC-Unicamp serão classificados. A ocorrência de um professor estar no grupo professor e no grupo diretores devem ser preservadas e mantidas para que os serviços e funcionalidade da rede sejam disponibilizados conforme sua função e atividade. Exemplo, somente o grupo de diretores pode ter acesso aos arquivos financeiros do IC-Unicamp, e não o grupo de professores.

Padronização de Hardware

Dentro da padronização de hardware se faz necessária apenas a classificação do perfil de uma estação de trabalho, não havendo a necessidade de classificar o perfil dos outros hardwares envolvidos, tais como: hubs, switches, roteadores, impressoras etc. Mas é importante haver o registro de cada hardware quanto a sua localização, modelo, tipo, taxa de utilização, descrição detalhada etc. Estas informações são importantes para a gerencia da rede de computadores como um todo, alem de servir de documentação da mesma.

O conceito de **perfil da estação** permite criar uma definição funcional para uma estação específica, ou um grupo de estações que tenham a mesma funcionalidade. Com base neste perfil da estação, o administrador de rede executará as rotinas de manutenção de software definidas para este perfil. Este conceito permite criar uma documentação sobre a funcionalidade desta estação no ambiente da rede, contribuído também na criação de uma política de manutenção de software para esta estação.

O administrador de rede usa este perfil da estação para instalar, configurar, atualizar e remover software da estação de trabalho, mantendo assim a funcionalidade da estação.

Por exemplo:

Estação	**Localização**	**Perfil**
Tiete	Laboratório 10	Servidor_Comunicação
Jaguari	Laboratório 10	Servidor_E-mail Servidor_BD
Lab_A-01	Laboratório A	Estação_Básica
Lab_A-02	Laboratório A	Estação_Básica Estação_Estudante Estação_BD Estação_Compiladores

Tabela 2.1: Tabela de perfis do IC_Unicamp.

A estação Lab_A-01 possui o perfil Estação_básica, que determina que esta estação tem apenas o sistema operacional Solaris instalado e um conjunto de aplicativos. Havendo um problema com esta estação, o administrador de rede identificará o seu perfil (Estação_Básica) e executará os procedimentos corrigir o seu problema, mas respeitando o seu perfil.

O administrador de rede deve analisar as características físicas de uma estação para determinar sua possível funcionalidade. Os fatores analisados são: velocidade do processador, número de processadores, memória principal, memória secundaria, *floppy*, cd-rom, interface de rede e outros.

A tabela de perfis deve ser criada pelo administrador da rede com base nos tipos de funcionalidades que as estações executam dentro do ambiente de rede da organização. No exemplo do IC-Unicamp, foram criados os perfis de Servidor_Comunicação, Servidor_Intranet, Servidor_Backup, Servidor_Impressão, Servidor_E-mail, Servidor_BD, Servidor_SparcX, Estação_Básica, Estação_BD, Estação_Estudante, Estação_Compiladores, Estação_Avançada e Estação_Gráfica.

Algumas das soluções para a manutenção de software registram as informações dos perfis das estações em arquivos como em [10] e outras soluções registram em banco de dados (SGDB) como em [03].

É permitido/possível criar um perfil que é composto de dois outros perfis. Por exemplo, o perfil Estação_avançada é composto do perfil de Estação_Básica e Estação_Gráfica.

Após a definição da tabela de perfis pelo administrador de rede, este deve classificar os software da classe de aplicativos dentro de cada tipo de perfil. Por exemplo, o aplicativo *Netscape* será classificado dentro do perfil de Estação_Básica, por ser utilizado por todos os tipos de usuário e deve estar disponibilizado em todas as estações. Outro exemplo, o banco de dados *Mysql Server* deve ser classificado dentro do perfil de Servidor_BD, por ser um aplicativo que deve ser instalado em estação com perfil de servidor de banco de dados.

Como conseqüência, a cada novo software comprado, shareware ou livre, este deverá ser classificado dentro de um perfil. E somente depois desta classificação o mesmo poderá ser instalado nas estações.

O conceito de perfil não registra um inventário das estações de trabalho, como informações do tipo: velocidade do processador, a quantidade de memória primária, a quantidade de memória secundária, se tem *floppy* ou não. A solução de registrar inventários de equipamentos serão descritas na secção 4.4 do capítulo 4.

Com o conceito de perfil de estação, o administrador da rede terá um registro (definição) de quais softwares serão instalados, configurados e atualizados em um estação especifica.

Com base na implantação do conceito de **Perfil da estação**, os seguintes fatos serão observados na manutenção de software em um ambiente de rede:

- todas os procedimentos para manutenção de software nas classes sistemas operacionais e aplicativos serão padronizados e de fácil gerenciamento, garantindo a qualidade e reutilização destes procedimentos;

- todos os procedimentos de manutenção serão documentados, o que permite responder perguntas do tipo: que estações têm instalado o aplicativo *Netscape* e onde estão localizadas;
- todos os procedimentos poderão ser planejados em termos quantitativos e temporais, permitindo conhecer as estações que precisam ser atualizadas e quanto tempo será gasto neste procedimento (estimativa). As decisões sobre gerenciamento serão mais rápidas por haver uma base de dados para referenciá-las;
- permite que todos os procedimentos sejam definidos especificamente para um perfil, o que permite criar mecanismos para monitorar quanto a sua execução;

Em resumo, este conceito de perfil da estação exige que o administrador de rede refaça a classificação das estações de trabalho, respeitando sua atividade fim. Terá como retorno a documentação dos softwares instalados na rede e a padronização dos procedimentos de manutenção de software.

Na literatura atual, o termo perfil da estação possui o sinônimo de **classe da estação**, como em [51] que utiliza esta nomenclatura para definir os procedimentos de manutenção dos softwares nas estações da rede.

A figura 2.2 ilustra os vários estágios de configuração que um estação de trabalhos registra ao longo de seu ciclo de vida, conforme [07].

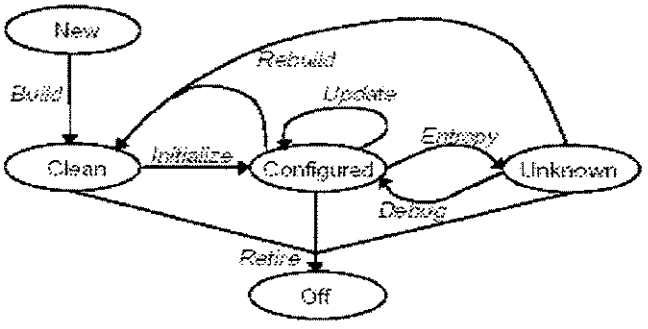

Figura 2.2: Procedimentos de manutenção de software.

Estes estágios podem ser convertidos diretamente para os procedimentos de manutenção de software a serem descritos dentro das classes de sistemas operacionais e na classe aplicativos.

2.1 Classe de Sistemas Operacionais

Nesta secção serão descritos os problemas inerentes à manutenção de software na **classe de sistemas operacionais**, sendo que os procedimentos a serem discutidos serão: instalação, atualização, configuração e remoção.

O ambiente Solaris [15] da Sun Microsystems é composto do sistema operacional SunOS e um conjunto de ferramentas que se interagem e com atribuições próprias. Dentro deste contexto, as ferramentas podem ser referenciadas por: `AdminSuite`, `automount`, `NIS+`, `DNS`, `NFS [SUN92a]`, `Patch Administration`, `SAF`, `Solaris Print Manager`, `Solstice`, `Openwindows` e outros.

A tarefa de administrar o ambiente Solaris será descrita nos procedimentos de instalação, configuração, atualização e remoção.

Procedimento de instalação

Situação A: O administrador de rede recebeu uma solicitação para instalar uma nova versão do sistema operacional Solaris em três estações de trabalho, como os nomes de Secr-01, Secr-02 e Secr-03, localizadas na secretaria do IC-Unicamp. Com base nesta solicitação o administrador de rede executa os seguintes passos:

Passo 1: o administrador se desloca de sua sala até a secretaria do IC-Unicamp.

Passo 2: identifica a primeira estação, de nome Secr-01, para a instalação do sistema operacional.

Passo 3: verifica as características físicas da estação: memória principal, memória secundaria, *Network Interface Card*, ou *NIC*, unidade de *floppy*, unidade de cd-rom.

Passo 4: verifica se a memória secundária (disco rígido) tem espaço suficiente para a instalação da nova versão de sistema operacional.

Passo 5: verifica a existência de dados (informações) que necessitam serem copiados, garantindo um *backup* destes dados, possivelmente para uma unidade de fita de *backup* da rede.

Passo 6: executa a instalação do sistema operacional: se a estação tem cd-rom, com o comando:

```
ok boot cdrom - install
```

Mas se a estação não possui cdrom, inicia via rede:

```
ok boot net - install
```

Passo 7: através do programa de instalação, de forma interativa, verifica as partições do disco rígido, se necessário fará as devidas alterações para a instalação.

Passo 8: através do programa de instalação, de forma interativa, seleciona os pacotes de software (patches) do sistema operacional que define a funcionalidade desta estação.

Passo 9: após a conclusão da instalação do sistema operacional (passo 8), executar um conjunto de rotinas, normalmente *scripts*, que configuram esta estação na rede e permite que ela utilize todos serviços da rede interna (Intranet do IC-Unicamp).

Passo 10: retornar o *backup* das informações para o disco rígido.

Passo 11: executar a instalação de aplicativos necessários para manter a funcionalidade desta estação, tais como editor de texto, programa de *e-mail* etc.

Passo 12: disponibilizar a estação para o usuário, normalmente um mini-curso que explicará as alterações ocorridas devido à instalação da versão do sistema operacional Solaris.

O administrador de rede deve repetir novamente os procedimentos do Passo 2 ao Passo 11 para as estações Secr-02 e Secr-03. Em média, este procedimento de instalação pode gastar um tempo de 2 horas e 30 minutos, logo a sua execução total deve consumir menos de 6 horas, devido ao fato do administrador iniciar a instalação do sistema operacional das estações Secr-02 e Secr-03 em paralelo com a estação Secr-01.

Os problemas relacionados à instalação do sistema operacional Solaris para uma situação típica são descritos como:

- O administrador de rede deve estar disponível para atender a solicitação de instalação, por ser da forma interativa, e exigindo sua presença durante todo o procedimento.
- Por não existir uma documentação detalhada das estações ou um software de inventário, o administrador deve verificar pessoalmente as características físicas das estações e respeitar os requisitos mínimos da instalação.
- Por não existir um padrão do repositório dos dados ou informações do usuário, o administrador de rede deve verificar todo o disco rígido e fazer um *backup* das informações importantes. Em alguns casos, o usuário deve estar à disposição do administrador para a confirmação do procedimento de *backup*.
- Repetição do procedimento, de forma interativa, para as outras estações Secr-02 e Secr-03.
- Documentação destes procedimentos para futuras reutilizações.
- Registrará a execução de tais procedimentos: quem executou, quando, onde e porque, visando à montagem de um banco de dados (auditoria).

Estas estimativas de tempo podem variar conforme os pacotes de softwares que serão instalados. Estes garantem a funcionalidade desejada para a estação. No pior caso, o tempo gasto pode chegar a 6 horas para uma única estação e no melhor caso em 2 horas.

Procedimento de configuração

Situação B: o administrador de rede recebeu uma solicitação para configurar uma nova impressora HP MV4 nas estações da secretaria do IC-Unicamp. Com base nesta solicitação o administrador de rede executa os seguintes passos:

Passo 1: identifica as estações via rede, de nomes Secr-01, Secr-02 e Secr-03, para configurar a nova funcionalidade das estações.

Passo 2: verifica as características físicas da estação: memória principal, memória secundária, *Network Interface Card*, ou *NIC*, unidade de *floppy*, unidade de cd-rom.

Passo 3: planeja o servidor de impressão e a nova fila de impressão, assim como os procedimentos para fazer sua configuração.

Passo 4: executa os procedimentos para configurar uma estação, normalmente via *script*.

Passo 5: imprime algumas folhas para confirmar se os procedimentos estão corretos.

Passo 6: se o teste de impressão tem erros então o administrador de rede deve se deslocar até a secretaria para resolver manualmente o problema.

Passo 7: após a conclusão da configuração, divulgará via *e-mail* aos usuários sobre a disponibilidade deste novo serviço de rede.

O administrador de rede deve repetir novamente os procedimentos do Passo 4 ao Passo 7 para as estações Secr-02 e Secr-03. Em média, este procedimento de configuração pode gastar um tempo médio de 15 minutos, logo a sua execução total deve consumir pelo menos 45 minutos, devido ao fato do administrador executar as configurações seqüencialmente e validá-las individualmente.

Os problemas relacionados a configurar uma nova impressora em um secretaria para uma situação típica são descritos como:

- por não haver padronização dos procedimentos, o administrador de rede deve reutilizar um procedimento parecido ou adaptar algum procedimento que tenha a mesma funcionalidade.
- repetição do procedimento, de forma interativa, para as outras estações Secr-02 e Secr-03.
- documentar estes procedimentos para futuras reutilizações.
- registrar a execução de tais procedimentos, quem executou, quando, onde, e porque, visando a montagem de um banco de dados (auditoria).

Estas estimativas de tempo podem variar conforme o tipo de configuração que se está realizando no ambiente da rede para disponibilizar a novas funcionalidades para as estações. Em alguns casos, o tempo gasto pode chegar a 2 horas para uma única estação.

O cálculo do tempo total a ser gasto é linear e pode ser obtido pela equação:

Tempo = Nun_estação X Tempo_configuração

onde Num_estação é o número de estações que serão configuradas, e Tempo_configuração é o tempo médio para realizar a configuração. Para obter o tempo total mínimo: seria o melhor caso, quando não ocorrem erros de impressão que exijam o deslocamento do administrador da rede; ou tempos total máximo: seria o pior caso, quando administrador se desloca em todas as configurações.

Procedimento de atualização

Situação C: o administrador de rede recebeu uma solicitação para aplicar um novo *patch*, de número #18092888, do sistema operacional Solaris para todas as estações dentro do IC-Unicamp. Com base nesta solicitação o administrador de rede executa os seguintes passos:

Passo 1: identificar todas as estações, via rede, que serão atualizadas com uma nova funcionalidade.

Passo 2: dividir as estações em grupos, tendo como parâmetros suas características físicas, localização e funcionalidade dentro da rede, assim como sua disponibilidade para ser reinicializada.

Passo 3: planejar como executar os procedimentos dentro de cada grupo ou estação e os respectivos horários em que serão atualizadas.

Passo 4: divulgar via *e-mail* aos usuários de determinadas estações, o horário em que estas estações serão reinicializadas para que a atualização tenha efeito.

Passo 5: executar os procedimentos para atualizar cada estação de um grupo, normalmente via *script*. Alguns *patches*, quando atualizados, exigem que a estação seja reinicializada.

Passo 6: confirmar a atualização em cada estação, via rede.

Passo 7: Em caso de erro na atualização, analisar as causas e tentar novamente o Passo 5.

Passo 8: após a conclusão da atualização, divulgar via *e-mail* aos usuários a disponibilidade desta correção na rede.

O administrador de rede deve repetir novamente os procedimentos do Passo 4 ao Passo 8 para as todas as estações do grupo e para todos os grupos criados neste procedimentos de atualização. Normalmente, estes tipos de procedimentos são executados em horários pré-programados e fora do horário comercial, para que o impacto sobre a rede seja mínimo.

Em média, este procedimento de atualização pode gastar um tempo de 15 minutos, entre executar o procedimento, via script, e reinicializar a estação.

Os problemas relacionados ao procedimentos de atualizar um novo *patch* para uma situação típica são descritos como:

- Por não existir padronização dos procedimentos, o administrador de rede deve reutilizar um procedimento parecido ou adaptar algum procedimento que tenha a mesma funcionalidade.
- Repetição do procedimento, de forma interativa, para as outras estações.

- Documentar estes procedimentos para futuras reutilizações.
- Registrar a execução de tais procedimentos, quem executou, quando, onde e porque, visando a montagem de um banco de dados (auditoria).

As estimativas de tempo total podem variar conforme o tipo de atualização e os horários planejados de sua execução para minimizar o impacto sobre a rede. Em alguns casos, pode se calcular dias para uma simples atualização, devido as reinicializações das estações.

Procedimento de remoção

É importante ressaltar que o procedimento de remoção existe e quando é usado pelos administradores de rede, tem o mesmo grau de complexidade e dificuldade do procedimento de atualização.

De forma resumida foi possível descrever os vários passos para a realização de manutenção de software na classe de sistemas operacionais e os problemas envolvidos dentro de cada procedimento tratado. A solução de tais problemas é a completa documentação e automatização de tais procedimentos sem a necessidade de deslocamento do administrador de rede.

2.2 Classe de Aplicativos

Nesta secção serão descritos os problemas inerentes à manutenção de software na classe de aplicativos, sendo que os procedimentos a serem discutidos serão: instalação, atualização, configuração e remoção.

Os aplicativos podem ser distribuídos pelo fabricante através de duas formas que são: ***Package*** ou ***Script***. As particularidades de cada forma de distribuição serão descritas a seguir:

Forma *Script*

A primeira forma de distribuição de software que surgiu no ambiente Unix foi o ***Script***. O grupo de analistas/programadores de uma organização, que desenvolveu o aplicativo, realiza os seguintes passos para distribuir este produto aos usuários (consumidores):

Passo 1: o coordenador ou chefe ou responsável pelo projeto define nome e versão que o produto receberá e será identificado no mercado. Por exemplo: emacs_v20.34.

Passo 2: registra a relação de arquivos que compõem o aplicativo (executável, configuração e documentação). Normalmente, são registrados nos arquivos do tipo: `readme.txt` ou `install.txt`, mas podem estar descritos em arquivos do tipo man. Deve haver um arquivo de *script* para cada procedimento de manutenção, que são identificados pelo seu próprio nome. Por exemplo: install_emacs_v20.34, config_emacs_v20.34, update_emacs_v20.34 e remove_emacs_v20.34.

Passo 3: verifica-se todos os arquivos que compõe o software estão identificados e documentados quanto a sua funcionalidade.

Passo 4: o aplicativo é agrupado usando uma ferramenta `tar` ou `gzip`, gerando um único arquivo que será usando na sua distribuição. Por exemplo: emacs_v20.34.tar ou emacs_v20.34.gzip

Passo 5: são realizados testes em vários ambientes e versões diferentes de Unix para garantir sua funcionalidade. Havendo erros, o processo retorna ao Passo 2 para sua correção.

Em resumo, na forma de distribuição via *Script*, o aplicativo encapsulado via *tar* ou *gzip*, é distribuído, e os procedimentos de manutenção (instalação, configuração, atualização e remoção) são escritos via *Script*.

É importante esclarecer que os passos anteriores são uma forma de distribuição do aplicativo, mas muitos desenvolvedores só distribuem o *Script* de instalação do aplicativo. Como conseqüência, o administrador da rede deve analisar o *Script* de instalação e desenvolver os *Scripts* de configuração, atualização e remoção.

As desvantagens desta forma de encapsular o aplicativo para sua distribuição podem ser resumidas em:

- o administrador de rede deve ter conhecimentos avançados dos mais variados tipos de *Shell*, do tipo: *Kornshell* [04] e [21], *Cshell* e *Bourne Shell* [08], para realizar as manutenções necessárias no aplicativo;
- a documentação, na sua falta, baseando-se nas versões anteriores do aplicativo e na experiência adquirida pelo administrador da rede nestes procedimentos de manutenção.
- o aplicativo pode não testar suas dependências em relação a outros softwares ou mesmo do ambiente Unix, que torna a sua manutenção mais complexa. Por exemplo: um aplicativo `ViewDraw` necessita de ambiente gráfico `OpenView` para sua execução, mas não foi verificada tal situação no procedimento de instalação.

A única vantagem desta forma de distribuição de aplicativo é que o administrador da rede tem a liberdade de redefinir a forma como o aplicativo sofrerá manutenção, mas deverá testar sua funcionalidade de forma exaustiva para garantir sua qualidade e confiabilidade.

Forma *Package*

O termo *Package* ou *Packaging* reference-se genericamente aos métodos de distribuição e instalação de um software da classe aplicativo.

Define-se um *Package* como uma coleção de arquivos e diretórios compactados dentro de um formato pré-definido. A especificação deste formato é descrita em detalhes no suplemento do *System V Interface Definition* dentro da seção *Application Binary Interface*, ou ABI, em [18]. O ambiente operacional Solaris dispõe de um conjunto de ferramentas que interpreta este formato e trata as informações contidas no *package*, permitindo executar os procedimentos de instalar, atualizar e remover um software. Além destes procedimentos, é possível consultar os arquivos que compõem um *package* e verificar se um determinado *package* está instalado ou não no ambiente Solaris.

Dentro do ambiente Solaris existe um banco de dados que armazena as informações sobre cada *package* instalado na estação, registrando informações do tipo: nome do *package*, versão, categoria, arquitetura, diretório base, vendedor (empresa responsável), descrição resumida, data de instalação, número `pstamp`, *Status* e um resumo dos arquivos que compõem o *package*.

Os arquivos que estão compactados dentro do *package* podem ser tipo: executável, documentação ou configuração.

Quando o procedimento de instalar um *package* é executado através do software `admintool` ou `pkgadd` para o ambiente Solaris, são executados os seguintes passos:

Passo 1: extrair as informações sobre o *package*.

Passo 2: verificar se o *package* está instalado.

Passo 3: verificar sua dependência em relação a outros *packages*.

Passo 4: descompactar e copiar os arquivos contidos no *package* para o diretório destino, informado pelo administrador de rede ou utilizar o diretório *default*.

Passo 5: registrar as informações sobre o *package* no banco de dados.

Passo 6: verificar os arquivos copiados e as informações registradas no banco de dados e informar o administrador que o procedimento de instalação foi executado com sucesso.

Este procedimento de instalação é interativo, mas pode ser parametrizado para que o administrador não seja questionado.

Quando o procedimento de remover um *package* é executado através do software `admintool` ou `pkgrm` são executados os seguintes passos:

Passo 1: consulta as informações sobre o *package* no banco de dados.

Passo 2: verifica os *package* que tem dependência em relação a este, questionando o administrador de rede sobre o que fazer.

Passo 3: remove os arquivos que compõem o *package* do diretório instalado.

Passo 4: remove informações sobre o *package* do banco de dados.

Passo 5: verificar os arquivos removidos e as informações registradas no banco de dados e informar o administrador que o procedimento de remoção foi executado com sucesso.

Este procedimento de remoção é interativo, mas pode ser parametrizado para que o administrador não seja questionado.

A forma de manutenção de software via *Package* contém inúmeras vantagens em relação à forma *Script*, mas existe uma forma híbrida que contém as duas formas anteriores, onde o Script de instalação contém a parametrização correta com os comandos de manipulação dos *packages*.

Para gerar um *package*, o(s) desenvolvedor(es) do aplicativo utiliza(m) uma ferramenta do tipo `mkpkg` em [39] para encapsular os programas fontes, `libraries`, o programa `make` e a documentação necessária do aplicativo. Este exemplo de ferramenta registra: o titulo (nome) do aplicativo, descrição do aplicativo, a lista de arquivos que compõe o package, a lista de dependências do aplicativo e scripts de pré-instalação, configuração e pós-instalação.

Nas próximas seções serão descritos os procedimentos de instalação, atualização, configuração e remoção para a manutenção de software na classe de aplicativos, descrevendo em detalhes as particularidades e seus problemas.

Procedimento de instalação

Situação D: O administrador de rede recebeu uma solicitação para instalar uma nova versão do editor de texto *emacs* no laboratório L02, em vinte estações de trabalho com os nomes de EA-01, EA-02, ..., ... e EA-20. Com base nesta solicitação o administrador de rede executa os seguintes passos:

Passo 1: o administrador verifica a existência de outras versões do *emacs*.

Passo 2: identifica o servidor (estação), no qual será executado o procedimento de instalação.

Passo 3: planeja como executar o procedimento, respeitando as versões anteriores. Os aplicativos, na sua grande maioria, são instalados nos diretórios `/usr/local/bin/` como padrão, ou em outros casos em `/usr/local/sbin`. Mas existe a opção de instalar em outro diretório, conforme parâmetro do administrador de rede.

Passo 4: executa a instalação da nova versão.

Passo 5: identifica os usuários de *emacs*.

Passo 6: comunica via *e-mail* aos usuários sobre a nova versão e quais as alterações que eles devem realizar para que a nova versão execute em seu ambiente de trabalho (sessão), por exemplo arquivo o *.login*.

Passo 7: registra o procedimento como executado, com data, hora e quem executou.

Este exemplo pode ter outras variações em função da forma que o software foi instalado e distribuído dentro do ambiente de rede.

Os problemas relacionados à instalação de uma versão do editor *emacs* no ambiente operacional Solaris para uma situação típica são descritos como:

- tratar a estrutura de diretórios para manter as várias versões do mesmo software, garantindo um ambiente estável e sem conflitos. Esta estrutura de diretórios deve ser flexível, confiável e autodocumentável. Este problema é descrito como **problema de espaço de nomes**.
- disponibilizar, documentar, identificar, controlar e tratar as várias configurações das versões do mesmo aplicativo no ambiente Solaris. Este problema é descrito como **problema de controle de versões**.
- comunicar somente aos usuários de *emacs* que existe uma nova versão, e que os mesmos devem configurar seus ambientes Solaris para a nova versão. Os usuários que não utilizam emacs não devem receber tal comunicação. Este problema de identificação dos usuários e das estações, é descrito como **problema do monitoramento do uso de software**.
- documentar estes procedimentos de instalação para futuras reutilizações.
- registrar a execução de tais procedimentos: quem executou, quando, onde e porque, visando à montagem de um banco de dados (auditoria).

A estimativa de tempo a ser gasto pode variar conforme a dificuldade e complexidade do procedimento de instalação na resolução dos problemas de espaço de nomes, controle de versão e monitoramento do uso de software. Nesta situação D, o problema das N repetições do mesmo procedimento foi reduzido a uma única instalação do aplicativo no servidor que atende a demanda do laboratório. Mas em alguns casos, o desempenho do servidor ou da rede pode inviabilizar esta instalação, gerando a necessidade do aplicativo ser instalado e configurado nas N estações do laboratório.

Para os aplicativos que não permitem alterar o parâmetro de diretório padrão de instalação, por exemplo `/usr/local/bin`, o administrador tem mais um problema a resolver no procedimento de instalação: obter os programas fonte do aplicativo e recompilá-lo com o novo diretório padrão. E para os casos em que recompilar não é possível, o administrador de rede deverá tratar como um problema de espaço de nomes, devido ao fato de ocorrer a sobreposição de uma versão sobre a outra.

Dentro do contexto do problema de controle de versões, o administrador deve identificar: a origem de cada arquivo instalado, a sua versão e quando foi instalado.

Procedimento de configuração

Situação E: O administrador de rede recebeu uma solicitação para reconfigurar o aplicativo *Autocad* do laboratório L02 nas vinte estações de trabalho, com os nomes de EA-01, EA-02, ..., ... e EA-20. Esta reconfiguração deve informar o novo repositório dos arquivos gráficos gerados pelos usuários. Com base nesta solicitação o administrador de rede executa os seguintes passos:

Passo 1: o administrador verifica quais estações têm o aplicativo instalado e quais versões serão reconfiguradas.

Passo 2: planeja como executar o procedimento, em quais estações e em quais versões do aplicativo.

Passo 4: executa a reconfiguração do aplicativo.

Passo 5: comunica via *e-mail* aos usuários sobre a nova configuração e quais as alterações que eles devem realizar para que a nova configuração tenha efeito em seu ambiente de trabalho (sessão), por exemplo o arquivo *.login*.

Passo 6: registra o procedimento como executado, com data, hora e quem executou.

Este exemplo pode ter outras variações em função do tipo de aplicativo e da configuração a ser realizada no software dentro do ambiente da rede.

Os problemas relacionados à configuração de software no ambiente operacional Solaris para uma situação típica são descritos como:

- como identificar e tratar as estações que serão configuradas. Este problema deve ser tratado como **problema do monitoramento do uso de software** .
- como documentar, registrar e tratar as várias versões do mesmo aplicativo no ambiente Solaris. Este problema é tratado como **problema de controle de versões**.
- como comunicar somente aos usuários do *Autocad* que existe uma nova configuração, e que os mesmos devem corrigir as configurações do seu ambiente Solaris. Já os usuários que não utilizam *Autocad* não devem receber tal comunicação. Este problema de identificar que software está instalado e quem está utilizando, é descrito como **problema do monitoramento do uso de software**.
- como documentar estes procedimentos de configuração para futuras reutilizações.
- como tratar a execução das N repetições do mesmo procedimento.
- registrar a execução de tais procedimentos, quem executou, quando, onde, e porque, visando a montagem de um banco de dados (auditoria).

Dentro da situação E, o problema de espaço de nomes não foi descrito e tratado, mas pode vir a estar no contexto do procedimento de configurar um aplicativo.

As estimativas de tempo a ser gasto podem variar conforme o número de estações, a dificuldade e complexidade do procedimento de configuração na resolução dos problemas de controle de versões, espaço de nomes e monitoramento do uso de software.

Procedimento de atualização

Situação F: O administrador de rede recebeu uma solicitação para atualizar as novas fontes de letra do aplicativo `xfig` do laboratório L02 nas vinte estações de trabalho, com os nomes de EA-01, EA-02, ..., ... e EA-20; Com base nesta solicitação o administrador de rede executa os seguintes passos:

Passo 1: o administrador verifica quais estações têm o aplicativo instalado e quais versões serão atualizadas.

Passo 2: planeja como executar o procedimento, em quais estações e em quais versões do aplicativo.

Passo 4: executa a atualização do aplicativo.

Passo 5: comunica via *e-mail* aos usuários sobre as novas fontes de letras e quais as alterações que eles devem realizar para que a nova atualização tenha efeito em seu ambiente de trabalho (sessão).

Passo 6: registrar o procedimento como executado, com data e hora, e quem executou.

Este exemplo pode ter outras variações em função do tipo de aplicativo e da atualização a ser realizada no software dentro do ambiente da rede.

Os problemas relacionados à atualização do aplicativo `xfig` no ambiente operacional Solaris para uma situação típica são descritos como:

- como identificar e tratar as estações que serão atualizadas. Este problema deve ser tratado como **problema do monitoramento do uso de software** .
- como documentar, registrar e tratar as várias versões do mesmo aplicativo no ambiente Solaris. Este problema é tratado como **problema de controle de versões**.
- como comunicar somente aos usuários do `xfig` que existe uma nova coleção de fonte para letras, e que os mesmos devem atualizar seu ambiente Solaris. Já os usuários que não utilizam `xfig` não devem receber tal comunicação. Este problema de identificar que software está instalado e quem está utilizando, é descrito como **problema do monitoramento do uso de software**.
- como documentar estes procedimentos de configuração para futuras reutilizações.
- como tratar a execução das N repetições do mesmo procedimento.
- registrar a execução de tais procedimentos, quem executou, quando, onde, e porque, visando a montagem de um banco de dados (auditoria).

Dentro da situação F, o problema de espaço de nomes não foi descrito e tratado. Mas pode vir a ser tratado no contexto do procedimento de atualização.

As estimativas de tempo a serem gasto podem variar conforme o número de estações, a dificuldade e complexidade do procedimento de atualização na resolução dos problemas de controle de versões, espaço de nomes e monitoramento do uso de software.

Procedimento de remoção

Situação G: O administrador de rede recebeu uma solicitação para remover as versões do editor de texto `vi` que não foram utilizadas (executadas) nos últimos 6 meses no laboratório L02, nas suas vinte estações de trabalho, com os nomes de EA-01, EA-02, ..., ... e EA-20. Com base nesta solicitação o administrador de rede executa os seguintes passos:

Passo 1: o administrador identificará quais versões de `vi` estão instaladas e sua localização. Selecionará as que não são utilizadas nos últimos 6 meses.

Passo 2: planejará como executar o procedimento, em quais estações e em quais versões do aplicativo.

Passo 4: executará a remoção do aplicativo.

Passo 5: comunicará via *e-mail* aos usuários sobre a remoção de tais versões e quais as alterações que eles devem realizar para que as versões instaladas continuem ativas e confiáveis em seu ambiente de trabalho (sessão).

Passo 6: registrará o procedimento como executado, com data, hora e quem executou.

Este exemplo, pode ter outras variações em função do tipo de aplicativo e da remoção a ser realizada no software dentro do ambiente da rede.

Os problemas relacionados de remoção de uma ou várias versões de um aplicativo no ambiente operacional Solaris para uma situação típica são descritos como:

- como identificar as versões a serem removidas. Este problema deve ser tratado como **problema do monitoramento do uso de software**.
- como identificar, documentar e remover os arquivos que compõem uma versão do aplicativo. Este problema é tratado como o **problema de controle de versões**.
- como remover, documentar e tratar as várias versões removidas no ambiente Solaris. Este problema é tratado como **problema de espaço de nomes**.
- como comunicar somente aos usuários que utilizavam as versões removidas que as mesmas não existem mais, e que os mesmos devem atualizar seu ambiente Solaris. Para os outros usuários, não informar nada. Este problema de identificar que software esta instalado e quem esta usando, é descrito como **problema do monitoramento do uso de software**.
- como documentar estes procedimentos de remoção para futuras reutilizações.
- como tratar a execução das N repetições do mesmo procedimento.

- registrar a execução de tais procedimentos, quem executou, quando, onde, e porque, visando a montagem de um banco de dados (auditoria).

As estimativas de tempo a serem gasto podem variar conforme o número de estações, a dificuldade e complexidade do procedimento de remoção. Deve ser ressaltado que o tempo gasto para a remoção de aplicativos que tem a forma de instalação do tipo *Script* são muito superiores ao do tipo *package*.

Pode-se resumir que os problemas a serem tratados nesta dissertação sobre a manutenção de software nas classes de sistemas operacionais e aplicativos visam obter procedimentos automatizados, padronizados e documentados. E descrevermos as soluções para a classe de aplicativos nos problemas de espaço de nomes, controle de versões e monitoramento do uso de software.

Capítulo 3
Manutenção do sistema operacional Solaris

Será descrito neste capítulo a manutenção de software na classe de sistemas operacionais para os procedimentos de instalação, configuração, atualização e remoção (*patches*). As soluções descritas neste capítulo servem para todas as versões de Solaris, independentemente da arquitetura ser Intel ou Sparc [44].

Os modelos aqui apresentados foram agrupados e documentados em função da metodologia utilizada para a manutenção do sistema operacional Solaris. As formas usadas para coletar tais metodologias de manutenção de software foram entrevistas, *e-mail*, *internet*, fóruns de discussões, pesquisas em livros de administração de rede e anais de congressos [01] e [02].

3.1 Modelo Tradicional

O modelo **Tradicional** é o método mais utilizado pelos administradores de rede para o procedimento de manutenção de software devido a sua simplicidade de execução e acompanhamento.

Procedimento de instalação

O administrador pode optar por três formas de instalar o sistema operacional Solaris: a partir de uma unidade de cd-rom da própria estação; ou de uma unidade de cd-rom montada remotamente, usando NFS, ver [SUN92a]; ou utilizar uma imagem do cd Solaris, que foi copiada do cd Solaris para um disco rígido do servidor da rede. Esta última opção é a forma mais rápida de instalação, alem de liberar a unidade cd-rom para uma outra utilização.

Os seguintes passos descrevem a metodologia para executar o procedimento de instalação do sistema operacional Solaris:

Passo 1: O administrador recebe uma solicitação para executar um procedimento de instalação.

Passo 2: analisa o tempo a ser gasto no procedimento.

Passo 3: agenda a execução do procedimento.

Passo 4: desloca-se de sua sala até o local da estação.

Passo 5: realiza o *backup* das informações do(s) usuário(s) da estação.

Passo 6: executa o procedimento de instalação do sistema operacional de forma interativa.

Passo 6.1: o administrador confirma as informações de: *date e time*, *time zone*, *site netmask* e a *language-specific*.

Passo 6.2: define o tipo de instalação a ser realizada: *Standalone System*, *Server* ou *Dataless Client*.

Passo 6.3: redefine a partição de disco local e salva.

Passo 6.4: seleciona e instala os pacotes de software do Solaris a serem instalados, respeitando o perfil da estação.

Passo 7: configura a estação para o ambiente de rede, via manual ou *script*.

Passo 8: instala os pacotes de aplicativos na estação ou configura o ambiente do Solaris para utilizar os aplicativos via rede.

Passo 9: restaura o *backup* das informações de usuário para a estação.

Passo 10: verifica se os procedimentos foram executados corretamente e comunica aos usuários.

Passo 11: retorna a sua sala.

Em resumo, o modelo **Tradicional** é composto de três etapas: instalar o sistema operacional, configurar o ambiente para a rede, e instalar os aplicativos necessários.

As desvantagens observadas no modelo **Tradicional** para o procedimento de instalação do sistema operacional Solaris são inúmeras e descritas como:

- Execução do procedimento é de forma interativa e exige um acompanhamento e tratamento manual das mensagens oriundas da execução do procedimento.
- Falta de documentação dos procedimentos executados, como: as partições definidas, pacotes do Sistema Operacional instalados, os aplicativos instalados e configurados.
- Falta de um histórico dos procedimentos executados. Ter o registro de quando o procedimento foi executado, por quem e quais os passos executados.
- Falta de padronização dos procedimentos. Permitir a reutilização dos procedimentos já utilizados.
- Falta de planejamento da execução, onde se restringe ao horário de inicio da execução do procedimento e previsão do seu término.

- Falta de comunicação padronizada com o(s) usuário(s), informando sobre os procedimentos executados e as suas conseqüências.

Em resumo, por ser o método mais simples é o que contem as maiores falhas.

O tempo gasto no procedimento de instalação é linear, com uma média de 4 horas para sua execução. Havendo a necessidade de sua execução em outras estações, exemplo da situação A da seção 2.1 do capítulo 2, o tempo total médio passa de 120 horas, o que representaria 5 dias corridos, o que seria inexeqüível. Se considerarmos a hipótese de meio período, o tempo disponível de administrador de rede, seria executado em 30 dias. Este cálculo pode ser representado pelo gráfico de tempo médio de execução abaixo.

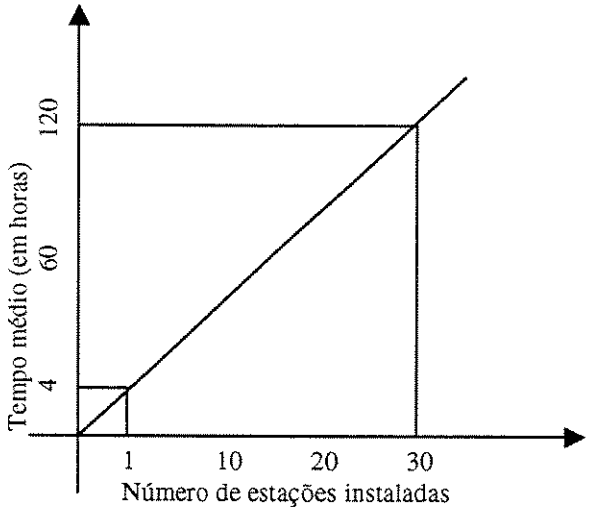

Figura 3.1: Tempo médio de instalação por número de estações no modelo Tradicional.

O tempo médio pode não ser linear se a forma de instalação for a opção da imagem do cd-rom Solaris no disco local do servidor, onde o administrador consegue instalar no máximo duas estações em paralelo nos passos 5, 6.4, 8 e 9, quando ocorre a leitura do Solaris (imagem do cd-rom) e a sua gravação no disco local da estação destino. A primeira instalação consome 4 horas em média e a segunda terminaria 2 horas e 30 minutos após a primeira, totalizando 6 horas e 30 minutos para duas estações. Realizando a instalação do Solaris nas 30 estações em 15 dias, haveria uma melhora de 50% do tempo gasto.

3.2 Modelo IC-Unicamp – Jumpstart

Nesta seção, será descrito em detalhes o modelo de instalação do sistema operacional Solaris no IC-Unicamp que utiliza a tecnologia Jumpstart criada pela Sun Microsystems.

A metodologia **Jumpstart** é baseada no processo de boot via rede. Este processo aproveita a estrutura de boot criada nas estações do tipo *Diskless Client*, que não possui disco rígido para armazenar o sistema operacional Solaris. A estação *Diskless Client*, quando ligada, encontra um Servidor de Boot, ou *Boot Server*, que faz a carga do sistema operacional Solaris, do servidor para a estação, montando um sistema de arquivos do servidor usando NFS [SUN92a]. Esta tecnologia esta disponível desde da versão SunOS 2.x, mas somente na versão SunOS 5.x foi disponibilizada para o procedimento de instalação.

O *Boot server* identifica a estação pelo seu endereço de *ethernet*, e sua localização deve ser na mesma sub-rede da estação.

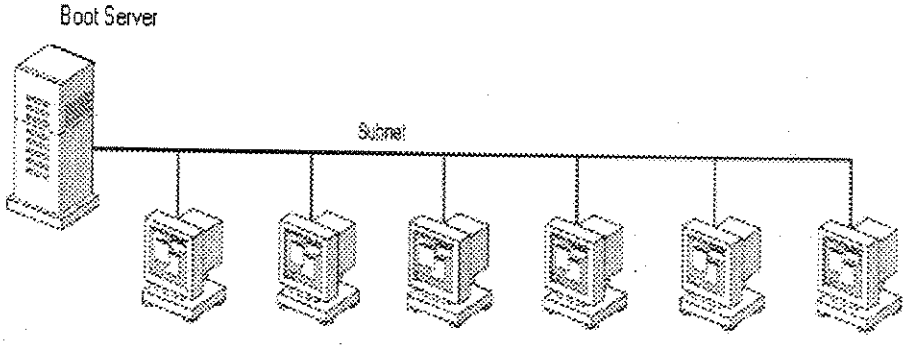

Figura 3.2: Exemplo de *Boot Server*.

Metodologia Jumpstart do Solaris

A método **Jumpstart** foi criado pela Sun microsystems para automatizar o procedimento de instalação do sistema operacional Solaris. A estrutura necessária descrita no apêndice A que deve ser criada pelo administrador da rede para que o método funcione é:

- **Boot Server:** um servidor de boot que identifica a estação e disponibiliza via rede a carga do sistema operacional Solaris para a estação, que monta um sistema de arquivos usando NFS [42].
- **Install Server**: um servidor de instalação que possui o conteúdo do cd-rom Solaris. Disponibiliza os pacotes de software do Solaris a serem instalados na estação a partir do diretório `/export/install`.
- **Name Server**: um servidor de nomes, do tipo NIS ou NIS+ [45], que contem informações sobre o ambiente de rede, do tipo: endereços *ethernet*, endereços IP, *host-names*, *time zone*, *geografic region*, *netmask* e outras informações necessárias ao procedimento de instalação, através do programa `sysidtool`.

- **Profile Server**: um servidor de perfil que identifica e define o perfil de instalação da estação. No diretório compartilhado `/jumpstart` reside o arquivo `rules.ok` que identifica o perfil da estação (*rule*), e o arquivo `profile` que define a forma como será instalado o Solaris, quanto ao tipo de instalação, as partições e os pacotes do Solaris que serão instalados.

O *Boot Server*, o *Install Server* e o *Profile Server* podem ser criados em uma única estação (servidor) em qualquer ponto da rede local, mas a localização do *Boot server* deve ser na mesma sub-rede da estação para funcionar corretamente.

Eis a seqüência de passos que o método Jumpstart executa:

Passo 1: a estação inicializa o procedimento de instalação via rede. O servidor de boot identifica e disponibiliza a carga do sistema operacional Solaris, e monta um sistema de arquivos usando NFS [42].

Passo 2: o software de instalação localiza o arquivo `rules.ok` que identificará o perfil de instalação da estação no servidor de install.

Passo 3: o programa `sysidtool` identifica as informações de configuração da rede no servidor de nomes, o que automatiza o procedimento. Este programa utiliza outros utilitários, como `kdmconfig`, `sysidconfig`, `sysifnet`, `sysidnis`, `sysidroot` e `sysidsys`, para obter as informações necessárias.

Passo 4: o software de instalação identifica uma *rule* (regra), ou perfil de instalação, no arquivo `rules.ok`.

Passo 5: baseado no perfil, o software de instalação identifica e localiza o arquivo `profile` da estação (perfil). O administrador de rede configurou um arquivo para cada tipo de perfil das estações.

Passo 6: o software de instalação inicia o procedimento de instalação do sistema operacional Solaris de forma automatizada, com base nas informações que definem o perfil da estação, contidas no arquivo `profile`.

Quando o software de instalação não encontra uma *rule* (regra) no arquivo `rules.ok` que identifica o perfil da estação, o procedimento torna-se interativo. O administrador de rede deve configurar o perfil através das várias janelas do software de instalação, criando o perfil da estação.

Utilizando o conceito de Perfil da estação, descrito em detalhes na seção "Padronização de Hardware" do capítulo 2, o administrador de rede cria o arquivo `rules.ok`, registrando uma *rule* para cada estação da rede ou para um grupo de estações que têm o mesmo perfil.

O arquivo `rules.ok` contém palavras-chaves que são utilizadas para identifica o perfil da estação. Estas palavras-chaves comparam as características físicas da estação ou a configuração que a estação utiliza na rede para identificar o perfil da estação. Os tipos de palavras-chaves ou critérios de identificação podem ser:

- **Arch**: identifica o tipo de arquitetura da estação, podendo ser do tipo: Sparc ou Intel [44].

- **Domainname**: identifica a estação que se localiza sob um domínio. Com base nas informações do *Name server* esta *rule* é valida ou não. Por exemplo, instalar as estações do subdomínio Las.IC.Unicamp.Br.
- **Disksize**: permite comparar o parâmetro de *disk_name* que identifica o disco rígido da estação, e o parâmetro *size_range* que identifica o tamanho do disco. Exemplo: `disksize c0t3d0 10-30` onde identifica as estações que têm disco rígido alocados como c0t3d0 e do tamanho de 10 a 30 Gigabytes.
- **Hostname**: compara o nome de *host* da estação.
- **Installed**: permite comparar os parâmetros de *slice*[2] e *version*, onde é comparado o *slice* do sistema operacional instalado na estação e sua versão. A informação de *slice* e versão estão registradas no arquivo `INST_RELEASE` no diretório `/var/sadm/softinfo`.
- **Karch**: compara a arquitetura do *kernel* da estação, aceita os seguintes valores: sun4, sun4e, sun4d, sun4c, sun4m ou x86.
- **Memsize**: compara o total de memória principal da estação, em megabytes.
- **Model**: compara o *model_name* para identificar o modelo de estação. A tabela de modelos está descrita em detalhes no Apêndice A.
- **Network**: verifica o endereço de IP da estação.
- **Totaldisk**: verifica o tamanho do disco rígido da estação.
- **Any**: identifica e aceita qualquer estação.

Esta última palavra-chave (any) deverá ser utilizada como última opção de identificação no arquivo de `rules.ok`.

O arquivo `rules.ok` residente no diretório `/jumpstart`, e tem uma linha para cada possível perfil de estação da rede, comparando as palavras-chaves com as características físicas da estação e sua configuração na rede. Um exemplo de rules.ok:

```
# rule keywords and rule value      begin script      profile        finish script
#-----------------------------------  ---------------   --------       ---------------
hostname eng-1                       -                 basic.prof     -
model SUNW, SPARCstation-LX          -                 lx.prof        complete.fin
network 193.144.2.0 && \
  karch i86pc                        scripts/setup.beg x86.prof       scripts/done.fin
any                                  -                 any.prof       -
```

[2] *Slice*: é uma partição do disco rígido. Durante o procedimento de instalação do Unix, um disco rígido deve ser particionado em slices e formatado.

Este arquivo `rules.ok` tem a seguintes regras definidas:

- O exemplo 1 identifica que se a estação tem o nome de *host* eng-1, se afirmativo, o perfil `basic.prof` definira o perfil da estação a ser instalado no procedimento.
- No exemplo 2, a rule compara o modelo da estação e define o perfil `lx.prof` e mais a execução de um *script* após a instalação do perfil.
- O exemplo 3 compara o endereço IP da estação e sua arquitetura de *kernel* para identificar o perfil de instalação como sendo: executar um *script* antes, chamado `setup.beg`; executa o perfil `x86.prof`; e depois executa o *script* final, `done.fin`.
- No exemplo 4, para cada estação que não foi identifica pelos parâmetros anteriores, a *rule* executará o perfil `any.prof`.

Em resumo, quando o administrador de rede define uma *rule* (regra) para uma estação ou grupo de estações, o método **Jumpstart** permite definir um *script* inicial (tipo `.beg`), o arquivo de *profile* (tipo `.prof`) e um arquivo de *script* final (`.fin`), que serão executados nesta seqüência em que foram descritos.

Estes *scripts* de início e término devem ser escritos em *Bourne shell* para que possam ser executados pelo método Jumpstart.

Quando o administrador de rede cria ou altera um arquivo de *rule*, deve validar o mesmo através do programa `check` que verifica a sintaxe e a existência dos *profiles* e *scripts* definidos. Desta validação, o programa `check` gera o arquivo `rules.ok`.

O arquivo de *profile* eng-1 define a forma como será instalado o sistema operacional Solaris na estação, por exemplo: tipo de instalação, tipo de sistema, partições do disco e os pacotes Solaris a serem instalados. Para definir estes parâmetros da instalação utilizam-se palavras-chaves do tipo:

- **System_type**: define o tipo de instalação: *standalone*, *dataless* ou *server*. Na falta desta definição, o método Jumpstart assume como *default* o tipo *standalone*.
- **Client_arch**: quando o system_type for *server*, este parâmetro deve ser definido para a arquitetura a ser utilizada pelo servidor para suportar o boot das estações *diskless* e *dataless*. Os valores aceitos são: sun4, sun4e, sun4d, sun4c, sun4m ou x86.
- **Client_swap**: quando o system_type for do tipo *server*, este parâmetro deve ser definido para alocar espaço de *swap* para cada cliente *diskless*. Na falta desta definição, o método Jumpstart assume como *default* o valor de 24 mbytes.
- **Cluster**: define o tipo de pacote do Solaris que será instalado: *Core*, *End user*, *Developer*, *Entire distribution* e *Entire distribution plus OEM*. Na falta desta definição, o método Jumpstart assume como *default* o tipo *End user*. Este pode definir também a instalação ou remoção de um pacote de software do Solaris, ver descrição detalhada no Apêndice A.

- **Dontuse**: define os discos que não serão usados no procedimento de instalação. Exemplo: `dontuse c0t0d0`.
- **fdisk**: define como serão criadas, alteradas ou mantidas as partições no procedimento de instalação, permitindo definir o nome, tipo e tamanho das partições.
- **Filesys**: define a montagem do sistema de arquivos durante o *boot* da estação, permite definir o servidor, o *path*, os pontos de montagem, e opções para *mount*.
- **Install_type**: define o tipo de instalação: initial_install ou *upgrade*.
- **Locale**: define a língua dos pacotes de software selecionados para a instalação. Na falta desta definição, o método Jumpstart assume como *default* o tipo *English*.
- **Num_clients**: quando o system_type for *server*, este parâmetro deve ser definido como o número de estações do tipo *diskless* que o servidor suportará. Na falta desta definição, o método Jumpstart assume como *default* o valor 5.
- **Package**: define o *package* que será instalado ou removido do *Cluster* já definido.
- **Partitioning**: define como os discos são divididos em *Slices* para o sistema de arquivos durante a instalação; os valores válidos são: *default*, *existing* e *explicit*. Na falta desta definição, o método Jumpstart assume o valor *default*.
- **Usedisk**: define um disco a ser usado pelo software de instalação. Existe a restrição de não poder definir o parâmetro *dontuse* e *usedisk* no mesmo perfil.

Estes parâmetros devem definir a forma como o sistema operacional será instalado.

O arquivo de profile eng-1 criado e armazenado no diretório `/jumpstart` tem uma definição completa de como o sistema operacional será instalado, respeitando o perfil da estação na rede. Porém antes de ser utilizado na instalação, deve ser validado quanto a sua sintaxe e definições através do comando `pfinstall`. Exemplo do *profile* eng-1:

```
Install_type initial_install
System_type standalone
Partitioning existing
Cluster SUNWCuser
```

Este exemplo de *profile* define que a instalação será do tipo `inicial`, preservando as partições existentes, instalar o tipo de sistema para *Standalone* e instalar o pacote de software do Solaris `SUNWCuser`.

O modelo **IC-Jumpstart** baseia-se na metodologia do Jumpstart Solaris mas não utiliza os *script* de início e término, permitidos nos arquivos de `rules.ok`, ou seja, existe o `rules.ok` que identifica o perfil

da estação no procedimento de instalação e existe o arquivo de profile que define a forma de instalação. A seqüência de passos que são executados no modelo IC-Unicamp são:

Passo 1: O administrador recebe uma solicitação para executar um procedimento de instalação.

Passo 2: analisa as alterações necessárias para o método Jumpstart ser utilizado corretamente.

Passo 3: configura os arquivos de *rules* e de *profile*. Este último arquivo pode ser reutilizado de outra instalação similar ou será criado um novo.

Passo 4: realiza o *backup* das informações do(s) usuário(s) da estação (existe uma partição específica na estação que armazena somente as informações dos usuários), minimizando o tempo gasto para realizar o *backup*.

Passo 5: executa o procedimento de instalação do sistema operacional na estação, usando como base as definições feitas no Jumpstart.

Passo 6: configura a estação para o ambiente de rede, por via manual ou através de *script*.

Passo 7: instala os pacotes de aplicativos na estação ou configura o ambiente do Solaris para utilizar os aplicativos via rede.

Passo 8: restaura o *backup* das informações do(s) usuário(s) para a estação.

Passo 9: verifica se os procedimentos foram executados corretamente e comunica aos usuários.

Em resumo, o modelo IC-Unicamp tem a etapa de instalar o sistema operacional automatizada, mas as demais ainda são manuais e com acompanhamento do administrador.

O passo 4 que realiza o *backup* das informações dos usuários que utilizam esta estação é facilitado pelo fato do IC-Unicamp ter implantado um sistema de *backup* automatizado descrito em [22], onde as informações dos usuários são armazenadas em partição especifica, facilitando o processo de *backup*, ou seja, apenas uma partição ou estrutura de diretório é usado no processo de *Backup*.

As desvantagens observadas no modelo IC-Unicamp para o procedimento de instalação do sistema operacional Solaris são menores que o modelo Tradicional, mas são descritas como:

- Execução do procedimento é de forma interativa no passo 6 e 7, e exige um acompanhamento e tratamento manual das mensagens oriundas da execução destes passos.
- Falta de padronização dos passos 6 e 7, não permitir a reutilização dos procedimentos já utilizados.
- Falta de um histórico dos procedimentos executados. Manter um registro de quando o procedimento foi executado, por quem e quais os passos executados.
- Falta documentação dos outros passos executados, como as configurações do Solaris e dos aplicativos instalados.
- Falta de comunicação padronizada com o(s) usuário(s), informando sobre os procedimentos executados e a suas conseqüências.
- Falta de planejamento da execução do procedimento de instalação como um todo.

Em resumo, este método automatiza (passo 5) a tarefa do administrador de rede, mas não resolve todos os problemas.

As vantagens deste modelo IC-Unicamp para a instalação do sistema operacional Solaris via rede são descritas como:

- Documentação do tipo e da forma de instalação do Solaris, registrando as informações de todas as estações da rede.
- Permite a reutilização desta documentação para novas instalações; havendo a coincidência de perfil, o administrador cria uma nova *rule* para a nova estação ou grupo de estações.
- Define uma padronização do passo 6, utilizando o conceito de perfil da estação, o que torna a tarefa de administrar a rede mais simples.
- O planejamento do passo 6 fica mais fácil e simplificado. O administrador tem o registro de uma execução deste passo.

O tempo gasto no procedimento de instalação é linear, com uma média de 3 horas para sua execução. Havendo a necessidade de sua execução em outras estações, exemplo da situação A, o tempo total médio passa de 90 horas, o que representaria quase 4 dias ininterruptos, o que seria inexeqüível. Se considerarmos a hipótese de meio período como o tempo disponível do administrador de rede, este procedimento de instalação seria executado em 22 dias. Este cálculo pode ser representado pelo gráfico de tempo médio de execução abaixo.

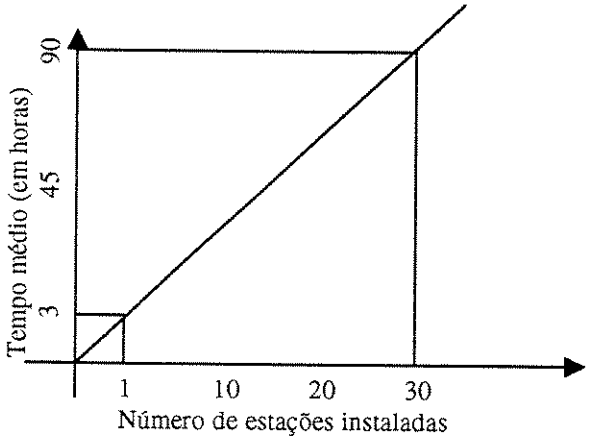

Figura 3.3: Tempo médio de instalação por número de estações no modelo IC-Unicamp.

O tempo médio pode não ser linear se o administrador de rede executar varias instalações em paralelo no passo 6, mas calcular o tempo total a ser gasto nesta modalidade de instalação torna-se impossível devido ao fato de ser um número variável de instalações em paralelo.

3.3 Proposta Custom Jumpstart

Nesta seção será descrito em detalhes o modelo proposto de instalação do sistema operacional Solaris tendo como base o modelo IC-Unicamp, que utiliza a tecnologia Jumpstart criada pela Sun Microsystems.

Este modelo proposto é caracterizado pela completa automatização do procedimento de instalação do sistema operacional Solaris. Esta automatização será alcançada com a utilização do método **Custom Jumpstart** [23], onde os *scripts* de início e término, e uma *rule* no arquivo `rules.ok` realizariam tal automatização.

Analisando os passos executados no modelo IC-Unicamp (seção 3.2): o *Script* de início seria responsável pela automatização do passo 4, e o *script* de término seria responsável pela automatização dos passos 6, 7 e 8.

A automatização do passo 4 (*backup*) será executada, via script *Bourne Shell* [08], utilizando o seguinte algoritmo proposto:

Passo 1: Identificar uma estação na rede para *backup*.

Passo 2: Identificar a partição ou estrutura de diretórios que serão usados no *backup*.

Passo 3: Gravar o *backup* com o nome da estação e mais a data do *backup*.

Passo 4: Verificar se o *backup* está correto e sem erros.

A obrigatoriedade de gravar um *backup* com o nome de host mais a data permitirá que o procedimento de instalação do sistema operacional Solaris seja executado em paralelo, via rede, para várias estações sem a sobreposição de *backup*.

O algoritmo para automatizar o passo 6 (configurar ambiente de rede), via *script Bourne Shell*, utiliza os seguintes passos para a sua execução:

Passo 1: Analisar o perfil da estação.

Passo 2: Identificar todos os serviços de rede que a estação utilizará.

Passo 3: Criar todos os *scripts* de configuração para os serviços de rede, gravando individualmente cada *script*.

Passo 4: Criar um único *script*, com o nome do perfil da estação, que executará os outros *scripts* de configuração.

Passo 5: Verificar e testar os *script* quanto a sua funcionalidade.

A obrigatoriedade de gravar um *script* para cada configuração permitirá sua reutilização por outros procedimentos.

O algoritmo para automatizar o passo 7 (instalar os aplicativos), via script *Bourne Shell*, utiliza os seguintes passos para a sua execução:

Passo 1: Analisar o perfil da estação.

Passo 2: Identificar todos os aplicativos que estão contidos neste perfil de estação.

Passo 3: Criar todos os *scripts* de instalação para os aplicativos, gravando individualmente cada *script*.

Passo 4: Criar um único *script*, com o nome do perfil, que executara a instalação deste perfil, contendo os *scripts* de instalação de todos os aplicativos do perfil.

Passo 5: Verificar e testar os *scripts* quanto a sua funcionalidade.

A obrigatoriedade de gravar um *script* para cada instalação de um aplicativo permitirá sua reutilização por outros procedimentos ou reutilização em outros perfis.

A automatização do passo 8 (restaurar o *backup*) seria executada, via script *Bourne Shell*, utilizando o seguinte algoritmo proposto:

Passo 1: Identificar a estação na rede que armazenou *backup*.

Passo 2: Identificar o arquivo de *backup* da estação.

Passo 3: Restaurar o *backup*.

Passo 4: Verificar se a restauração do *backup* está correto e sem erros.

As vantagens deste modelo **Custom Jumpstart** para o procedimento de instalação do sistema operacional Solaris são:

- Documentação completa de todos os passos executados no procedimento de instalação do sistema operacional Solaris.
- Reutilização da documentação para novas instalações; havendo a coincidência de perfil, o administrador cria uma nova `rule` para a nova estação ou grupo de estações.
- Padronização de todos os procedimentos e tarefas executadas pelo administrador da rede na manutenção do sistema operacional Solaris.
- Padronização da documentação, devido ao conceito de perfil de estação.
- Planejamento de todos os procedimentos e passos a serem executados.
- Monitoramento à distância da execução dos procedimentos através de envio de mensagens e/ou *e-mail* para o administrador de rede, informando o *status* atual da execução.
- A possibilidade de auditoria dos procedimentos executados.
- Execução em paralelo com monitoramento à distância.

Devemos ressaltar que o modelo proposto **Custom Jumpstart** permite dimensionar melhor o tempo a ser gasto em cada procedimento. Para o administrador de rede este modelo obriga o seu planejamento, padronização e documentação dos procedimentos a serem executados.

A única desvantagem deste método proposto é o não registro das informações sobre a execução dos procedimentos de instalação do Solaris. Informações do tipo: quem executou, em que dia, quanto tempo durou etc. Mas este problema será tratado no capítulo 4, seção 4.3 sobre monitoramento do uso de software.

Procedimentos de configuração, atualização e remoção (*patch*)

Os procedimentos de configuração, atualização e remoção para manutenção do sistema operacional Solaris podem ser classificados dentro do modelo **Tradicional** de manutenção, com todas as suas falhas e desvantagens já descritas.

O mesmo modelo de **Custom Jumpstart** se aplicaria para estes procedimentos, onde não se criaria um *profile* para tais tarefas, apenas um *script*, de início ou término, que executaria um procedimento de configuração, atualização ou remoção. É importante usar uma nomenclatura clara e simples para cada *script* criado e gravado.

As mesmas vantagens e a única desvantagem do procedimento de instalação seriam alcançadas com o uso do método **Custom Jumpstart** para os procedimentos, configuração, atualização e remoção, utilizados na manutenção do sistema operacional Solaris.

A partir da versão 2.x, o sistema operacional Solaris foi redefinido quanto a sua forma de manutenção para o tipo *package*, simplificando a construção de *scripts* para manutenção de *patches* do Solaris.

Para cada *patch* do Solaris existe um arquivo `install.info` com as informações detalhadas sobre como instalar e remover. Também existe um arquivo `readme` que contem informações específicas sobre o *patch*. Cada *patch* é identificado por um único nome alfanumérico: um código do *patch*, um hífen, e um número que representa a versão do *patch*. Por exemplo, o *patch* 101977-02 é um *patch* do Solaris 2.4 para atualizar o *daemon* `lockd`.

O administrador de rede obtém os *patches* através de FTP ou *World Wide Web*. Os clientes da Sun Microsystems que optaram pelo contrato de manutenção podem acessar um conjunto de *patches* adicionais e um completo banco de dados de informações sobre *patches*, distribuídos regularmente em cd-rom.

A tabela 3.1 descreve os comandos disponíveis e utilizados pelo administrador de rede para escrever os *scripts* de manutenção do sistema operacional Solaris:

Comandos	Função do comando

patchadd pkgid	Instala o patch pkgid do Solaris na estação.
patchrm pkgid	Remove o patch pkgid do Solaris na estação.
showrev –p	Exibe todos os patches instalados na estação.
pkgparam pkgid PATCHLIST	Exibe todos os patches instalados e com dependência do patch pkgid do Solaris.
pkgparam pkgid PATCH_INFO_patch-number	Exibe a data de instalação e o nome do host onde o patch foi instalado.
patchadd –p	Exibe todos os patches instalados na estação. Equivalente a showrev –p.

Tabela 3.1: Comandos para manutenção de patch do Solaris.

O administrador de rede utiliza o comando `patchadd` para aplicar um novo *patch* na estação. As informações desta atualização, como data, nome do *host* e versão do Solaris, são registradas pelo `patchadd` no arquivo `pkginfo`.

O arquivo `pkginfo` também registra informações sobre os *patches* obsoletos, as dependências entre os *patches*, e as incompatibilidades entre os mesmos.

Durante o processo de instalar ou atualizar um *patch*, o comando `patchadd` cria um arquivo de log no diretório `/var/sadm/patch/patch-number/log` para registrar eventuais erros do processo.

O administrador de rede deve verificar as seguintes condições quando automatizar o procedimento de instalar ou atualizar um *patch* do Solaris, via *script*:

- Se a arquitetura do *patch* é diferente da arquitetura da estação.
- Se o *patch* que está sendo instalado requer outros *patches* não instalados.
- Se o *patch* que está sendo instalado é incompatível com outro *patch*(es) já instalado(s).
- Se o *patch* a ser instalado é uma versão mais velha que a versão instalada, e com a mesma base de código.
- Se as informações registradas em `pkginfo` estão corretas e válidas para os patches instalados.

Em resumo, o administrador de rede deve garantir a qualidade do *script*, para que sua execução seja correta, válida e automatizada.

O administrador de rede utiliza o comando `patchrm` para remover um *patch* do sistema operacional Solaris na estação, restaurando todos os arquivos modificados pelo *patch*. Um arquivo de log registra as informações deste processo no diretório `/tmp/backoutlog.process_id`.

Este procedimento de remoção de *patch* pode não ser executado se o comando *patchrm* verificar as seguintes situações:

- O *patch* foi instalado com patchadd –d , onde o comando não registra em `pkginfo` as informações da instalação ou atualização.
- O *patch* é requerido por outro *patch*.
- O *patch* está obsoleto para um *patch* mais recente.

Para o primeiro caso, o administrador de rede deve reinstalar o *package*, sem a opção –d, para que ocorra o registro das informações do mesmo.

O tempo gasto durante a execução dos procedimentos de manutenção do sistema operacional podem ser reduzidos se configurações do NIS forem alteradas para melhor atender as solicitações do método Custom-Jumpstart conforme [32].

O método Custom-Jumpstart pode ser adaptado para realizar o procedimento de clonar ou o procedimento de snapshot de uma estação conforme [25].

O **Apêndice A** descreve os procedimentos para preparar o ambiente Solaris para executar o método Custom-Jumpstart de forma automatiza: configurando o *NIS*, configurando o servidor de *install client*, o servidor de *boot* e o servidor de *profile*. Aborda também a criação de disquete para a instalação do Solaris em arquitetura Intel e o processo de boot.

No **Apêndice B** estão exemplificados e testados os scripts Bourne Shell para automatizar o método de Custom-Jumpstart para os procedimento de manutenção do Sistema Operacional Solaris do tipo: instalação, configuração, atualização e remoção (patch).

Capítulo 4
Manutenção de aplicativos

Será descrito neste capítulo a manutenção de software na classe de aplicativos para os procedimentos de instalação, configuração, atualização e remoção. As soluções descritas neste capítulo servem para todas as versões de Solaris, independentemente da arquitetura ser Intel ou Sparc [44].

Este capítulo está dividido em três seções que tratam dos problemas relacionados à manutenção de aplicativos no ambiente Solaris: problema de espaço de nomes, problema de controle de versões e problema de monitoramento do uso de software.

4.1 Problema de espaço de nomes

O problema de espaço de nomes, já definido na secção 2.2 do capítulo 2, surge quando o procedimento de instalação de um aplicativo é executado no ambiente Unix instalando o aplicativo no diretório `/usr/local`.

Os problemas gerados são: sobreposição dos arquivos de outro(s) aplicativo(s); dificuldade da identificação dos arquivos que compõem um aplicativo; gerenciamento das várias versões do mesmo aplicativo, para os procedimentos de instalação, configuração, atualização e remoção;

Um exemplo do problema de espaço de nomes: quando se instala o software *perl* e o software *emacs*, os arquivos: `a2p.1`, `ctags.1`, `emacs.1`, `etags.1`, `h2ph.1`, `perl.1` e `s2p.1` são criados no diretório `/usr/loca/man/man1`. A tarefa de manutenção dos aplicativos, feita pelo administrador de rede, torna-se um jogo de memorização, exigindo lembrar que arquivos compõem o *package* de cada aplicativo.

Outro exemplo: uma nova versão do software *emacs* será instalada e ficará em observação até que o administrador de rede tenha a garantia de não haver conflitos com outro(s) aplicativo(s) ou versões do mesmo software.

Por exemplo, a figura 4.1.4a ilustra o caso de duas versões de emacs estáveis e confiáveis (v.20.36 e v.2045) no ambiente EM PRODUCAO e uma nova versão a ser instalada (v.22.01) em /usr/local.

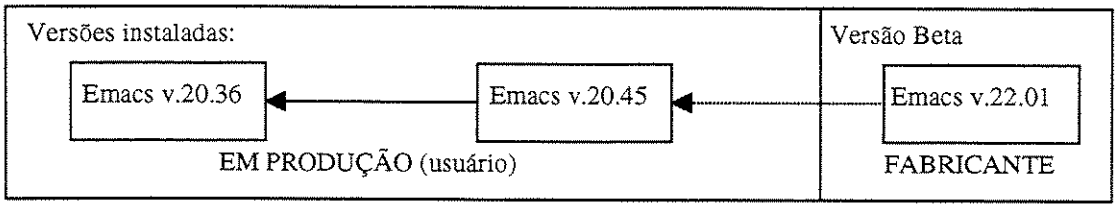

Figura: 4.1.4a: Manutenção de uma nova versão de *emacs*.

Para aplicativos da classe *package* este conflitos são mais simples de serem tratados, porque o *package* registra os arquivos que o compõem e permite redirecionar o aplicativo para outro diretório de instalação. Porém o administrador de rede deve criar um espaço de nomes (estrutura hierárquica de diretórios) que atenda às suas necessidades e resolva os conflitos.

Os aplicativos da classe *scripts* devem ser analisados pelo administrador de rede, descompactando os arquivos, e comparando-os com os aplicativos instalados, verificando manualmente os possíveis conflitos. A complexidade e a dificuldade desta tarefa é maior se existirem somente softwares da classe *script* instalados.

O administrador de rede deve ter conhecimento de todos os passos executados no procedimento de instalação do aplicativo, como foi instalado, onde foi instalado e que arquivos foram instalados. Deve registrar estas informações que serão usadas nos procedimentos de configuração, atualização e remoção deste aplicativo.

Os quatro modelos a serem apresentados a seguir foram agrupados e documentados em função da metodologia utilizada para a manutenção de aplicativos no ambiente Solaris. As formas usadas para coletar tais metodologias de manutenção de software foram entrevistas, e-mail, internet, fóruns de discussões, pesquisas em livros de administração de rede e anais de congressos [01] e [02].

4.1.1 Método Tradicional

Este modelo **Tradicional** é o método mais utilizada pelos administradores de rede para o procedimento de manutenção de aplicativos, devido a sua simplicidade de execução e acompanhamento. Basicamente, o aplicativo é instalado na estrutura de diretório definida pelo fabricante do software e o administrador de rede deve resolver os conflitos manualmente.

Procedimento de instalação

O procedimento de instalação de uma nova versão do aplicativo pode ser descrito através dos seguintes passos no modelo **Tradicional**:

Passo 1: O administrador recebe uma solicitação para executar um procedimento de instalação.

Passo 2: analisa as alterações necessárias para evitar os conflitos com outro(s) software ou outra(s) versão(ões) do mesmo aplicativo.

Passo 3: executa um *backup* do ambiente atual da estação Solaris.

Passo 4: executa o procedimento de instalação do aplicativo, respeitando o diretório de instalação definido pelo fabricante.

Passo 5: verifica os conflitos gerados e corrige-os.

Passo 6: verifica todos os aplicativos instalados, quanto a funcionalidade e execução.

Passo 7: disponibiliza a nova versão para os usuários, informando-os sobre a nova versão e como configurá-la.

Em resumo, este método se caracteriza por ser manual e empírico, não havendo ferramentas que auxiliam o administrador de rede para solução dos conflitos.

As soluções utilizadas pelo administrador de rede para resolver os conflitos gerados pelo procedimento de instalação são:

- Instalar a nova versão em outro diretório, ou mesmo, em outra estação.
- Remover as versões mais antigas do aplicativo.
- Mover os arquivos conflitantes para diretórios distintos, e criar *links* simbólicos para os mesmos.

Em alguns casos, quando os conflitos não são solucionados rapidamente, o administrador de rede restaura o *backup* do Passo 3, e disponibiliza a estação para uso, assim pode estudar mais profundamente uma solução para os conflitos.

Para os aplicativos da categoria *package*, sua instalação em outro diretório é automática, baseando-se no parâmetro do novo diretório fornecido pelo administrador da rede. Para a categoria *Script*, o administrador de rede pode verificar os *scripts* de instalação e providenciar as devidas alterações, ou obter os fontes e recompilar para que seja instalado no diretório correto, conforme descrito na seção 2.2 do capítulo 2.

As desvantagens encontradas no método **Tradicional** para o procedimento de instalação de um aplicativo podem ser resumidas como:

- Não existe um espaço de nomes (estrutura de diretório) que separe os aplicativos em categorias ou versões.
- Não existe uma documentação do procedimento de instalação.

- Na classe *Script* de aplicativos, não existe um registro ou histórico dos aplicativos instalados.
- As soluções encontradas são empíricas, não permitindo sua automatização e posterior reutilização.
- O ritmo de trabalho é ditado pelos conflitos que vão surgindo, por que o administrador de rede não possui nenhuma ferramenta para ajudar.
- Identificar os usuários de uma versão do aplicativo para efetuar a comunicação das alterações realizadas.

Em resumo, por ser o método mais simples é o que contém as maiores falhas.

Procedimento de configuração, atualização e remoção.

Conforme a definição do espaço de nome adotada pelo administrador de rede neste procedimento de instalação do método **Tradicional**, os procedimentos de configuração, atualização e remoção de aplicativos têm os mesmos problemas citados quanto à instalação do aplicativo.

Devido à forma como a classe *package* é criada e definida, os aplicativos que a compõem são mais fáceis de serem administrados, mas o administrador de rede deve criar um espaço de nome (estrutura de diretório) para manter as várias versões do mesmo software instaladas e sem conflitos.

4.1.2. Método IC-Unicamp

O modelo IC-Unicamp foi criado para facilitar a tarefa de administrar a manutenção de aplicativos. Baseia-se na criação de um espaço de nomes que separa os aplicativos em categorias, ou seja, existe uma hierarquia de diretórios para as várias categorias de aplicativos. Por exemplo:

- `/n/net` : para os aplicativos relacionados à rede, como: `mosaic/WWW`, `gopher`, `ftp`, utilitários DNS e outros.
- `/n/dtp` : para os aplicativos do tipo: `framemaker`, `TeX`, `dvi`, `previewers OS` e outros.
- `/n/draw` : com os aplicativos `AutoCAD`, `xfig` e outros pacotes gráficos.
- `/n/gnu` : para os aplicativos da Gnu software: `GC++`, `Gpascal`, `Gfortran` e outros.
- `/n/db` : para os aplicativos do tipo banco de dados: `mysql`, `sybase`, `oracle` e outros.

Dentro de cada categoria de aplicativos, existe uma estrutura de diretórios necessários para sua correta execução, do tipo: `./bin`, `./etc`, `./man`, `./lib` e outros.

A nomenclatura /n, ou ainda /network, define que os aplicativos estão instalados e configurados para todos os usuários da rede. Existem outras nomenclaturas e estruturas de diretórios: /l , /h e /p.

A estrutura de /l significa uma partição do disco local (*local file system*), onde os arquivos de usuários são armazenados, ou mesmo, repositórios de uso geral. Esta estrutura facilita e simplifica o processo de *backup* da rede. Permite também a subdivisão em outros diretórios, como: /l/repXX para repositórios, ou /l/homeXX e /l/projXX.

A estrutura /h, ou ainda /home, é usada para a estrutura de home dos usuários, com suas subhierarquias do tipo /h/staff, /h/phd e /h/grad, separando os tipos de usuários por categorias e funcionalidades dentro do IC-Unicamp.

Para a estrutura /p, ou ainda /project, ocorre a classificação dos vários projetos do IC-Unicamp em atividade e/ou os subprojetos derivados.

Para os usuários, tal espaço de nomes é transparente, e são disponibilizados através de *automount* do NFS [42], onde uma variável de ambiente identifica o tipo de sistema operacional Solaris da estação e monta a hierarquia de diretórios válida. Por exemplo, em /l/rep00/Gnu existe a hierarquia solaris2/ , sunos4/, linux/, solarisX86/ e irix/, com os respectivos binários para sua execução correta. E dentro da hierarquia /l/rep00/Gnu/solaris2 tem os seguintes diretórios bin/, etc/, lisp/, info/ e outros.

Os modelos propostos por Paul Anderon em [05] e por Bjorn Satdeva em [31] motivarão a criação do modelo IC-Unicamp, porém com procedimentos de manutenção de software mais simples em complexidade.

Em resumo, esta hierarquização facilita os procedimentos de manutenção de software permitindo a separação física dos softwares em categorias, e disponibiliza a correta versão de software conforme a versão do sistema operacional e arquitetura de hardware.

Procedimento de instalação

No modelo IC-Unicamp, o procedimento de instalação de uma nova versão do aplicativo pode ser descrito através dos seguintes passos:

Passo 1: O administrador recebe uma solicitação para executar um procedimento de instalação.

Passo 2: analisa o espaço de nomes e as alterações necessárias para evitar os conflitos com outro(s) software ou outra(s) versão(ões) do mesmo aplicativo.

Passo 3: executa um *backup* do ambiente atual da estação Solaris.

Passo 4: executa o procedimento de instalação do aplicativo, fornecendo o diretório de instalação dentro do espaço de nomes ativo.

Passo 5: verifica os conflitos gerados e corrige-os.

Passo 6: verifica todos os aplicativos instalados, quanto a funcionalidade e execução.

Passo 7: disponibiliza a nova versão para os usuários, informando-os sobre a nova versão e como configurá-la.

Em resumo, este método se caracterizar por facilitar o procedimento de instalação, separando os aplicativos em categorias.

As soluções utilizadas pelo administrador de rede para resolver os conflitos gerados pelo procedimento de instalação são:

- configurar o ambiente Solaris para a nova versão do software, através da alteração do *path* no arquivo de configuração de *shell* do(s) usuário(s) (`.login`).
- Remover as versões mais antigas do aplicativo, eliminado o conflito com esta versão sendo instalada.
- Mover os arquivos conflitantes para diretórios distintos, respeitando as versões, e criar *links* simbólicos para os diretórios.

Em alguns casos, quando os conflitos não são solucionados rapidamente, o administrador de rede restaura o *backup* do passo 3, e disponibiliza a hierarquia de diretório (espaço de nomes) para utilização dos usuários.

As vantagens do método IC-Unicamp para resolver o problema de espaço de nomes são descritas como:

- Categorias de softwares: os softwares são separados em categorias conforme sua funcionalidade. O administrador de rede tem a liberdade de criar as categorias que necessita para separar os softwares que administra.
- Redução dos conflitos: apenas dentro da mesma categoria de software, por exemplo: os softwares da categoria `/n/gnu` têm os mesmos procedimentos de instalação definidos pelo fabricante.
- Documentação do procedimento: o administrador de rede tem conhecimento prévio da localização dos aplicativos. Uma documentação do procedimento de instalação é gerada para cada versão do aplicativo instalado, o diretório destino, quando foi instalado e quais os arquivos que compõem tal software.
- Antecipação dos conflitos: com o espaço de nomes definido e ativo, os conflitos gerados são restritos à mesma categoria de software.
- Disponibilidade de software para os usuários: dentro da categoria o usuário tem a liberdade de analisar todos os aplicativos instalados, e escolher o melhor software que atende suas necessidades.

Em resumo, este método apresenta algumas vantagens em relação ao método Tradicional.

As desvantagens do método IC-Unicamp para resolver o problema de espaço de nomes são descritas como:

- Não ocorre a separação física dos aplicativos dentro da mesma categoria. O administrador de rede deve analisar os procedimentos de instalação e remoção de um aplicativo para resolver os conflitos com as várias versões instaladas.
- Não ocorre a separação física das versões de um aplicativo: os conflitos entre as várias versões do mesmo aplicativo são reais e exigem do administrador de rede solução para estes conflitos.
- As soluções encontradas são empíricas, não permitindo sua automatização e posterior reutilização. O administrador de rede deve analisar os conflitos dentro da categoria do aplicativo.
- O ritmo de trabalho continua sendo ditado pelos conflitos que vão surgindo, devido ao fato do administrador de rede não possuir nenhuma ferramenta para ajudar.
- Continua não sendo possível a identificação dos usuários de uma versão do aplicativo, visando à comunicação das alterações realizadas.

O principal problema deste método ocorre na não separação da versões de um aplicativo.

4.1.3. Método Andrew

O Sistema Operacional Andrew [17] começou como um projeto de pesquisa [30] que amadureceu no curso de Ciência da Computação na universidade `Carnegie Mellon`. Durante este processo, seus projetistas encontraram os mesmos problemas de administração de software do padrão Unix em `/usr/local` sobre o sistema de arquivo *Andrew File System*, ou AFS [12], mas totalmente compatível com o ambiente Unix.

Esta seção descreve a forma e estrutura de administrar a classe de aplicativos dentro do sistema operacional Andrew:

Cada categoria ou grupo de aplicações tem sua própria *collection*. O nome da *collection* deve ser único, e será associado permanentemente com o software. Por exemplo, `gnu-emacs` e `batmail` são nomes de *collection*. O nome da *collection* tem que ter 9 caracteres ou menos e não pode conter nenhum caractere especial.

Pode haver várias *versions* para uma *collection* do aplicativo. Cada versão é determinada com três dígitos numéricos. A primeira versão é 001, a segunda é 002, e assim sucessivamente.

Por razões administrativas de espaço, o número de versões que são mantidas para uma *collection* é limitado. Para os softwares que sofrem a manutenção feita pelo administrador da rede, podem ter no máximo quatro versões; para os softwares de usuários podem ter no máximo três versões.

Cada *collection* de software é parte de uma *tree*. Há três árvores atualmente suportadas no ambiente Andrew: *local*, *contributed* e *host*. A estrutura *local* contém software desenvolvido ou mantido pelo administrador de rede, e *contributed* contém software desenvolvido e mantido por usuários. A árvore *host* contém software mantido pelo administrador de rede, que é usado pelo sistema operacional Andrew da estação.

Dentro de cada árvore, vários tipos de *system* são mantidos. Uma aplicação pode ser compilada para mais de um hardware suportado no ambiente Andrew [12]. Por exemplo, sun4411 e pmax_42 são dois tipos de arquitetura de hardware diferentes.

O espaço de nomes criado no ambiente Andrew [17] tem a seguinte estrutura:

```
/afs/andrew/<system>/<tree>/<collection>/<version>
```

Quando um software é produzido, deve ser disponibilizado para os ambientes: *alpha*, *beta* e *gama*. O ambiente *alpha* é o ambiente local de uma estação em `/usr/local`, que é utilizado por um usuário para desenvolver software. Quando se torna estável, este é disponibilizado para o ambiente *beta*, que será testado pelo administrador de rede, quanto a sua funcionalidade, confiabilidade e estabilidade para o ambiente Andrew [12]. Quando aprovado pelo administrador da rede, ou seja, configurado, validado e resolvidos todos os conflitos de execução com todos os softwares instalados no ambiente Andrew, o software será disponibilizado para o ambiente *gama*, que é o ambiente de produção dos demais usuários. Só existe uma versão da aplicação para cada ambiente.

Ambiente de desenvolvimento de software

As áreas fixadas para o desenvolvimento de software estão na forma de volumes do AFS [37]. Estes volumes são áreas de disco, que são criadas e montadas como um diretório no *Andrew File System*, ou AFS. Para os usuários, os volumes são vistos como diretórios. Porém, os volumes podem ter mais de um *pathname* e têm uma quota fixa de espaço de armazenamento. Um exemplo de um volume é o diretório de *home*. O diretório *home* de um usuário é um volume montado sobre o AFS com o seu *userID* [37].

O administrador de rede ou desenvolvedor do aplicativo define quatro volumes para um software: *source*, *dest*, *common* e *object*.

A área de *source* armazena o código fonte do software. Por exemplo:

```
/afs/andrew/system/src/local/gnu-emacs/001
```

O administrador de rede pode organizar os conteúdos de seus volumes source de forma conveniente, contanto que tudo dentro do volume esteja relacionado à sua *collection.*

A área de *dest* armazena as versões de binários para o ambiente Andrew, também chamados de *releases volumes*. Deve ser criado um volume *dest* para cada tipo de binário no ambiente Andrew onde o aplicativo será executado. O administrador da rede deve compilar os fontes do aplicativo para gerar os binários necessários. Por exemplo:

```
/afs/andrew/system/dest/pmax_42/local/gnu-emacs/001
```

Quando o usuário, que desenvolveu um aplicativo, tem uma quantidade significativa de informação que é compartilhada entre os tipos de sistemas que ele dá suporte, o usuário pode requisitar um volume *common*. Este é o único volume que contém arquivos do aplicativo que não são dependentes do ambiente Andrew, por exemplo, os arquivos de help (man), que são montados dentro de cada volumes *dest*. Por exemplo:

```
/afs/andrew/system/dest/pmax_42/local/gnu-emacs/001/common
```

A área *object* armazena os objetos e executáveis, quando o usuário compila sua aplicação. Por *default*, esta área estará em seu disco local, em:

```
/usr/obj/local/gnu-emacs/001
```

O administrador de rede tem dois tipos de grupo de usuários possíveis, os *commanders* ou os *visitors*, que utilizarão a estrutura de arquivo do ambiente Andrew. O *commander* tem permissão de escrita nos aplicativos, e o *visitor* apenas permissão de leitura.

Ferramenta de manutenção de ambiente

O software `Environment Maintenance Tool`, ou EMT, é uma aplicação usada para administrar as *collections* sobre o ambiente AFS para os procedimentos de instalar, remover ou atualizar um software. Este software garante que o espaço de nomes (ou estrutura de diretórios) no ambiente Andrew esteja correto e válido, assim como os volumes e *links* simbólicos.

Os *Gatekeepers*, ou administrador(es) de rede(s), são encarregados das tarefas de administrar o

ambiente Andrew, com acesso a todas as funcionalidades de EMT dentro de uma determinada árvore. Pelo programa CARPE [11], o usuário envia requisições ao *Gatekeeper* para montar e configurar sua(s) *collection*(s), ou remover *collection*(s), ou mesmo atualizar seu(s) software(s) para *Beta* ou *Gama*. Todas as requisições são enviadas por `bboards` eletrônicos. O usuário deverá monitorar os retornos de confirmação, através dos `bboards`, para assegurar que foram recebidos e processados.

Para requisitar ao *Gatekeeper* que crie sua *collection*, o usuário deve prover as seguintes informações:

- O nome da árvore que o seu software pertence. Por exemplo, *local* ou *contributed*.
- O nome da coleção da sua aplicação. Por exemplo, gnu-emacs.
- O número da versão que o usuário esta utilizando. A versão inicia com 001, e é incrementada consecutivamente para cada nova versão do aplicativo.
- O ambiente Andrew do aplicativo ou tipo de binário para os quais o software será compilado.
- A quantidade de espaço em disco que será utilizada.
- Os usuários que pertencem ao grupo *commanders*.
- O tipo de volume a ser criado: *source*, *object* ou *common*.
- Executar o procedimento de copiar o código fonte da versão atual para uma nova versão.

O usuário fica responsável pela manutenção local desta *tree* e como configurar a sua quota de espaço.

Em resumo, o usuário utiliza o aplicativo `Carpe` [11] para se comunicar com o *Gatekeeper*, que utilizará o aplicativo EMT para realizar as tarefas de manutenção no espaço de nomes do ambiente Andrew. Uma solicitação de criar uma coleção, normalmente, é executada em um a dois dias, mas se for uma coleção *contributed*, pode levar aproximadamente uma semana.

O aplicativo EMT é normalmente executado de forma interativa, mas pode ser parametrizado para uma execução automática.

Para o procedimento de remoção de uma *collection*, o programa `Carpe` pedirá os números de versões e os tipos de sistemas suportados que se deseja remover e pedirá uma confirmação para cada remoção. O usuário pode remover mais de uma versão por requisição.

Banco de dados

Todas as manutenções de software realizadas no ambiente Andrew são analisadas, verificadas e gravadas em um banco de dados. Estas informações estão disponíveis para o pessoal de *staff* através de um conjunto de programa `ts`, que permite pesquisas do tipo: quando uma *collection* foi atualizada a uma *tree* ou *system* (binário); ou para informar, quais *collections* foram atualizadas no ambiente *beta* e/ou *gama*.

As opções disponíveis para pesquisar o banco de dados podem ser resumidas através dos programas:

Programas	Função executada
ts_chkst	Verifica o estado de uma *tree*, ambiente ou *system* em um determinado instante.
ts_changes	Informa as últimas atualizações em uma *tree*, ambiente ou *system*.
ts_differ	Compara as diferenças entre o ambiente *beta* e *gama* para uma determinada *tree* e system.
ts_since	Quais as versões de uma *collection* que foram atualizadas em uma árvore, ambiente ou *system*, desde uma data especifica.

As informações registradas neste banco de dados são um histórico de toda manutenção de aplicativos realizada no ambiente Andrew e será descrita em detalhes na seção "Como Depot organiza os softwares" deste capítulo.

Compilando e instalando software

O programa `Makefile` deverá criar a estrutura de diretórios apropriada para o volume *dest*. As `man pages` deve ser copiadas para o subdiretório `man` apropriado; os arquivos de ajuda devem ir para o subdiretório `help`, bibliotecas em `lib`, e a imagem do executável em `bin`.

Para que o software seja instalado corretamente no volume *dest* deverá ser usada a variável `DESTDIR` para especificar o volume *dest*. Um exemplo de `Makefile`:

```
install: myprogram myprogram.man libmy.a myprogram.help
-mkdir $(DESTDIR)
-mkdir $(DESTDIR)/man
-mkdir $(DESTDIR)/man/man1
-mkdir $(DESTDIR)/bin
-mkdir $(DESTDIR)/lib
-mkdir $(DESTDIR)/help
install -s myprogram $(DESTDIR)/bin
install <lib> .a $(DESTDIR)/lib
install -m 644 myprogram.man $(DESTDIR/man/man1 / myprogram.1
install -m 644 myprogram.help (DESTDIR)/help
```

Quando instalar binários deve-se usar a opção -s em *install*. Isto descreve como os arquivos serão instalados e economizam espaço no sistema.

Quando estiver instalando arquivos não-executáveis como `text`, `help`, ou `man pages`, a opção "-m 644" configura os bits de proteção adequadamente durante a instalação.

O programa `build` será executado na estação do mesmo tipo de *system*, processando o `Makefile`, em vez de deixar o `Make` fazê-lo. O programa `build` tem a mesma funcionalidade de `Make` com duas exceções:

- Compila o fonte da área *object*.
- Configura automaticamente várias variáveis que podem ser usadas em seu `Makefile`, como a variável `DESTDIR`.

Antes de compilar o software, deve-se ter certeza que os *pathnames* de destino do software estão corretos e não são os *pathnames* da área de desenvolvimento.

Para testar o aplicativo compilado, com acesso de *root* para uma estação, o usuário pode atualizar sua aplicação usando `Local Disk Depot`, antes de atualizar o ambiente *Beta*.

Atualizando seu Software

Quando terminar de compilar e testar seu software, o usuário pode atualizá-lo para os seguintes ambientes no Andrew:

- *Beta*, o qual permite o pessoal de *staff* testar e reportar os bugs.
- *Gama*, o qual atualiza seu software para todos os *systems* disponível em todas as estações.

Este software deve ser estável, documentado e confiável antes de ser atualizado para *Gama*.

Para software em local, as atualizações para o ambiente *beta* levam normalmente um dia de trabalho, e atualizações para *gama* normalmente levam três dias de trabalho. Os softwares em *local* devem estar em ambiente *beta* durante pelo menos três dias antes que possam ser atualizados para o ambiente *gama*. Para software em *contributed*, as atualizações podem levar vários dias.

No ambiente *beta*, é necessário remover a *version* mais recentemente lançada de seu software para corrigir bug(s). Para fazer isto, deve-se requisitar a remoção desta *version* do software. Entretanto, se esta *version* está danificada, o desenvolvedor deve corrigir o(s) bug(s), compilar, testar e requisitar uma nova *version*.

Quando o programa EMT recebe uma requisição para atualizar um software, as seguintes tarefas são realizadas:

- Montar o seu volume para o ambiente apropriado.
- Configurar as proteções de forma que o usuário não possa escrever em arquivos do sistema, e sua área seja protegida de acessos indevidos.
- Executar o programa `Depot` [14] para criar uma série de *links* simbólicos, permitindo que os softwares sejam executados sem conflitos.
- Atualiza o software para que a comunidade do campus possa usá-lo.

As seções seguintes descrevem como opera o programa Depot [14] dentro do ambiente Andrew para executar os procedimentos de manutenção de software [13].

Como Depot organiza os softwares

O programa Depot [20] é uma ferramenta que administra ambientes de software integrando múltiplas *collections* independentes em uma única hierarquia de diretório, ou espaço de nomes.

Para cada execução, o programa depot processa um único ambiente de software. O ambiente de software começa com uma hierarquia de diretório especificada, incorporando todos os arquivos deste diretório, inclusive os subdiretórios.

Depot mantém cada collection em uma estrutura de diretório diferente, permitindo que várias *collections* sejam mantidas na mesma hierarquia. Mas o programa Depot falhará antes de uma atualização se existir um conflito entre as *collections*.

A estrutura de diretório do ambiente Andrew [17] pode ser configurada por um conjunto de opções. Estas opções determinam quais *collections* serão ativas e como serão integradas no ambiente. Conflitos entre as *collections* podem ser solucionados especificando qual *collection* ignora a outra ou usando opções de configuração para mover arquivos ou diretórios da hierarquia.

Existem duas formas de uma *collection* ser integrada ao ambiente: através de cópia ou de *link* simbólico. Para as *collections* que possuem de *links* simbólicos, os links são feitos do ambiente de execução para a localização onde reside a *collection*. Para reduzir o número de *links*, eles são feitos no nível dos diretórios, quando possível. Com a opção de cópia, todos os arquivos e diretórios são copiados no ambiente designado, assim como os *links* internos da *collection*.

Os arquivos de índices são mantidos freqüentemente em seus diretórios. Por exemplo, o arquivo fonts.dir no diretório de fontes do X11. Através de uma opção de configuração, estes arquivos de índices são atualizados de um modo automatizado. Podem ser executados sempre que uma *collection* ou diretório for alterado.

Antes de integrar as *collections* em um ambiente, depot verifica se o ambiente é consistente. Não deve haver nenhum conflito no ambiente. Qualquer conflito encontrado resultará no término da execução do programa depot. Os arquivos que não pertencem a uma coleção ou que não estão marcados como arquivos especiais nos arquivos de configuração, são apagados pelo depot, que em seguida verifica os *paths* da *collection*. Após estas verificações, todas as novas *collections* serão inseridas ao ambiente e as alterações produzidas em outras *collections* serão registradas no banco de dados depot.db.

O processo de integração pode ser feito no sistema de arquivos compartilhado ou no disco local da estação do usuário. Freqüentemente, o ambiente construído na estação do cliente depende de *collections* do

sistema de arquivo compartilhado. Quando um software é atualizado no sistema de arquivo compartilhado, o `depot` deve ser executado na estação para atualizar esta nova configuração, mantendo o ambiente consistente e funcional. Esta execução pode ser iniciada pelo usuário ou pelo EMT no servidor, gerando um efeito em cascata.

Quando uma *version* é integrada ao ambiente, `depot` configura o ambiente para mantê-lo estável e consistente. Um número razoável de versions são mantidas, mas nenhuma *collection* será removida antes que uma estação tenha a oportunidade de executar o `depot`.

Integrando múltiplas *collections* independentes em um único ambiente, o programa `depot` alcança independência e integração. Com o espaço de nomes (estrutura hierárquica), os números de *versions* e as estratégias de atualização diferentes, projeta uma mobilidade que permite a integração de novas *versions* ou de diferentes pacotes de software em localizações diferentes.

A manutenção de software realizada pelo `depot` está limitada ao ambiente de `/usr/local`. Para os softwares ou arquivos que precisam ser copiados nas áreas do sistema operacional Andrew, será usado outro programa.

Nesta versão, o programa `depot` só opera no nível de *collection*. Não há modo de especificar arquivos individuais ou diretórios a serem copiados: ou as *collections* inteiras foram copiadas ou as *collections* inteiras foram linkadas. Esta regra, não permitindo as operações sobre os arquivos, mostrou ser muito rígida. Opções específicas para permitir copiar arquivos individuais ou diretórios, como *links* e remoções, foram incluídas na *collection* de origem. Por exemplo, várias *collections* podem instalar fonts no diretório `lib/X11/fonts`, e o administrador de rede pode desejar copiar tais arquivos sem copiar a *collection* de origem. Semelhantemente, o administrador de rede pode escolher unir todos os arquivos integrados no diretório man com os arquivos do diretório `doc` para reduzir espaço.

Por permitir que um arquivo específico seja movido da *collection*, mapeando-o em uma outra *collection*, e não modificando a estrutura hierárquica, preservando os arquivos da *collection* e a estrutura do ambiente Andrew, torna a sua administração mais flexível.

`Depot` mostrou-se ineficiente para tratar de ambientes muito grandes. Consome-se muito tempo para pesquisar o banco de dados quando ocorre uma inclusão de uma nova *collection*. Algumas melhorias foram feitas para acelerar a verificação de collections dentro de volumes AFS [37], mas não foram suficientes. Uma completa redefinição e customização do banco de dados foi planejada pelos responsáveis do sistema operacional Andrew.

Algumas ferramentas adicionais são necessárias para a distribuição e localização de informações no ambiente atual, conforme Mark Held em [17], que descreve algumas das outras ferramentas que são usadas com o programa `depot` [13] para administrar o ambiente Andrew de software.

As vantagens do método Andrew para resolver o problema de espaço de nomes na manutenção de software são descritas como:

- Os softwares são separados em ambientes: *system*, *tree*, *collection* e *version*. Esta estrutura hierárquica de diretório atende as necessidades do administrador da rede.
- O sistema permite registrar até 999 versões para um aplicativo, mas na prática não passam de três.
- Documentação dos procedimentos: todos os procedimentos de instalação, configuração, atualização e remoção de software são registrados no banco de dados `depot.db`, permitindo que o administrador de rede analise, verifique e consulte estas informações.
- Antecipação dos conflitos: o administrador da rede verifica os possíveis conflitos na manutenção de uma nova versão do software antecipadamente.
- Padronização das atividades de desenvolvimento, teste e validação dos softwares no ambiente Andrew.
- Os procedimentos são parametrizados para sua execução automática, sem a interação com o administrador da rede.
- Para os softwares da classe *script*, sua manutenção é resolvida através de `Makefile` e `Build`.
- As tarefas de administração de software são planejadas e dimensionadas no tempo.

Em resumo, este método apresenta muitas vantagens em relação ao método IC-Unicamp.

As desvantagens do método Andrew para resolver o problema de espaço de nomes são descritas como:

- O tempo gasto para executar as manutenções de software são de dias, o que torna as tarefas de administração muito demoradas.
- Continua não sendo possível a identificação dos usuários de uma versão do aplicativo, visando à comunicação das alterações realizadas.
- O programa `depot` só trata os softwares instalados no diretório `/usr/local`.
- O usuário deve configurar seu ambiente de *Shell* (`.login`) para ter acesso às novas versões do aplicativo no ambiente Andrew.
- O banco de dados `depot.db` demonstrou ser lento para as atualizações e consultas.
- Com um banco de dados centralizado, sua eventual indisponibilidade pode inviabilizar as manutenções de software.

O principal problema deste método ocorre no tempo gasto na manutenção dos softwares e na efetiva disponibilização para os usuários. Melhorias para o desempenho do ambiente Andrew foram propostas para alterar o sistema de arquivo do Andrew (AFS) em [38] e na flexibilização das rotinas do AFS em [50].

O `Local disk Depot` é uma extensão do programa Depot do ambiente Andrew. Criando um no disco local da estação a estrutura e configurações necessários para os aplicativos serem executados conforme [33].

Outra extensão é `Depot-Lite` que permite que os próprios usuários optem pelo tipo de instalação do software: *module* ou *colletion*. Também permite a criação de tree do tipo Skeleton que utiliza menos automount para os aplicativos, conforme [34].

4.1.4 Método Stow-Gnu

O método **Stow-Gnu** [40] foi desenvolvido a partir do método Andrew. Tem a mesma funcionalidade do programa `depot`. Caracteriza-se por ser uma ferramenta para administrar a manutenção de múltiplos softwares empacotados na mesma árvore de diretório em tempo de execução. Porém é substancialmente mais simples e mais seguro.

O método Andrew, através do programa `depot`, possui um banco de dados para registrar as informações de cada pacote de software, localização, tamanho, conflitos, *links* relativos, *links* absolutos e outros dados, sendo que tais informações são utilizadas para fazer a manutenção no ambiente Andrew. O método Stow-Gnu não possui um banco de dados para registrar tais informações; quando é executado, o programa `stow` (de preparação do ambiente) faz uma análise dos softwares instalados, através do espaço de nomes e *links* relativos, e define as ações a serem tomadas sobre os *links* simbólicos. Nesta situação, não há nenhum perigo de deixar de tomar uma decisão porque o banco de dados não está disponível ou deixou de registrar alguma informação importante.

Inicialmente o administrador deve instalar o aplicativo em uma localização específica determinada pelo próprio administrador, segundo sua política própria. A instalação deve ser feita para que o aplicativo possa ser executado a partir desta localização. O mecanismo usado pelo programa `stow` é o de copiar a hierarquia do aplicativo para debaixo da hierarquia própria do `stow` [40]; em seguida deve analisar o *package* do aplicativo (software), identificando os arquivos executáveis na hierarquia original, removendo-os e criando *links* simbólicos apontando para os executáveis debaixo da hierarquia do próprio `stow`. Normalmente o diretório do stow é criado como `/usr/local/stow`, embora o administrador da rede tenha a liberdade de escolher outra localização qualquer. O programa `stow` poderá criar (quando possível) um único *link* simbólico de um diretório para o correspondente na nova estrutura hierárquica, evitando assim criar vários *links* simbólicos para cada arquivo deste diretório.

Quando, por exemplo, o aplicativo `Perl` for instalado e o administrador escolher `/usr/local/perl`, seu *package* utilizará o subdiretório `bin` para os executáveis `perl` e `a2p`; o diretório

`info` para a documentação em `Texinfo`; o diretório `lib/perl` contendo as `libraries perl`; e o diretório `man/man1` para as `man pages` do aplicativo *perl*, conforme a figura 4.1.4b. Notar que esta já não é uma instalação convencional, pois a maioria dos pacotes fazem sua instalação em `/usr/local`, misturando seus arquivos com os dos demais pacotes instalados.

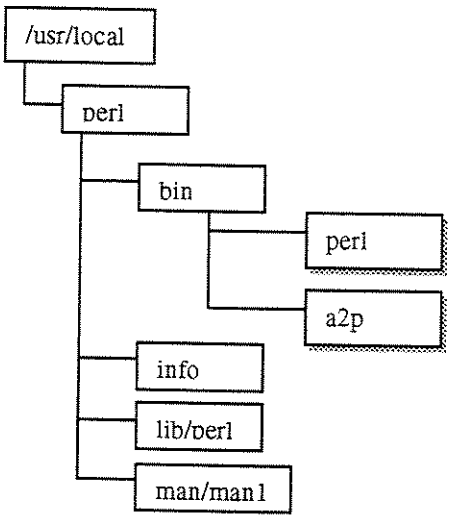

Figura 4.1.4b: Estrutura de diretório do aplicativo *perl*.

Executando o programa `stow` para o aplicativo `perl`, a hierárquica original do *perl* será copiada para `/usr/local/stow`; desta forma, uma nova hierarquia completa aparecerá sob `/usr/local/stow/perl`, conforme a figura 4.1.4c.

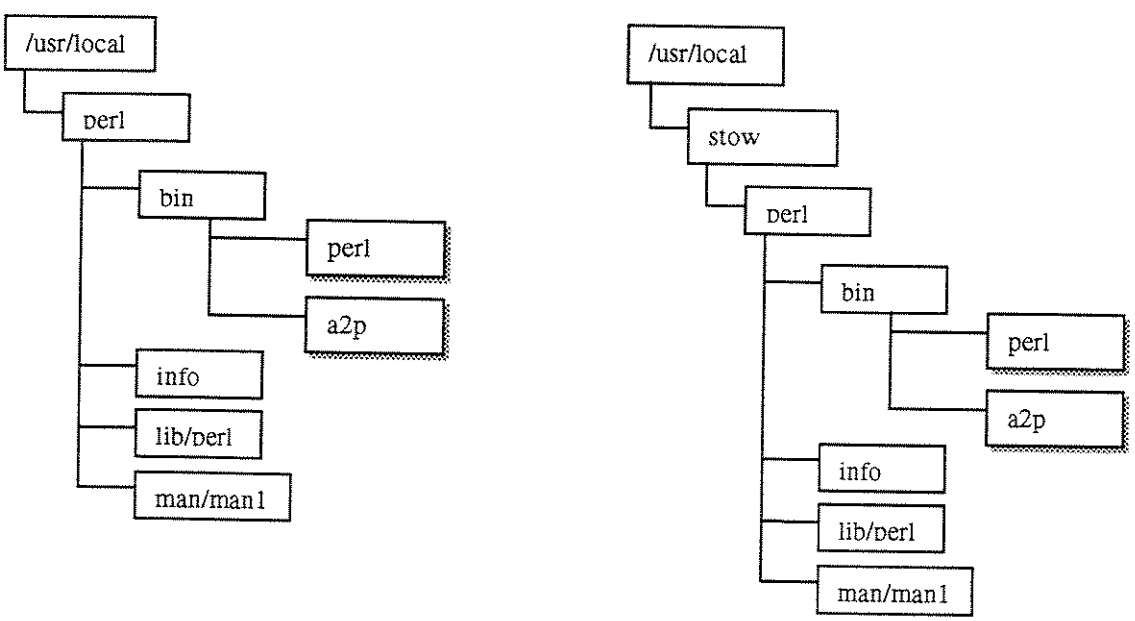

Figura 4.1.4c: Estrutura de diretório do aplicativo perl após execução do `stow`.

O programa `stow` cria todos os *links* simbólicos necessários para aplicativos `perl`, por exemplo: os *links* simbólicos dos executáveis `perl` e `a2p` do diretório `/usr/local/perl/bin` para o diretório `/usr/local/stow/perl/bin`, conforme figura 4.1.4d.

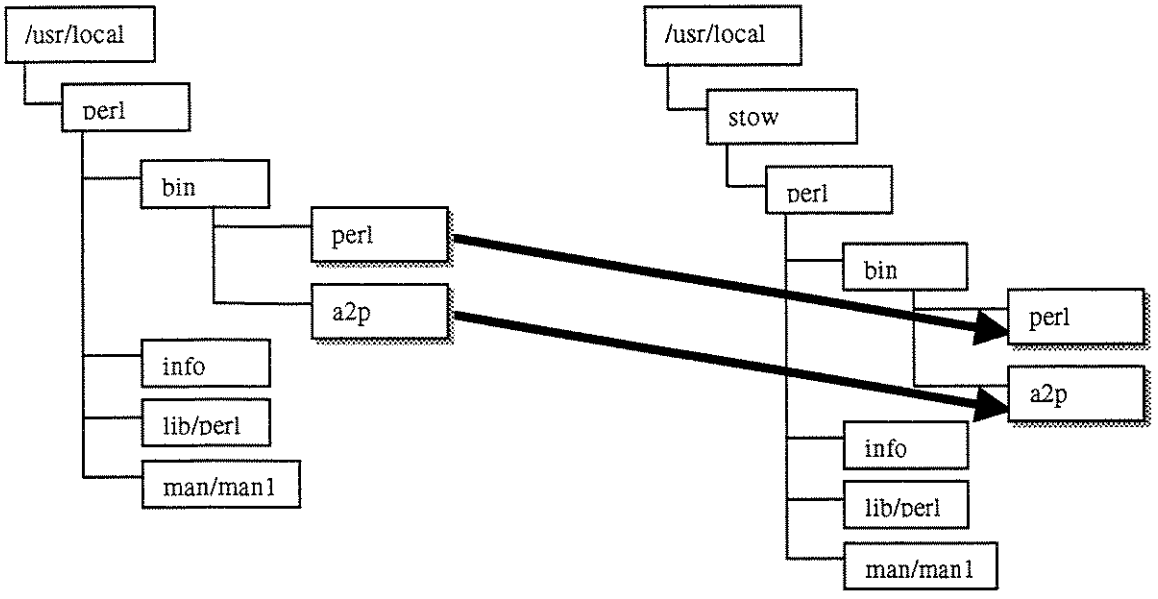

Figura 4.1.4d: Os *links* simbólicos dos executáveis `perl` e `a2p` do aplicativo `perl`.

A seqüência de passos que o administrador da rede deve executar para administrar um espaço de nomes (estrutura hierárquica de diretórios) através do programa `Stow`:

Passo 1: o administrador instala o aplicativo na localização desejada para utilização pelos usuários.

Passo 2: o administrador executa o programa `stow` para o *package* do aplicativo.

Passo 3: após executar o programa `stow`, o administrador testará o aplicativo quanto a sua funcionalidade.

Passo 4: o administrador de rede comunica e disponibiliza o aplicativo.

Em um outro exemplo, depois de instalar o aplicativo `perl`, o administrador de rede instalará o aplicativo `emacs` e executará novamente o programa `stow`. No procedimento de instalação do aplicativo `emacs` será utilizado o diretório `/usr/local/emacs/bin` para os executáveis `emacs` e `etags`.

O programa `stow` verifica o *package* do aplicativo `emac`, e cria a estrutura de diretório necessária em `/usr/local/stow/emacs`; Move os arquivos do aplicativo `emacs` para a nova estrutura; E cria os *links* simbólicos necessários, conforme figura 4.1.4e.

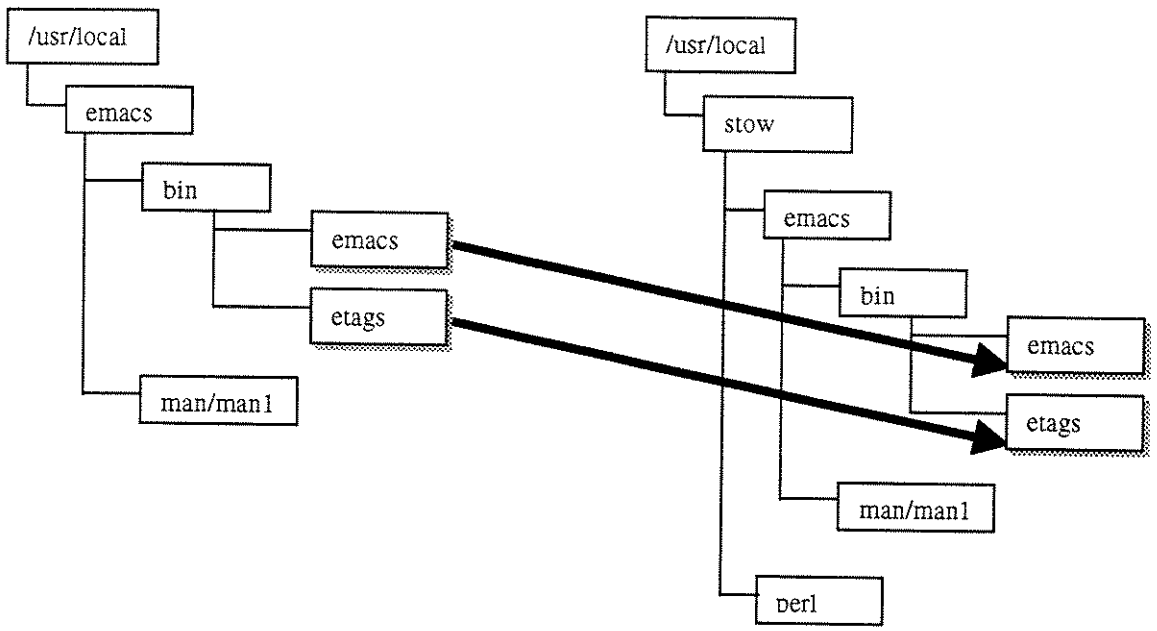

Figura 4.1.4e: Exemplo da nova estrutura de diretório para o aplicativo *emacs*.

O procedimento de remoção de um aplicativo para o programa `stow` executa os seguintes passos: remove fisicamente a hierarquia de diretório do aplicativo em `/usr/local/stow`; e remove os *links* simbólicos dos arquivos que apontam para a estrutura. Por exemplo, remover o aplicativo `perl`: o programa `stow` remove toda a estrutura de diretório de `/usr/local/stow/perl`; e remove os *links* simbólicos dos executáveis `perl` e `a2p` de `/usr/local/perl/bin`.

Com a nova estrutura hierárquica de diretório, gerada pela remoção de um software, o programa `stow` analisa e verifica a possibilidade de criar um único *link* simbólico de um diretório original do software para a estrutura hierárquica do stow, evitando assim criar vários *links* simbólicos para cada arquivo deste diretório. Por exemplo, criar um *link* simbólico de `/usr/local/emacs/lib` para `/usr/local/stow/emacs/lib`, se existir apenas arquivos de *libraries* do aplicativo `emacs`.

O programa `stow` permite que aplicativos que não são tratados por ele para o problema de espaço de nomes residam no diretório `/usr/local`, com a inclusão de um arquivo marca (`.stow`) dentro do diretório deste aplicativo. Quando o programa `stow` localiza o arquivo `/usr/local/foo/.stow`, este aplicativo `foo` não é analisado nos procedimentos de instalação e remoção de *package*.

Conflitos

Os Conflitos gerados e registrados durante a manutenção de aplicativos utilizando o programa stow ocorrem se:

- Durante a instalação, um arquivo ou *link* simbólico existe no diretório `/usr/local` (caso se opte por instalação plana dentro de `/usr/local`) e tem o mesmo nome que o que o programa `stow` deve criar.
- Um nome de diretório existe onde o programa `stow` precisa criar um *link* simbólico para um arquivo.
- Durante a instalação, o programa *stow* não consegue remover um *link* simbólico de um subdiretório do diretório `/usr/local` apontando para o diretório de `/usr/local/stow`.

Quando um conflito ocorre, a execução do programa `stow` para imediatamente e imprime uma advertência, a menos que a opção "-c" tenha sido definida, forçando a execução a continuar.

A execução do programa stow com as opções de "-n" ou "-c" não modifica o sistema de arquivos, apenas verifica os conflitos. Mas esta opção pode gerar um falso conflito pela não execução de alterações necessárias na estrutura hierárquica.

Bugs conhecidos

Os bugs registrados e divulgados do programa `stow` durante a sua execução para o procedimento de instalação ou remoção de um aplicativo são descritos como:

- Problema de diretório vazio. Se o aplicativo `foo` tem um subdiretório `temp` vazio, digamos `/usr/local/stow/foo/temp`, então:
  1. Se nenhum outro pacote tem o mesmo subdiretório `temp` vazio, está tudo bem (correto).
  2. Se outro *package* instalado, `quux`, tem um subdiretório `temp`, e quando se executa o programa `stow`: será criado um subdiretório em `/usr/local/stow/quux/temp`; e o *link* simbólico do diretório `/usr/local/temp` será removido. Mas quando um dos dois aplicativos (`quux` ou `foo`) for removido pelo programa `stow`, o subdiretório em `/usr/local/temp` será removido por estar vazio. Para evitar este bug, o administrador de rede deve criar um arquivo "`.placeholder`" dentro do diretório temp para conservá-lo.

A tarefa de administrar os conflitos gerados pelo programa `stow` para resolver o problema de espaço de nomes é simplificada quando se conhecem os eventos que geram os bugs dentro do programa `stow`.

As vantagens do método **Stow-Gnu** para resolver o problema de espaço de nomes na manutenção de software são descritas como:

- Documentação dos procedimentos: todos os procedimentos de instalação e remoção possuem regras claras e definidas para a manutenção de aplicativos, mas não existe um banco de dados que registre estas informações; o programa `stow` analisa, verifica e toma as decisões com base na estrutura hierárquica atual e os links simbólicos existentes.
- Padronização das atividades de manutenção de software no ambiente Solaris, principalmente os procedimentos de instalação e remoção.
- Os procedimentos são parametrizados para sua execução automática, sem a interação com o administrador da rede.
- O administrador de rede pode refazer o espaço de nomes sem perda de informações ou aplicativos. O programa `stow` permite um parâmetro de refazer (`restow`), onde se remove e instala o aplicativo, recriando os *links* simbólicos e diretórios.
- As tarefas de administração de software são planejadas e dimensionadas no tempo.

Em resumo, este método apresenta algumas vantagens em relação ao método Andrew, principalmente no tempo a ser gasto nos procedimentos de manutenção, mas também com relação à não dependência de sistemas externos.

As desvantagens do método **Stow-Gnu** para resolver o problema de espaço de nomes são descritas como:

- Não ocorre o registro da execução dos procedimentos de manutenção de aplicativos, quem executou a manutenção ou quando foi.
- O programa `Stow` só trata e corrige os softwares instalados no diretório `/usr/local`.

O fato do método **stow** só tratar os softwares do tipo *package* não constitui uma desvantagem para o método, porque o administrador de rede tem a possibilidade (liberdade) de tratar os softwares do tipo *Script* usando a metodologia e estrutura hierárquica de diretórios do `stow`. O administrador deve analisar e criar *scripts* que automatizem o procedimento de instalação e remoção deste tipo de aplicativo, respeitando a lógica e padronização adotada pelo método de *package* nativo do Solaris.

Baseando na metodologia do Stow são construídas ferramentas, como SEPP de [49], que também automatizam o problema de espaço de nomes como o programa Stow.

4.2 Problema de controle de versão

O problema de controle de versão, já definido na secção 2.2 do capítulo 2, é ativado quando existem mais de uma versão instalada de um mesmo aplicativo no ambiente Solaris.

O administrador de rede deve configurar ou disponibilizar as configurações que permitirão aos usuários utilizarem as varias versões instaladas de um aplicativo. Deve existir uma configuração para a versão do aplicativo instalado no espaço de nomes gerado pelo administrador da rede.

Por exemplo, para executar uma aplicação do `Sybase`, o administrador de rede deve configurar a variável de ambiente `DSQUERY` para identificar o banco de dados ativo da aplicação. Entretanto, outros usuários podem executar outras aplicações do `Sybase` com a variável `DSQUERY` definida com outro valor.

Serão apresentados dois modelos: Tradicional e o Wrappers-Sun que solucionam o problema de controle de versão.

4.2.1 Método Tradicional

O modelo **Tradicional** de controle de versão é aplicado para qualquer modelo de espaço de nomes descrito na seção 4.1 deste capítulo.

O administrador de rede deve criar para cada versão de software instalado uma configuração e instalar esta configuração para cada grupo de usuários, respeitando o perfil do usuário. Caso o usuário não utilize tal versão de software, deve de forma manual configurar seus arquivos de configuração do *Shell* (`.login`) para os parâmetros corretos de execução para a versão do aplicativo desejado.

Eis os passos que administrador de rede deve executar para definir a configuração de um aplicativo no ambiente Solaris:

Passo 1: recebe um pedido de solicitação de configuração para uma versão do aplicativo.

Passo 2: identifica a localização da versão do aplicativo no espaço de nomes, definido para este ambiente Solaris.

Passo 3: identifica as variáveis de ambiente que a versão do aplicativo utiliza. Normalmente, estas variáveis estão definidas nos arquivos de instalação ou help do aplicativo.

Passo 4: define o *path* da versão do aplicativo, para as várias versões de *Shell* utilizadas na rede.

Passo 5: Se necessário, define a montagem de diretórios para a correta execução do aplicativo.

Passo 6: Descreve todo o processo de configurar o ambiente Solaris.

Passo 7: Disponibiliza esta configuração para todos os usuário da rede, às vezes via *e-mail*, ou *bboard* eletrônico, que registra e armazena estas configurações de aplicativos.

Esta configuração do aplicativo pode reconfigurar todo o ambiente do usuário: variáveis de ambiente, *path*, montagem de diretórios etc.

Em alguns casos, o administrador de rede comunica via *e-mail* a todos os usuários da rede as novas versões e as alterações necessárias para que a nova versão do aplicativo execute de forma correta. O usuário

fica responsável pelas alterações em seu ambiente Solaris. Em outros casos, o administrador de rede pode disponibilizar as configurações via *bboard* eletrônico, registrando e mantendo todas as configurações possíveis para os aplicativos instalados. Em casos mais extremos, o administrador da rede pode remover as versões antigas do aplicativo e manter uma única versão, configurando o ambiente Solaris para esta versão.

Neste modelo, o administrador ou o usuário deve ficar reconfigurando seu ambiente Solaris para executar a versão desejada do aplicativo. Em alguns casos, as variáveis de ambiente são idênticas para os vários tipos de aplicativos instalados, obrigando a reconfiguração do ambiente a cada nova execução de um aplicativo.

As desvantagens do método **Tradicional** para resolver o problema de controle de versão são descritas como:

- O administrador de rede não registra e não controla individualmente as configurações de cada versão instalada do aplicativo.
- O usuário tem a liberdade e responsabilidade de configurar manualmente o seu ambiente Solaris para as versões de aplicativos que deseja utilizar, configurando o seu ambiente *Shell* (`.login`).
- Continua não sendo possível a identificação dos usuários de uma versão do aplicativo, visando a comunicação das alterações realizadas.
- O administrador de rede não tem informações sobre a utilização das versões instaladas de um aplicativo.
- O usuário pode ter muita dificuldade para configurar seu ambiente Solaris se o espaço de nomes utilizado na rede for o modelo Tradicional, onde todos os softwares são instalados em `/usr/local`.
- O número de solicitações de ajuda para configurar seu ambiente Solaris enviadas ao administrador de rede pode ser polinomial, em função do número de versões instaladas X aplicativos instalados X numero de usuários da rede.

O principal problema deste método se resume na responsabilidade do usuário de configurar seu ambiente Solaris para utilizar as versões de aplicativos desejadas.

4.2.2 Método Wrappers-Sun

Este modelo **Wrappers-Sun** foi desenvolvido nos laboratórios da SunSoft (Sun microsystems) como uma solicitação da *Sun Information Resources* [18].

Neste modelo, o administrador de rede deve criar um script que configura o ambiente Solaris para executar uma versão específica do aplicativo. Este *script* configura todo o ambiente necessário para esta versão do aplicativo em tempo de execução. Desta forma, ao término da execução deste *script*, as configurações geradas serão descartadas pelo ambiente Solaris.

Este encapsulamento de tempo de execução que o *script* possui permite criar ambiente corretamente configurado e de forma dinâmica para uma versão de aplicativo específica. Este ambiente Solaris configurado é valido apenas para este único usuário.

O administrador de rede pode identificar durante a execução de um *script* os vários parâmetros que definem o ambiente Solaris do usuário. Pode configurar o ambiente do usuário com base em: perfil de usuário, a estação que o usuário está utilizando, a versão do sistema operacional Solaris, a arquitetura de hardware da estação e outros. Com base nestes parâmetros, o administrador de rede pode configurar um aplicativo para sua execução de forma correta e dinâmica.

Os passos que o administrador de rede deve executar para definir a configuração de um aplicativo no ambiente Solaris:

Passo 1: recebe uma solicitação de criar um `Wrapper` para uma versão do aplicativo.

Passo 2: identifica a localização da versão do aplicativo no espaço de nomes no ambiente Solaris. Verifica as versões de binários e configura os pontos de montagem dos diretórios, se necessário.

Passo 3: identifica as variáveis de ambiente do aplicativo para sua correta execução.

Passo 4: escreve o *script* de execução desta versão do aplicativo, respeitando os parâmetros do usuário e do Solaris.

Passo 5: testa a execução do *script.*

Passo 6: define o nível de proteção do *script* e nome do *script.*

Passo 7: disponibiliza e comunica aos usuários este novo `wrapper`.

O administrador de rede cria um `Wrapper` para cada versão de um aplicativo instalado, identificando o `Wrapper` pelo nome do aplicativo e sua versão. Por exemplo: emacs20.19 ou perl4.05

Os testes que o administrador de rede deve aplicar no `Wrapper`, devem permitir a validação do `wrapper` para os tipos de binários do aplicativo com a versão do sistema operacional do Solaris e da arquitetura da estação do usuário que está executando o `wrapper`.

A figura 4.2.2a exemplifica um `Wrapper` de nome FOOV2.0, que verifica o ambiente Solaris do usuário, e depois configura as variáveis de ambiente e executa o aplicativo para versão 2.0.

```
# Wrapper: FOOV2.0 – Define ambiente de execução do aplicativo FOO versão 2.0, criando
# em 10-10-1998 por CAS.
#
# Define Shell como sendo Bourne Shell
!/bin/sh
        # Obtem o comando a ser executado atraves do programa basename,
        # que armazena o valor na variável cmd
        cmd=`/bin/basename $0`
        # Exporta a variavel de ambiente
        export FOOHOME
        # Define a variavel HOME de ambiente do aplicativo
        FOOHOME=/usr/apps/pkgs/footool_v2.0
        # Verifica tipo de arquitetura da estação
        case $arch in
                # Se arquitetura é Sun4 entao EXISTE binário para este aplicativo
                sun4)
                        ;;
                # Para qualquer outra arquitetura: enviar mensagem que não existe
                # Binário (aplicativo) para esta arquitetura
                *)
                        echo >&2 "Sorry, $cmd not available for $arch architecture."
                        exit 1
                        ;;
        esac
        # Configura a variavel para executar o aplicativo conforme ambiente do usuário
        command=$FOOHOME/bin/$cmd
        # Executa o aplicativo, respeitando a configuração do ambiente e do usuário
        exec $command ${1+"$@"}
```

Figura 4.2.2a: Exemplifica um wrapper FOOV2.0.

A linha 5 define o tipo de *Shell* que o `wrapper` foi escrito e será executado. Os tipos de *Shell* mais utilizados pelos administradores de rede são:

Bourne Shell	!/bin/sh
C shell	!/bin/csh –f
Korn Shell	!/bin/ksh

O mais utilizado no ambiente Unix é o *Bourne Shell* [08] por permitir a definição de funções e estrutura de controle do tipo *pipe into* e *pipe out*. Alguns administradores de rede estão utilizando `Perl` [52] pela flexibilidade e compatibilidade que a linguagem tem dentro do ambiente Unix.

Neste exemplo (`wrapper` FOOV2.0), o aplicativo foi instalado e configurado no espaço de nomes `/usr/apps/pkgs/footool_v2.0`. Com base nesta informação o administrador de rede cria um wrapper, que configura todo o ambiente Unix para sua correta execução nesta hierarquia de diretórios.

A figura 4.2.2b exemplifica um `Wrapper` de nome `QuuxV1.1`, que verifica o ambiente Solaris do usuário, e depois configura as variáveis de ambiente e executa o aplicativo para versão 1.1.

```
# Wrapper: QUUXV1.1 – Define ambiente de execução do aplicativo QUUX versão 1.1, criado
# em 08-10-1997 por CAS-Fenix.
#
# Define Shell como sendo Bourne Shell
!/bin/sh
        # Define a variável de ambiente library
        library=/usr/apps/pkgs/library_1.1
        # Disponibiliza a library correta para a arquitetura do aplicativo
        . $library/sh/arch.sh.fctn
        # Obtem o comando a ser executado atraves do programa basename,
        # que armazena o valor na variável cmd
        cmd=`/bin/basename $0`
        # Exporta a variavel de ambiente
        export QUUXHOME
        # Define a variavel HOME de ambiente do aplicativo
        QUUXHOME=/usr/apps/pkgs/Quux_v1.1
        # Verifica tipo de arquitetura da estação.
                # Se arquitetura é Sun4 então EXISTE binário para este aplicativo
                sun4)
                        ;;
                # Para qualquer outra arquitetura: enviar mensagem que não existe
                # Binário (aplicativo) para esta arquitetura
                *)
                        echo >&2 "Sorry, $cmd not available for $arch architecture."
                        exit 1
                        ;;
        esac
        # Configura a variavel para executar o aplicativo conforme ambiente do usuário
        command=$QUUXHOME/bin/$cmd
        # Executa o aplicativo, respeitando a configuração do ambiente e do usuário
        exec $command ${1+"$@"}
```

Figura 4.2.2b: Exemplifica um `wrapper QuuxV1.1`.

No exemplo `wrapper QUUXV1.1`, o aplicativo foi instalado e configurado no espaço de nomes `/usr/apps/pkgs/Quux_v1.1`. Com base nesta informação o administrador de rede cria um `wrapper`, que configura todo o ambiente Unix para sua correta execução nesta hierarquia de diretórios.

O administrador de rede deve criar um diretório para todos os `wrapper` disponíveis no ambiente Solaris, e disponibilizar o acesso para todos os usuários. Este usuário pode verificar as várias versões de `wrapper` disponíveis para um aplicativo.

As vantagens do método **Wrapper-Sun** para resolver o problema de controle de versão são descritas como:

- O administrador de rede tem o total controle e registro das configurações ativas para as versões de um aplicativo no ambiente Solaris.
- Padronização da tarefa de configurar os ambientes de execução dos aplicativos.

- Tratar os erros de execução: informa ao usuário, se a versão do aplicativo não está disponível para a versão do sistema operacional Solaris ou para a arquitetura da estação.
- Documentação: os *scripts* de `Wrapper` podem ser documentados para descrever a sua lógica e estrutura de execução.
- O usuário deve apenas escolher a versão e executar; não tem responsabilidade de configurar manualmente o seu ambiente Solaris para as versões de aplicativos que deseja utilizar.
- Não existirá caso de solicitação de ajuda para configurar o ambiente Solaris enviadas ao administrador de rede pelo usuário, quando utilizar o método `Wrapper`.

Este modelo não gera trafego excessivo na rede (portanto não degrada a performance da rede) para cada `wrapper` criado e configurado para uma versão de um aplicativo.

O administrador da rede pode informar os usuários através do `wrapper` de uma versão do aplicativo que existe uma outra versão mais recente instalada, configurada e liberada para uso.

Baseando na metodologia do Wrapper-Sun são construídas ferramentas, como SEPP de [49], que também automatizam o problema de controle de versões através de scripts escritos em perl [52].

4.3 Problema de monitoramento do uso de software

O problema de monitoramento do uso de software, já definido na secção 2.2 do capítulo 2, ocorre quando o administrador de rede necessita identificar quais softwares ou versões de um software são as mais utilizadas, planejando a remoção das menos utilizadas no ambiente de rede.

O administrador de rede deve possuir um banco de dados contendo todas estas informações (Tempo x Usuário x Estação x Software) para facilitar e simplificar a tarefa de gerenciar a classe de aplicativos no ambiente Solaris.

O desempenho da rede pode ser melhorado: o administrador de rede pode mover os aplicativos mais utilizados para os servidores mais rápidos da rede. Pode também remover os aplicativos que não são utilizados nos últimos 6 meses no ambiente Solaris, liberando mais espaço no servidor [41].

A tarefa de monitoramento do uso de software não deve ser confundida com a função de alguns softwares que fazem o controle sobre o número de licenças instaladas, conhecidos como Gerenciadores de Licença. Este monitoramento deve ser implementado de forma a não gerar tráfego pesado na rede ou degradar o desempenho da estação e ser transparente para o usuário.

Uma solução descrita em [36] para o monitoramento do trafego da rede, através de Simple Network Management Protocol (SNMP), onde os agentes (daemon) registram as informações de trafego pacotes na

rede em um banco de dados. Estas informações registradas podem ser consultadas pelo administrador da rede para aferir e diagnosticas problema de tráfegos na rede.

Outra solução de monitoramento da rede descrita em [35] permite verificar se o servidor está ativos para os serviços que ele disponibiliza na rede, como login , NFS e outros. A informação de que um servidor esta desativo (se recusa a responder aos testes) gera um e-mail para administrador de rede sobre a não disponibilidade de determinados serviços deste servidor.

Em resumo, o monitoramento do uso de software procura identificar os quais software são utilizados pelos usuários.

Nesta secção serão apresentados dois métodos: o Tradicional e o Solaris-Wrapper, que solucionam o problema de monitoramento do uso de software no ambiente Unix.

4.3.1 Método Tradicional

O método **Tradicional** foi assim definido por não possuir um banco de dados com informações sobre os softwares ativos na rede, e pela falta de metodologia e ferramentas que resolvam o problema de monitoramento do uso de software.

As informações, quando solicitadas ou necessárias, são obtidas de forma empírica e sem critério pelo administrador de rede; por exemplo:

- Para o caso de identificar quais usuários utilizam o software `perl` versão 4.02:
    1) Técnica A: o administrador de rede utiliza a combinação de comandos `ps` e `grep` para identificar os usuários, durante intervalos fixos de tempo na semana.
    2) Técnica B: envia *e-mail* para todos os usuários comunicando a possível remoção do software `perl` versão 4.02; conta os *e-mail* de retorno reclamando da remoção.
    3) Técnica C: verifica a estação que tem o software instalado e o número de vezes que a estação foi utilizada pelos usuários.
    4) Técnica D: verifica os tipos de usuários que utilizam o software (através das solicitações de acesso para os usuários novos) e o número de vezes que foi executado na semana.
- Quais os softwares que não são utilizados e estão instalados no servidor `Iguaçu`; remover para liberar espaço no disco:
    1) Reutiliza a técnica A por um período de tempo maior, 2 semanas.
    2) Reutiliza a técnica B.
    3) Reutiliza a técnica D.

Para identificar a taxa de utilização de um software ou de uma versão especifica do aplicativo na rede, deve-se utilizar um método que não produza degradação da velocidade das estações ou desempenho da rede.

As desvantagens do método **Tradicional** para resolver o problema de monitoramento do uso de software são descritas como:

- Não existe um banco de dados com estas informações. O administrador de rede deve inventar uma técnica de pesquisa para obter as respostas desejadas de forma empírica e sem metodologia no ambiente Solaris.
- Por não existir um método, não existe documentação ou registro da técnica ou processo executado para obter tais informações.
- Todos os processos são manuais e interativos para obter as informações desejadas.
- Não existe ferramenta que possa atender/responder a todas as solicitações possíveis do administrador da rede.
- Não existe padronização das pesquisas realizadas para serem no futuro automatizadas.

Muitos administradores de rede não se preocupam com o monitoramento do uso de software, mas este gerenciamento da utilização do software visa orientar a tarefa de administrar a rede.

Em resumo, este método descreve a total falta de ferramentas e softwares para obter informações sobre o monitoramento do uso de software na rede.

4.3.2 Método Solaris-Sun

Este método **Solaris-Sun** foi desenvolvido também nos laboratórios da SunSoft (Sun microsystems) como uma solicitação da *Sun Information Resources*.

Esta metodologia para monitoramento do uso de software no ambiente Solaris pode ser resumida em três etapas:

1) Gerar uma informação (registro) de utilização do software em uma estação Cliente;
2) Transferir os registros de informação do cliente (na fila local) para o servidor de monitoramento de software.
3) Formatar e imprimir os registros de informação do software.

Em resumo, os registros são padronizados e centralizados em um servidor, que responderá às consultas realizadas pelo administrador da rede.

As informações (ou registro) são gravadas na estação cliente em `/var/opt/SUNWswusg/swusage` através do programa `swu_rpt`. Este registro deve conter as seguintes informações do software executado: nome_software, tipo_software, subtipo_software, versão_software, nome_host, nome_usuário, data_execução, domínio, subdomínio e outras.

Os registros são transferidos da estação cliente para o servidor de monitoramento de software através do programa `swu_queue` e são gravados em `/var/opt/SUNWswusg/swusage.log` (`master log`). O servidor está ativo se o *daemon* `swu_svr` está executando no servidor.

O administrador de rede utiliza o programa `swu_print` para formatar e imprimir as informações armazenadas no arquivo `master log` do servidor.

Para cada utilização de um aplicativo devem ser gerados dois registros de informação, um de início da utilização e outro de término da utilização, permitindo calcular o tempo total da utilização do software.

O administrador de rede pode optar pela criação de vários servidores de monitoramento de software dentro da mesma rede. Também pode optar pela gravação dos registros diretamente no servidor, sem utilizar uma estação cliente; esta opção obriga que a rede esteja 100% estável e disponível.

O método **Solaris-Sun** permite gerar um registro de uso para um determinado software através de três formas distintas:

- Executar o comando `swu_rpt` através de linha de comando. Este método impõe que o usuário digite na linha de comando do *prompt* o comando para registrar o uso do software, e em seguida execute sua aplicação. Após o termino da utilização do software, o usuário deve digitar o comando para registrar o término da utilização do software.
- Executar o comando `swu_rpt` dentro de um `Wrapper` (*Shell script*). Nesta forma, o comando `swu_rpt` deve incluir código no início do `wrapper` para registrar o inicio da utilização, e no final do `wrapper` para registrar o término da utilização do software.
- Embutir a função `swu_rpt` dentro de uma aplicação. Esta forma deve incluir a função dentro do código fonte da aplicação, gerando os registros do uso da aplicação cada vez que a aplicação é executada. Para implementar este controle, o administrador de rede deve ter acesso ao código fonte da aplicação.

Esta última forma de gerar um registro está disponível para as linguagens C e C++. Para outras linguagens, o administrador de rede deve utilizar o processo de linkar os códigos objetos das diferentes linguagens.

Quando executado em ambiente *Client/Server* o software de monitoramento se comporta conforme a Figure 4.3.2-a.

Estação Cliente

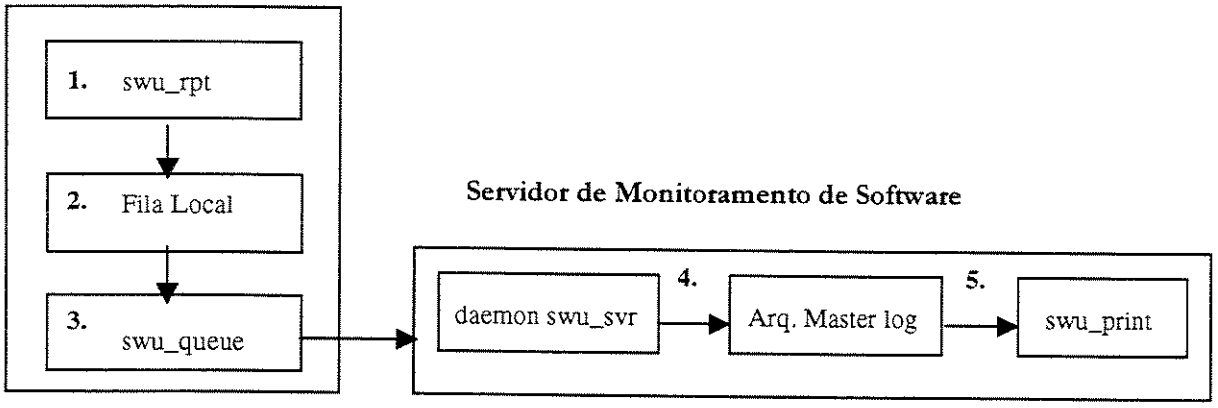

Figure 4.3.2a: Software de Monitoramento em ambiente *Client/Server.*

Os eventos de monitoramento do uso de software acontecem na seguinte seqüência:

1. O comando ou função `swu_rpt` inicializa o Servidor de Monitoramento de software ou a estação Cliente através das três formas possíveis (linha de comando, `Wrapper` ou dentro de uma aplicação).
2. O comando ou função `swu_rpt` cria um registro da informação na fila local para o software que está sendo executado. O tipo de registro depende das opções usadas com o comando ou função de `swu_rpt`. Se a fila local for inválida, nenhum registro será gravado.
3. Um job `cron` (executado a partir de `crontab`) usando o comando `swu_queue` na estação cliente ou no servidor, transferindo as informações da fila local para o *daemon* `swu_svr` que está no servidor. Estes arquivos podem ser transferidos manualmente pelo administrador, da fila local para o *daemon* `swu_svr` usando o comando `swu_queue` com a opção -F.
4. O *daemon* `swu_svr` aceita os registros de informação do comando `swu_queue` e salva no arquivo `/var/opt/SUNWswusg/swusage.log` (`Master de log`) ou em um arquivo de log especificado quando o *daemon* `swu_svr` foi inicializado no servidor. Se o *daemon* `swu_svr` não está executando o Servidor de Monitoramento de Software, especificado no registro, este registro de informação é removido da fila.
5. O comando `swu_print` transfere as informações do arquivo `master de log` (`/var/opt/SUNWswusg/swusage.log`) ou do arquivo de log especificado, copiando tudo, ou parte, ou um resumo das informações para outro arquivo, onde as informações podem ser pesquisadas usando `awk` ou outra ferramenta de pesquisa.

Esta seqüência de eventos mostra a ordem na qual os comandos são executados. Entretanto, o comando ou função `swu_rpt` pode ser executado um numero finito de vezes antes que o comando `swu_queue` transfira os registros de informações da fila local.

O Software de monitoramento opera de uma maneira semelhante no ambiente de compartilhamento de arquivo pela rede (*Share file system*), instalando o Servidor de Monitoramento de Software sobre um servidor, pode-se compartilhar os arquivos binários com outros *hosts* na rede.

Quando executado em ambiente de compartilhamento de arquivo, o software de monitoramento se comporta conforme a Figura 4.3.2b.

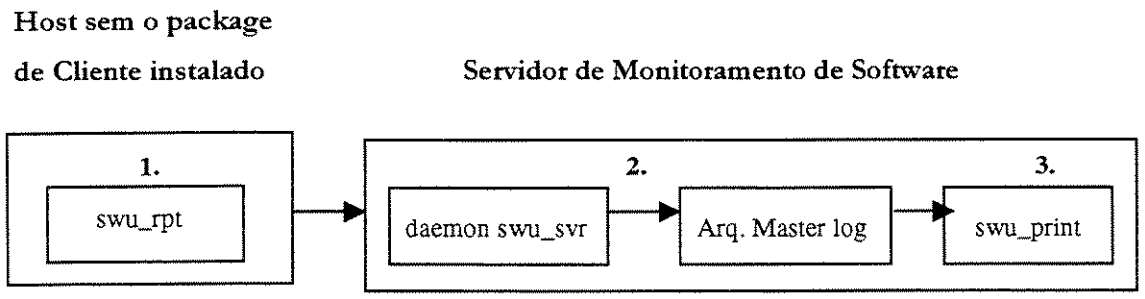

Figura 4.3.2b: Software de Monitoramento em ambiente compartilhamento de arquivo.

Os eventos de Monitoramento do uso de Software acontecem na seguinte seqüência:

1. O comando ou `função swu_rpt` é inicializada sobre um *host* a partir de uma linha de comando, `Wrapper`, ou dentro de uma aplicação.
2. O *daemon* `swu_svr` recebe a informação gerada a partir do comando ou função `swu_rpt` e grava as informações no arquivo Master de log (`/var/opt/SUNWswusg/swusage.log`) ou no nome de arquivo especificado quando o *daemon* swu_svr foi inicializado pelo *root*. Se o *daemon* `swu_svr` não está executando no servidor do *host*, o registro é perdido.
3. O comando `swu_print` transfere as informações do arquivo `Master de log` (`/var/opt/SUNWswusg/swusage.log`) ou do arquivo log especificado, copiando tudo, ou parte, ou um resumo das informações, para outro arquivo onde as informações podem ser pesquisadas usando `awk` ou outra ferramenta de pesquisa.

Para as formas de *Client/Server* e compartilhamento de arquivo, o administrador de rede deve configurar o ambiente Solaris de formas diferentes.

Usando linha de comando

Nesta interface o usuário deve digitar o comando `swu_rpt` com os parâmetros corretos para registrar uma informação de que um software está sendo utilizado. Normalmente, é gerado um registro de início e outro de término de utilização, mas é possível gerar ambas as informações em um único registro.

As desvantagens de utilizar esta forma de gerar o registro de utilização de um software são descritas como:

- Forma interativa e manual de gerar o registro: podem ocorrer erros nas informações geradas e gravadas.
- Disposição do usuário: este deve memorizar ou anotar todas as linhas de comandos para registrar o uso de todos as versões de softwares que utiliza na rede.

Usando Wrapper (Shell Script)

Utilizando o método de Wrapper-Sun para o problema de controle de versões, já descrito na seção 4.2 deste capítulo, o administrador de rede pode acrescentar as duas linhas de comando no `Wrapper` para gerar os registro de utilização de cada aplicativo no ambiente Solaris.

O administrador da rede deve parametrizar corretamente o comando `swu_rpt` com as informações de cada versão de um aplicativo dentro do `Wrapper` para garantir a execução de forma automatizada.

As vantagens de utilizar esta forma de gerar o registro de utilização de um software são descritas como:

- Forma automatizada e parametrizada de gerar o registro: o administrador pode configurar o `Wrapper` para gerar o registro quando executado.
- Padronização da tarefa de monitoramento do uso de software.
- Documentação da tarefa de monitoramento do uso de software.
- Forma de monitoramento é transparente e imperceptível para o usuário.

Esta forma de gerar o registro de utilização aproveita a simplicidade do método Wrapper-Sun, e acrescenta mais uma funcionalidade para o método.

Usando dentro de uma Aplicação

Esta forma de gerar o registro de monitoramento do uso de software impõe a obrigatoriedade do administrador da rede ter os códigos fonte do aplicativo para o acréscimo das linhas de código que geram o registro da utilização do aplicativo, recompilando o aplicativo para sua utilização.

A linguagem de programação utilizada nesta forma é o C ou C++. O administrador de rede deve acrescentar a linha de acréscimo da biblioteca `swusage.h`:

```
#include <swusage.h>
```

Incluir a função `swu_rpt` com os respectivos parâmetros para gerar o registro de início de utilização do aplicativo.

```
swu_rpt ("Server_name", "Identifier", SWU_BEGIN, "Product name", av1)
```

Depois, deve incluir a função `swu_rpt` com os respectivos parâmetros para gerar o registro de término de utilização do aplicativo.

```
swu_rpt ("Server_name", "Identifier", SWU_END, "Product name", av1)
```

Linkar as bibliotecas estáticas:

```
$ ... -I/opt/SUNWswusg/include -L/opt/SUNWswusg/lib -Bstatic -lswusage –lnsl ...
```

E linkar as bibliotecas dinâmicas:

```
$ ... -I/opt/SUNWswusg/include -R/opt/SUNWswusg/lib -L/opt/SUNWswusg/lib -lswusage -lnsl...
```

Um exemplo de aplicativo com código embutido para gerar os registros:

```
#include <stdio.h> /* definição de NULO */
#include <swusage.h> /* biblioteca para swu_rpt () */
#define ATTRIBUTE_COUNT 3

main()
{
    struct swusage_alist av1[ATTRIBUTE_COUNT];

    /*
    * Define uma estrutura a ser usado na registro de informações
    */
    av1[0].u_attr = "ATTR_1 ";
    av1[0].u_value = "val_1";
    av1[1].u_attr = "ATTR_2";
    av1[1].u_value = "val_2";

    /*
    * Termina a lista de atributos
    */
    av1[2].u_attr = NULL;
    av1[2].u_value = NULL;

    /*
    * Cria um registro de início de aplicação
    */
    swu_rpt("Server_name", "Identifier", SWU_BEGIN, "Product-name", av1);

    /*
    * O código de aplicação iria aqui.
    */

    /*
    * Cria um registro de término de aplicação
    */
    swu_rpt("Server_name", "Identifier", SWU_END, "Product-name", av1);
}
```

Os registros gravados na estação cliente na fila local devem ser transferidos para o arquivo `Master Log` no servidor de monitoramento de software, conforme figura 4.3.2c.

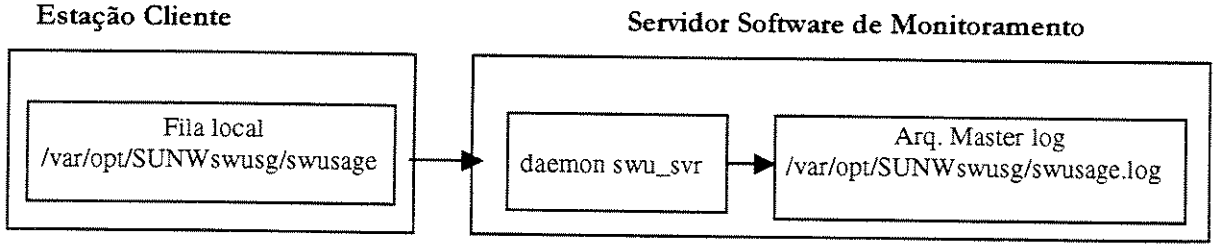

Figura 4.3.2c: registros de clientes sendo transferidos para servidor.

A fila local, localizada em `/var/opt/SUNWswusg/swusage`, contém os registros das informações de uso de um software. Cada registro é composto de uma linha de string alfanumérico.

Quando o Software de Monitoramento é instalado no ambiente Solaris, um job `cron` é incluído no `crontab` do *root* no host. Esta tarefa executa o comando `swu_queue` no intervalo entre meia-noite e 7 da manhã, transferindo as informações registradas na fila local da estação cliente para o arquivo `Master de log` em `/var/opt/SUNWswusg/swusage.log`.

Quando o comando `swu_queue` é executado, os arquivos de registros são transferidos da fila local para o arquivo `Master log`, e os arquivos de registros são removidos da fila local. Porém se os arquivos de registro não são transferidos (por exemplo, o *host* especificado no registro não estava executando), estes registros não são removidos da fila local, o que pode causar o enchimento do seu sistema de arquivos. Para impedir isto de acontecer, o administrador de rede deve desabilitar a fila local de receber registro de uso de software através do comando `swu_queue -d`.

Os possíveis problemas que podem acontecer quando se transferem os registros da fila local para o arquivo `Master log`:

- Se o comando `swu_queue` não pode se conectar com o daemon no servidor de monitoramento porque o host do registro não existe, então o registro é removido da fila local.
- Se o registro descreve um software que no host não existe, então o registro é removido da fila local.
- Se o *host*, especificado no registro, não está executando ou se há uma falha de *Remote Procedure Call* (RPC), o(s) registro(s) são mantidos na fila local para ser transferidos na próxima conexão.

As próximas conexões são realizadas no horário de meia-noite a 7 horas da manha.

O administrador de rede pode transferir ou imprimir estas informações do arquivo `Master log` para outro arquivo, `stdout` (tela) ou impressora. As informações dentro do arquivo `master log` não são formatadas para pronta visualização. O administrador de rede deve executar o comando `swu_print` para formatar e extrair as informações do arquivo de `master log` para outro arquivo ou dispositivo `stdout`. As opções de formatos de impressão para o comando `swu_print` são listadas na tabela 4.1a.

Formatando estes registros é possível a utilização de ferramentas do tipo `awk` ou outras ferramentas de busca por palavra-chave, como também permitir que os dados formatados possam ser transportados para um banco de dados ou utilitário.

O comando `swu_print` não apaga os registros de software do arquivo de `master log`, apenas tira uma cópia de tais registros para outro arquivo.

Opções	Exemplo de Formatos			
-a	PRODUCT SUBTYPE	TIME RECORD_ID	USER	HOST
	maker Begin	Feb 26 17:08:17Idstring	Jod	Sherlock
	vi Begin	Feb 27 19:09:19Idstring	Jad	Holmes
-d	TYPE=Admin/Usage Product=maker SubType=Begin Time=819653293 UserID=49740 User=jod Host=buck Domain=field.forest.com HostID=55003efc Locale=C Version=1 Server=Sherlock RecordID=Idstring			
-s	PRODUCT TIME	USER HOST		
	maker 01:31:47	jod buck		
	vi 19:09:19	jad doe		

Tabela 4.1a: Opções de formato de impressão para `swu_print`.

A tabela 4.1b descreve o formato das informações nos registros gerados, e sua descrição de cada formato.

As informações geradas em cada registro podem varia de campo para campo dentro de cada registro, mas a descrição de cada campo que compõe o registro não muda.

Para evitar que o arquivo `master log` torne-se muito grande, o administrador de rede deve limpá-lo periodicamente. As informações são copiadas para outro arquivo, como um *backup*, registrando o período de validade de tais informações (registros).

As vantagens do método **Solaris-Sun** para monitoramento do uso de software da classe aplicativos podem ser descritas como:

- Padronização da tarefa de monitoramento do uso de software: um único padrão para gerar, gravar, consultar e imprimir as informações sobre a utilização dos softwares.
- Documentação da tarefa: para cada `Wrapper` gerado a documentação da tarefa de monitoramento está implícita.
- Banco de dados centralizado com todas as informações sobre os aplicativos da rede: o administrador pode pesquisar por domínio ou subdomínios, facilitando a tarefa de gerenciamento.
- Método de monitoramento transparente e imperceptível para o usuário.
- Método automatizado para monitoramento: que permite registrar todas as informações necessárias para o correto gerenciamento do ambiente de rede, respondendo às consultas do administrador da rede.
- Método que não degrada a velocidade da rede e nem degrada o desempenho da estação do usuário.

Campo	Possível registro	Descrição	Quando usado
Type	Admin/Usage	Indica que o registro é do sistema de administração ou de um software usado.	Sempre
Product	Varia	Indica o nome do produto (ou programa) do software que está sendo registrado, especificado pelo comando ou função **swu_rpt**.	Sempre
User	Varia	Indica o usuário que esta executando o software. Esta informação será sempre o nome do usuário de login.	Sempre
SubType	Begin	Indica que o referido software iniciou a execução com o registro de inicio no tempo indicado.	Comando **swu_rpt**, opção de **-b** ou **–c**
Time	End	Indica que o referido software terminou a execução com registro de termino no tempo indicado.	Comando **swu_rpt**, opção **-e** ou **-c**
	Install	Indica que o referido software foi instalado com o registro de instalação.	Comando **swu_rpt**, opção **-i**
	Enable Queue	Indica que a fila local **swu_queue** foi habilitada.	Comando **swu_queue**, opção **-e**
	Disable Queue	Indica que a fila local **swu_queue** foi desabilitada.	Comando **swu_queue**, opção **–d**
	Varia	Indica o tempo que o software iniciou a execução, terminou a execução ou foi instalado. Este valor indica o valor em segundos, desde 00:00:00 UTC January 1, 1970.	Sempre
UserID	Varia	Indica o user ID do usuário que executou o software.	Sempre
Host	Varia	Indica o nome da estação de trabalho ou servidor onde o programa ou comando foi executado.	Sempre
Domain	Varia	Indica o nome do domínio onde o host está conectado.	Sempre
HostID	Varia	Indica o numero serial do hardware da estação ou servidor.	Sempre
Locale	Varia	Indica o valor da variável de ambiente LANG.	Sempre
Version	Varia	Indica versão do software de monitoramento (especificada pelo programa de monitoramento).	Sempre
Usage Server	Varia	Indica o Servidor para onde os registros estão sendo enviados.	Sempre
RecordID	Varia	Indica um único, numérico ID que é especificado com o comando **swu_rpt**. Se nenhum for especificado, serão usados como default o identificador do pai do processo ID e o seu tempo.	Sempre

Tabela 4.1b: Formato de cada registro de informação.

A única desvantagem deste método é fato de o administrador da rede ser obrigado a desenvolver rotinas para filtrar a base de dados gerada pelo monitoramento do uso de software. Estas rotinas podem ser automatizadas para gerar resumos de consultas ou relatórios totalizando as informações. Estas estatísticas de utilização de software não sofrem alterações no tempo.

4.4 Soluções Comerciais

Serão descritas nesta seção as soluções comerciais disponíveis para a manutenção de software no ambiente Unix para os procedimentos de instalação, configuração, atualização e remoção. As soluções foram divididas em duas categorias: para ambientes Solaris e ambientes Linux.

4.4.1 Soluções para ambiente Unix - Solaris

Serão descritas nesta seção duas soluções comerciais para o ambiente Unix – Solaris: a solução Tivoli da empresa IBM e a solução Unicenter da empresa Computer Associates. Ambas as soluções se enquadram na categoria de gerenciamento e administração do ambiente de rede Unix.

Solução Tivoli da IBM

Esta solução permite o gerenciamento e administração das estações de trabalho através de ambientes de redes corporativas. O sistema Tivoli [47] é dividido em maneiras diferentes de gerenciamento:

- **Tivoli IT Director:** para pequenas e médias empresas.
- **Tivoli Enterprise:** para grandes empresas dentro de um único local (interno: prédio, edifício, fabrica etc).
- **Tivoli Cross-site:** para grandes empresas distribuídas fisicamente em vários locais.

Para vários tipos de sistemas operacionais: Unix, NT, OS/2, Netware, OS/390 etc.

Os módulos disponíveis do sistema Tivoli para manutenção de software em ambiente de rede são descritos como:

- **Tivoli Inventory**: para registrar informações sobre uma estação em um banco de dados relacional (características físicas de uma estação e softwares instalados).
    A) As plataformas de sistemas operacionais para os servidores Tivoli de inventário são: AIX, HP-UX, Solaris, SunOS e Windows NT/2000.

B) As plataformas de sistemas operacionais para os gateways Tivoli de inventário são: AIX, HP-UX, Solaris, SunOS e Windows NT/2000.

C) As plataformas de sistemas operacionais para as estações que geram as informações para inventários podem ser: AIX, HP-UX, Netware, Sun Solaris, Windows 9x/Me/NT/2000, OS/2 e OS/400.

As informações que uma estação de arquitetura Intel podem registrar os seguintes itens: arquivos de configurações, informações do BIOS, recursos de I/O, memória, memória secundaria (discos rígidos), placas de rede, processador, aplicativos, sistema operacional e placa de vídeo.

As informações que uma estação de sistema operacional Unix pode registrar os seguintes itens: arquivos de configurações, recursos de I/O, memória, memória secundária, placa de rede, processador, aplicativos e sistema operacional.

- **Tivoli Software Distribution**: permite distribuir (instalar, configurar ou remover) um software através da rede.

  A) As plataformas de sistemas operacionais para os servidores Tivoli de distribuição de software podem ser: AIX 4.x, HP-UX 10.x e 11.x, Solaris e SunOS.

  B) As plataformas de sistemas operacionais para as estações que vão receber os softwares podem ser: Unix, Netware, OS/2 e Windows 9x/ME/Nt/2000.

  Este módulo permite criar um arquivo de perfil para o software que será instalado, registrando as informações necessárias para a correta manutenção do software.

  Para o ambiente Unix, os arquivos de configuração podem ser escritos em C, Shell scripts ou perl scripts.

- **Tivoli Distribution Monitoring**: permite monitorar os elementos da rede (estação, servidor etc) produzindo relatórios para o administrador da rede sobre: taxa de utilização da CPU, taxa de utilização da memória etc . O gerenciamento ocorre através de análise dos eventos gerados pelas estações, ou através de solicitações do administrador da rede especificas em parâmetros de pesquisa.

  As plataformas de sistemas operacionais que podem ser monitoradas são: AIX 4.x, HP-UX 10.x, Sun Microsystems (SunOS 4.1.2 to 4.1.4; Solaris 2.4 to 2.6), Windows 9x/NT/2000, NetWare, OS/2, Digital Unix, NCR Unix SVR4, SCO UnixWare, Siemens Nixdorf (Pyramid), Reliant Unix e Silicon Graphics.

As vantagens da solução Tivoli da empresa IBM para a manutenção de software em ambiente Unix são descritas como:

- Fornece uma ferramenta automatizada e padronizada para o administrador de rede utiliza na manutenção de software.

- Todos os procedimentos de manutenção de software (como instalação, configuração, atualização e remoção) estão documentados e especificados através de regras (perfil).
- O administrador possui ferramenta para gerenciamento do tráfego na rede.
- O administrador da rede tem um banco de dados com as características físicas e softwares das estações.

As desvantagens da solução Tivoli da empresa IBM para a manutenção de software em ambiente Unix são descritas como:

- O administrador de rede tem que resolver manualmente o problema de espaço de nomes para os procedimentos de manutenção de software. A solução Tivoli apenas executa o procedimento de manutenção baseado em regras.
- O administrador deve resolver manualmente o problema de controle de versões para a manutenção de software.
- O administrador de rede deve resolver o problema de monitoramento de uso de software para o ambiente Unix. O sistema Tivoli não possui solução para este problema;
- Não existe um perfil da estação de trabalho que define os procedimentos de manutenção de software.

Em resumo, um conjunto de ferramentas para automatizar a execução dos procedimentos de manutenção de software, ou seja, ele não tenta resolver os problemas descritos nesta dissertação.

Solução Unicenter da Computer Associates

O sistema Unicenter [48] foi criado pela empresa Computer Associates como um conjunto de soluções para o problema de gerenciar e administrar software via rede.

Os módulos disponíveis do sistema Unicenter para manutenção de software em ambiente de rede são descritos como:

- **Unicenter Aimit (Asset and Inventory Management)**: para registra informações (características físicas de uma estação e softwares instalados) sobre uma estação em um banco de dados relacional.

  As plataformas de sistemas operacionais para as estações que geram as informações para inventários podem ser: Windows 9x/Me/NT/2000, OS/2 e Macintosh.

As informações que uma estação de arquitetura Intel pode registrar as seguintes informações: CPU, disco rígido (tamanho e taxa de ocupação), informações do BIOS, memória, placas de rede, e arquivo de configurações.

Este módulo possui uma linguagem *script* que pode ser utilizada pelo administrador da rede para configurar informações de uma estação ou grupo de estações.

Permite definir ações (política de utilização da estação) para a ocorrência de eventos não autorizados, por exemplo: instalação de um aplicativo do tipo jogo pelo usuário.

- **Unicenter Shipit (Automated Software Delivery)**: permite distribuir (instalar, configurar ou remover) um software do tipo package através da rede.

  As plataformas de sistemas operacionais para os servidores que o módulo Unicenter Shipit permite distribuir software podem ser: Windows NT/2000, Unix, Linux (rpm), Netware, VMS e PalmOS.

  As plataformas de sistemas operacionais para as estações que o módulo Unicenter Shipit permite distribuir software podem ser: Windows CE/9x/NT/2000, Linux (rpm), DOS, e OS/2.

  Este módulo permite criar um arquivo de dependência para o software que será instalado, registrando as informações necessárias para a correta manutenção do software, e verificando os pré-requisitos do software antes de instalar.

  Possui mecanismo para desfazer uma manutenção de um software que teve um erro durante a execução do procedimento. O administrador recebe uma mensagem sobre esta ocorrência.

  O usuário pode escolher de um catálogo de software (banco de dados) quais software serão instalados em sua estação; Após este procedimento, as informações dos softwares instalados ficam registradas no Unicenter Aimit.

- **Unicenter Networkit**: permite monitorar os elementos da rede (estação, servidor etc) produzindo relatórios para o administrador sobre: taxa de utilização da CPU, taxa de utilização da memória, tráfego de byte pela rede etc . O gerenciamento ocorre através de análise dos eventos gerados pelas estações, ou através de solicitações do administrador da rede especificas em parâmetros de pesquisa.

  As plataformas de sistemas operacionais que podem ser monitoradas são: Unix, Windows NT/2000, Netware, AS/400, Tandem NSK, TCP/IP, DECnet, e SNA.

As vantagens da solução Unicenter da empresa Computer Associates para a manutenção de software em ambiente Unix são descritas como:

- Fornece uma ferramenta automatizada e padronizada para o administrador de rede utilizar na manutenção de software apenas para o ambiente Linux.

- Todos os procedimentos de manutenção de software (como instalação, configuração, atualização e remoção) estão documentados e especificados através de regras (perfil), apenas para o ambiente Linux.
- O administrador dispõe de uma ferramenta para gerenciamento do tráfego na rede.
- O administrador da rede tem um banco de dados com as características físicas e softwares das estações (alimentação manual destas informações) para o ambiente Linux.
- Existe um perfil para a estação de trabalho que define os procedimentos de manutenção de software para o ambiente Linux.

As desvantagens da solução Unicenter da empresa Computer Associates para a manutenção de software em ambiente Unix são descritas como:

- Esta solução só trata o ambiente Linux.
- O administrador de rede tem que resolver manualmente o problema de: espaço de nomes para os procedimentos de manutenção de software. A solução Unicenter apenas executa o procedimento de manutenção baseado em regras.
- O administrador deve resolver manualmente o problema de controle de versões para a manutenção de software.
- O administrador de rede deve resolver o problema de monitoramento de uso de software para o ambiente Unix. O sistema Unicenter não possui solução para este problema;

Em resumo, um conjunto de ferramentas para automatizar a execução dos procedimentos de manutenção de software, ou seja, ele não tenta resolver os problemas descritos nesta dissertação.

4.4.2 Soluções para ambiente Linux

As empresas que desenvolvem software para ambiente Unix estão direcionando suas atividades (soluções) para o ambiente Linux devido ao crescimento acelerado do número de estações que executam o sistema operacional Linux nos últimos 10 anos; estima-se em torno de 35 milhões de estações em todo o mundo para as mais variadas plataformas de hardware: Alpha, Intel, Sparc, PowerPC, m68k, SGI etc.

Com esta demanda de usuários, o ambiente Linux está oferecendo inúmeras soluções para resolver os problemas de manutenção de software, em oposição a outros sistemas operacionais compatíveis com Unix.

A empresa Red Hat produziu um padrão de distribuição de *package* para o ambiente Unix conhecido como Red Hat Package Manager, ou RPM [24]. Este padrão é utilizado pelos desenvolvedores de software sob a forma da *General Public License*, ou GPL.

Atualmente, o padrão de *package* RPM pode ser utilizado dentro das seguintes plataformas de sistemas operacionais: Linux (Sparc, Intel, PowerPc, Alpha, m68k e Sgi); OS/2; Solaris (Sparc/Intel, Solaris 2.4 e SunOS 4.1.3); Hewlett-Packard (HP-UX 9.04 e 10.20); SCO OpenServer 5.0.2; Osf 3.2; Aix (versões 3.2.5 e 4.1.4); Cygwin B20; LynxOs 3.0.1; Sinix 5.42; MachTen; Irix; Ncr-sys v4.3; FreeBSD; NetBSD; Mint; AmigaOS; e BeOS.

Em termos quantitativos, o sistema operacional Linux concentra o maior número de softwares desenvolvidos e distribuídos através do padrão de *package* RPM.

As soluções pesquisadas e que tratam especificamente do tema de manutenção de software para o ambiente Linux podem ser descritas como:

- Software `Aduva Manager/Aduva Director` da empresa Aduva.
- Software `Red Hat Network` da empresa Red Hat.
- Software `Enlightendsm` da empresa Enlightendsm.
- Software `Unicenter` da Computer Associates.
- Software `Xploy` da empresa Trustix.
- Software `Volution` [09] da empresa Caldera.

Uma comparação destas soluções para o ambiente Linux é apresenta através da tabela 4.1c.

Dentre todas estas soluções disponíveis para o ambiente Linux, esta dissertação descreverá a solução `Volution` [09] da empresa Caldera, porque dadas as suas características parece ser a solução mais interessante do ponto de vista da automatização da administração Unix.

Solução Volution da Caldera

O sistema `Volution` foi criado pela empresa Caldera Systems Inc. como um conjunto de soluções para o problema de gerenciar e administrar software via rede.

As principais características deste sistema `Volution` para manutenção de software em ambiente de rede são descritas como:

- **Interface do aplicativo de administração:** baseada em *Browser* (Web).
- **Distribuição eletrônica de software:** para os procedimentos de instalação e remoção de *package*

Comparação de manutenção de software no ambiente Linux						
Software	**Aduva Manager Aduva Director**	**Red Hat Network**	**Enlighttendsm**	**Unicanter TNG**	**Xploy**	**Volution**
Suporte p/ todas as distrib. Linux.	Não (só Red Hat)	Não (só Red Hat)	Sim	Sim	Sim	Sim
Distribuição eletrônica de Software.	Sim	Sim	Não	Não	Sim	Sim
Administrador pode instalar e configurar remotamente.	Sim	Sim (Agente pré-instalado em Red Hat 7.0)	Não	Sim	Sim	Sim
Instalação/Remoção de software usando RPM Linux.	Sim	Sim (não para remoção)	Não	Não	Sim	Sim
Inventário de Hardware e software em Linux.	Sim (inventario local)	Não (requer profile por estação)	Sim	Sim	Não	Sim
Agenda/gerenciador de evento para tarefas de administração.	Não (agenda instalação, atualização e remoção)	Não (agenda apenas instalação e remoção)	Não	Sim (Funções limitadas)	Não	Sim
Administrador define política, profiles, grupos e ações.	Não (só distribuição eletrônica de software).	Não (só distribuição eletrônica de software).	Não	Não (Limitada à política de sistemas linux).	Não	Sim
Disponível para ambiente Linux.	Sim	Sim	Não (requer ambiente Unix)	Não (Windows 9x/NT e/ou ambiente Unix)	Sim	Sim

Tabela 4.1c: Comparação de soluções para ambiente Linux.

do tipo RPM.

- **Inventário de Hardware:** verifica e registra a configuração de hardware de uma estação. Informações do tipo: CPU, memória, partições do disco, placa de rede, placa de vídeo etc. Ocorre o registro de um histórico das características físicas desta estação.
- **Inventário de Software:** verifica e registra a configuração de software de uma estação baseada nas informações destes *packages*. Ocorre o registro de um histórico de software instalados na estação.
- **Sistema de monitoramento do Linux:** permite acompanhar os aspectos críticos do sistema Linux e gerar mensagens de alerta através de SNMP ou SMTP para os eventos: de execução de aplicativos; número de processos; taxa de utilização da memória (em bytes ou percentagem); taxa de utilização da CPU; verificação do espaço livre do disco rígido; verificação do espaço livre de memória; verificação do número de usuários.
- **Configuração de impressora Linux:** permite a sincronização dos arquivos `printcap` no sistema Linux.
- **Criação de Profiles, grupos e ações:** permite criar arquivos que definem a política de administração da estação ou grupo de estações para os procedimentos de manutenção de software.
- **Repositório de dados:** registra todas as informações que manipula em um diretório `LDAP V3`.
- **Gerenciador de eventos:** permite enviar e receber mensagens para execução de procedimentos de manutenção de software via rede.

As vantagens da solução `Volution` da empresa Caldera Systems Inc. para a manutenção de software em ambiente Linux são descritas como:

- Fornece uma ferramenta automatizada e padronizada para o administrador de rede utiliza na manutenção de software do tipo *package* para o ambiente Linux.
- Todos os procedimentos de manutenção de software (como instalação, configuração, atualização e remoção) estão documentados e especificados através de regras (perfil) apenas o ambiente Linux.
- O administrador possui ferramenta para gerenciamento do tráfego na rede.
- O administrador da rede tem um banco de dados com as características físicas e softwares das estações (alimentação dinâmica destas informações) para o ambiente Linux.
- Existe um perfil para a estação de trabalho que define os procedimentos de manutenção de software para o ambiente Linux.

As desvantagens da solução `Volution` da empresa Caldera Systems Inc. para a manutenção de software em ambiente Linux são descritas como:

- O administrador de rede tem que resolver manualmente o problema de: espaço de nomes para os procedimentos de manutenção de software. A solução `Volution` apenas executa o procedimento de manutenção baseado em regras.
- O administrador deve resolver manualmente o problema de controle de versões para a manutenção de software.
- O administrador de rede deve resolver o problema de monitoramento de uso de software para o ambiente Linux. O sistema `Volution` não possui solução para este problema;
- Esta solução só trata o ambiente Linux.

Em resumo, um conjunto de ferramenta para automatizar a execução dos procedimentos de manutenção de software, ou seja, ele não tenta resolver os problemas descritos nesta dissertação.

Capítulo 5
Conclusões

Serão descritas neste capítulo as conclusões obtidas nesta dissertação em busca de ferramentas ou métodos que auxiliem a manutenção de software na classe de sistemas operacionais e aplicativos para o ambiente Solaris da Sun Microsystems.

A tarefa de administrar a manutenção de software em ambiente Unix está se tornando mais complexa e lenta com o passar dos anos, exigindo dos administradores de redes soluções rápidas, simples e automatizadas. Neste sentido, foram apresentados vários modelos de soluções para os vários problemas de manutenção de software para a classe de sistemas operacionais e aplicativos.

O objetivo final deste trabalho era apresentar soluções que reduzissem o tempo gasto nestas tarefas do administrador de rede, garantindo os objetivos de planejamento, padronização, documentação e acompanhamento. Porém esta redução só seria alcançada, para tais objetivos, se os procedimentos fossem automatizados. Como conseqüência, esta automatização obrigaria uma completa padronização e documentação dos procedimentos.

Na classe de sistemas operacionais, sua manutenção pode ser completamente automatizada através da solução **Custom JumpStart** [23], que permite a instalação, configuração, atualização e remoção (*patches*).

Como exemplo de automatização, o administrador de rede pode criar um *script* chamado Verificação_total, escrito em *Bourne Shell* [08], para: verificar se todos os softwares do Solaris estão instalados, através do comando *patchadd*; verificar as configurações do Solaris para rede; e verificar os aplicativos instalados. Havendo incompatibilidade do perfil da estação com os softwares instalados, este *script* fará as correções necessárias para respeitar o perfil da estação.

Em outro exemplo de automatização, o administrador de rede pode criar um *script* chamado Instalação_total, escrito em *Bourne Shell* para: fazer o *backup* da partição de dados da estação; formatar o disco; reinstalar o sistema operacional Solaris; configurar o Solaris para o ambiente de rede; instalar os aplicativos necessários; e restaurar o *backup* dos dados para a estação. Este *script* poderá reutilizar parte do *script*

Verificação_total. O produto final será uma estação completamente configurada para rede, com um único comando, e que respeita o perfil predefinido.

Os exemplos do *script* Verificação_total e do *script* Instalação_total obrigam o administrador a realizar um planejamento mais detalhado para a rede, porém obteria em troca uma padronização, documentação e automatização das tarefas para administrar o ambiente Solaris.

Exemplos de *scripts* para automatização das tarefas de administração de softwares estão exemplificados no Apêndice B.

A combinação do método **Andrew** para o problema de espaço de nomes, com o método **Wrapper** para o problema de controle de versões, juntamente com o método **Solaris-Sun** para o problema de monitoramento do uso de software constituem uma solução automatizada para os problemas de manutenção software para a classe de aplicativos. Esta combinação destes métodos tem a desvantagem de ser muito rígida, complexa e demorada para a execução das tarefas administrativas devido ao método Andrew. Porém, tem a vantagem extra de definir um ambiente para desenvolvimento de software no ambiente Unix.

A combinação do método **Stow** para o problema de espaço de nomes, com o método **Wrapper** para o problema de controle de versões, juntamente com o método **Solaris-Sun** para o problema de monitoramento do uso de software constituem uma solução automatizada para os problemas de manutenção software para a classe de aplicativos. Esta combinação destes métodos tem a vantagem de ser simples, flexível e mais rápida para a execução das tarefas administrativas. Tem ainda a vantagem de não possuir um banco de dados centralizado no método Stow que resolve o problema de espaço de nomes.

O acompanhamento da execução de um procedimento para a manutenção de software pode ser realizado através de envio de *e-mail* para o administrador, ou uma conta de *e-mail* específico para este fim, informando a execução de cada passo do procedimento ou os erros gerados. A geração e envio do e-mail deve estar contida dentro dos scripts.

Todas as soluções descritas nesta dissertação obrigam o administrador de rede a planejar, documentar e padronizar suas tarefas, que consomem um tempo enorme para sua implantação. Mas os resultados as justificam para se obter um ambiente Solaris completamente automatizado.

5.1 Resultados obtidos

Os cinco objetivos principais a serem alcançados, para resolver o problema de manutenção de software no ambiente Solaris, já descritos na seção "Classificação das tarefas" do Capítulo 2, são: planejamento, padronização, documentação, acompanhamento e procedimentos automáticos.

As conclusões obtidas nesta dissertação para o problema de criar um ambiente Solaris completamente automatizado são:

Classe Sistema Operacional Solaris

Dos métodos descritos para resolver o problema de instalar, atualizar, configurar e remover (*patches*) para o sistema operacional Solaris em ambiente de rede, o administrador deve optar pelo método **Custom Jumpstart** [23], que alcança os objetivos já descritos.

Para todo o ambiente Unix (com exceção ao Solaris) que não possuem a opção de "boot net" via hardware, o projeto Gnu desenvolveu um processo de boot equivalente ao Solaris, conhecido como `EtherBoot` [16]. E toda funcionalidade do método `Custom Jumpstart` (proposta) pode ser criado para outros sistemas operacionais Unix.

No Apêndice A foram descritos todos os procedimentos para implantar o método Custom Jumpstart em ambiente Unix.

Exemplificamos no Apêndice B os script para automatização dos procedimentos de instalação. configuração, atualização e remoção (patch) para o método Custom Jumpstart.

Classe Aplicativos

Dos métodos apresentados para resolver o problema do espaço de nomes para a classe de aplicativos no ambiente Solaris, o método **Stow** demonstrou ser, pelas vantagens que apresentou em relação aos outros métodos, a metodologia que mais se aproxima dos objetivos já descritos.

Para os métodos descritos para resolver o problema do controle de versão para a classe de aplicativos no ambiente Solaris, o método **Wrappers** demonstrou ser, pelas vantagens que apresenta em relação ao outro método, a metodologia que mais se aproxima dos objetivos já descritos.

Dos métodos apresentados para resolver o problema de monitoramento do uso de software para a classe de aplicativos no ambiente Solaris, o método **Solaris-Sun** demonstrou ser, pelas vantagens que apresenta em relação ao outro método, a metodologia que mais se aproxima dos objetivos já descritos.

Acreditamos que a combinação dos métodos: Stow, Wrappers e Solaris-Sun permitem automatizar os procedimentos de manutenção de software para a classe de aplicativos no ambiente Unix.

Atualmente, ferramentas, como SEPP [49], automatizam procedimentos de manutenção de software baseando-se nos métodos Stow e Wrappers, sendo que os scripts são escritos em perl.

Para executar os procedimentos de instalação, configuração, atualização e remoção na classe de aplicativos, o administrador de rede deve utilizar *scripts* para automatizar tais procedimentos através do método proposto **Custom Jumpstart**, que permite a utilização de scripts no início e término da sua execução.

5.2 Dificuldades encontradas

As dificuldades encontradas neste trabalho de produzir uma dissertação de mestrado podem ser resumidas em:

- **Ambiente de teste**: no inicio, foi sendo implementado e testado na própria rede do IC-Unicamp, mas devido à complexidade dos problemas e disponibilidade dos recursos envolvidos, como *software*, *hardware* e *staff*, tornou-se impossível manter este ambiente de teste. A solução encontrada foi à criação de uma rede particular e interna, mas com todas as características da rede do IC-Unicamp. Esta rede esta localizada no Laboratório de Administração e Segurança de Sistemas (LAS), criado especialmente para disponibilizar um ambiente de trabalho para pesquisa ligada as áreas.
- **Pouca literatura**: os problemas tratados nesta dissertação são elementares do cotidiano de manutenção de software, mas ao mesmo tempo, complexos e não resolvidos completamente. As soluções foram encontradas através da *Internet*, simulações e entrevistas, permitiu a troca de experiência com outros administradores de rede, através de *e-mail* , *chat* e principalmente através dos anais dos simpósios brasileiros de redes de computadores (SBRC), como em [01] e [02], e nos congressos internacionais do Usenix (LISA).
- **Conhecimentos de administração de rede**: para realizar um trabalho neste assunto, o pesquisador deve ter: experiência em sistemas operacionais Unix e suas configurações para as plataformas Intel e Sparc; conhecimentos de programa são em linguagem *Shell*, como *Bourne Shell*, *C shell* e *Korn Shell*; a dinâmica de compilar e recompilar um software para o problema de tempo de compilação e tempo de execução; ter o domínio sobre a estrutura física da rede, sabendo identificar as características físicas de uma estação, sua localização, o perfil definido, o tráfego gerado pelos softwares em rede.
- **Disponibilidade e tempo**: para acompanhar as tarefas diárias desenvolvidas pelo administrador da rede; para criar, instalar e configurar uma rede particular para os testes; e, principalmente, paciência para interagir com a rigidez do sistema operacional Unix, e as mensagens produzidas na manutenção de software.

Os bons profissionais envolvidos diretamente na administração de rede para ambiente Unix são aqueles que estão há muitos e muitos anos adquirindo e acumulando conhecimentos nesta área para produzir um ambiente estável e transparente para o(s) usuário(s).

5.3 Futuras extensões

As futuras extensões derivadas desta dissertação, que descreve procedimentos que automatizam a manutenção de software no ambiente Solarias, podem ser resumidas como:

- **Segurança**: implementar um método ou procedimento de segurança, até o nível de sugestão para a Sun Microsystems, que verifique via senha o procedimento de *Custom Jumpstart* (boot via rede). Isto impediria que qualquer usuário executasse o procedimento sem autorização, e principalmente, garantiria que o mesmo usuário não definisse uma senha para *root* durante o processo de instalação.
- **Login inteligente**: implementar a junção dos métodos apresentados nesta dissertação com o método de *login* inteligente e o método de cacheamento dos aplicativos, a nível de *automount*, como em [46], resultando em um ambiente configurado as necessidades do usuário durante o processo de login.
- **Ambientes diferentes**: implementar as soluções apresentadas para manutenção de software na classe de sistemas operacionais para outros ambientes Unix, como HP-UX, Linux e AIX. Assim como, implementar as soluções apresentados para os problemas de espaço de nomes, controle de versões e monitoramento do uso de software da classe de aplicativos para outros ambientes Unix.
- **Software Monitor Unix**: aproveitando a idéia da estrutura do método Solaris-Sun para o problema de monitoramento do uso de software na classe aplicativo, poderia ser desenvolvido um novo software (aplicativo) para monitoramento do uso de software no ambiente Unix, independente de fabricante e arquitetura da estação. Este aplicativo deve permitir a opção de formalizar as consultas, formatando os dados do banco de dados e permitir a impressão do resultado da consulta. Criar os parâmetros para as opções de: monitorar o tráfego na rede, critérios para remoção de um software e outros.
- **Software AcompManut Unix**: desenvolver e implementar um software para gerenciar a execução de procedimentos para a manutenção de software. Um centralizador (servidor) que recebe os *e-mails*/mensagens gerados nos procedimentos e verifica através de um *check list* a correta execução do procedimento. Enviaria uma única mensagem ou *e-mail* para o administrador de rede responsável pela execução relatando a completa execução do procedimento ou o erro gerado.
- **Software ServConfig Unix**: desenvolver e implementar um software para gerenciar os procedimentos criados e mantidos para manutenção de software no ambiente Solaris. Isto permitiria criar, editar e verificar a produção de *scripts*, *rules* e *profiles*, garantindo a padronização, documentação e planejamento desejados para estes procedimentos.

Apêndice A

Como configurar Custom Jumpstart

Este apêndice descreve os procedimentos para configurar o método Custom Jumpstart [23] no ambiente Solaris.

A.1 Como configurar os *defaults*: *date* e *time*, *time zone*, e *netmask*

Este procedimento habilita o programa sysidtool acessar os valores *defaults* de *date e time*, *time zone*, e *netmask* para uma estação. Este valores *defaults* estão configurados no NIS ou NIS+ [43].

1. No servidor de NIS ou NIS+, fazer login como *root*.
2. Editar o arquivo `/etc./hosts`, e criar um registro no formato *IP_address host_name timehost*
   Por exemplo: 129.44.0.3 marumbi timehost
3. Editar o arquivo `/etc./timezone`, e . criar um registro no formato *timezone domain_name*
   Por exemplo: Southamerica dcc.unicamp.br
4. Editar o arquivo `/etc./netmasks`, e criar um registro no formato *network_address netmask*
   Por exemplo: 128.32.0.0 255.255.255.0
5. Atualizar o servidor de NIS ou NIS+
   Se NIS usar: `cd /var/yp;make`
   Se NIS+ fazer:
   ```
   /usr/lib/nis/nisaddent -m -f /etc./hosts hosts
   /usr/lib/nis/nisaddent -m -f /etc./timezone timezone
   /usr/lib/nis/nisaddent -m -f /etc./netmasks netmasks
   ```

Durante o processo de instalação através de **Custom Jumpstart** via rede, o programa `sysidtool` automaticamente usará estes valores *defaults*.

A.2 Como configurar a língua *default* no *NIS*

Este procedimento configura a língua *default* para o *install client* no servidor de NIS [45].

1. No servidor de NIS, fazer *login* como *root*.
2. Editar o arquivo `/var/yp/Makefile`, e inserir a palavra locale na linha que começa com a palavra all.

   Por exemplo: all: passwd group hosts ethers networks rpc services protocols netgroup \
   bootparams aliases timezone locale

   Acrescentar o seguinte código depois do registro *.time.

```
locale.time: $(DIR)/locale
-@if [ -f $(DIR)/locale ]; then \
        sed -e "/^#/d"-e s/#.*$$// $(DIR) /locale \
        | awk '{for (i=2;i<=NF;i++) print $$i, $$0}' \
        | $(MAKEDBM) - $(YPDBDIR)/$(DOM)/locale.byname; \
        touch locale.time ; \
        echo update locale"; \
        if [ ! $(NOPUSH) ]; then \
                $(YPPUSH) locale.byname; \
                echo  "pushed locale"; \
        else \
        : ; \
        fi \
else \
        echo "couldn't find $(DIR) /locale"; \
fi
```

   No fim do arquivo, inserir `locale: locale.time` depois da entrada `map: map.time`

   Por exemplo: `timezone: timezone.time`
   `locale: locale.time`

3. Editar o arquivo `/etc./locale`, e inserir uma entrada no formato: `domain_name  locale`

   Os valores válidos para *locale* são: para `English` (*default* do Solaris) usar "C", `French` usar "fr", `German` usar "de", `Italian` usar "it", `Spanish` usar "es", e `Swedish` usar "sv".

   Por exemplo: `dcc.unicamp.br  C`

4. Fazer `Make` nos mapas NIS.

   Usar: `# cd /var/yp;make`

A.3 Como configurar a língua *default* no *NIS+*

Este procedimento configura a língua *default* para o `install client` no servidor de NIS+.

1. No servidor NIS+, fazer `login` como *root* ou um usuário do grupo de administração do NIS+
2. Criar a tabela `locale.org_dir` usando o comando `nistbladm`.

```
# nistbladm -D access=og=rmcd,nw=r -c locale_tbl name=SI,nogw= \
        locale=,nogw= corment-,nogw= locale-org_dir.'nisdefaults -d'
```

3. Acrescentar um registro na tabela `locale.org-dir` usando o comando `nistbladm` usado o formato: # nistbladm -a name=domain_name locale=locale comment=comment locale.org_dir. 'nisdefaults -d'

   Por exemplo: nistbladm -a name=dcc.unicamp.br locale=C comment=Inglesh German \
   locale.org-dir. 'nisdefaults -d'

A.4 Como configurar as informações *x86 default* no servidor *x86*

Este procedimento configura os valores x86 *defaults* para: *keyboard*, *display*, e *mouse* no servidor para x86. São requeridos pelo programa `sysidtool`, e são necessárias para o `OpenWindows` executar corretamente. Estas informações estão localizadas no arquivo `/etc./bootparams` no servidor de install ou de boot.

1. Fazer login como root no servidor de boot ou de install para x86. Este servidor guardará as informações do `install client` x86.
2. Usar o comando `kdmconfig` para configurar as informações do x86, no formato `# kdmconfig -c -s host_name`, onde –c inicia o programa de configuração, e -s host_name salva no arquivo `/etc./bootparams` do servidor as informações da estação x86 especificada pelo `host_name`.

   Por exemplo: `kdmconfig -c -s nova`. Este comando invoca o programa de configuração kdmconfig, que questionara as informações para configurar a estação nova. Estas informações serão inseridas no arquivo `/etc./bootparams` no servidor de install parana para o install client nova, no formato:

```
keyboard=nova:ATKBD;_layout_=US-English display=nova:ati;
_size_=15-inch pointer=nova:MS-S;_device_=/dev/tty00
```

Pode-se definir estas informações para uma família de estações que estão na mesma sub-rede, através de um simples (*):

```
* keyboard=nova:ATKBD;_layout_=US-English display=nova:ati;
_size_=15-inch pointer=nova:mS-S;_device_=/dev/tty00
```

A.5 Como configurar as informações *x86 default* no servidor *SPARC*

Este procedimento configura os valores x86 *default* para: *keyboard*, *display*, e *mouse* no servidor SPARC. Estas informações são requeridas pelo programa *sysidtool* e são necessárias para o *OpenWindows* executar corretamente. Estas informações são inseridas no arquivo `/etc./bootparams` no servidor de install ou de boot.

1. Usar o comando `kdmconfig` em qualquer estação x86 para inserir um registro no arquivo `/etc./bootparams`, no formato `# kdmconfig -c -s host_name`, onde –c inicia o programa de configuração, e -s host_name salva no arquivo `/etc./bootparams` do servidor as informações da estação x86 especificada pelo host_name.

   Por exemplo, `kdmconfig -c -s nova`, este comando invoca o programa de configuração `kdmconfig`, que questionara as informações para configurar a estação nova. Estas informações serão inseridas no arquivo `/etc./bootparams` no servidor de install parana (como exemplo) para o install client nova:

   ```
   nova keyboard=ecl:ATKBD;_layout_=US-English display=ecl:ati;
   _size_=15-inch pointer=ecl:MS-S;_device_=/dev/tty00
   ```

2. Deve-se alterar os valores de *keyboard*, *display*, e *pointer* para o host_name do servidor SPARC, por exemplo `parana`.

   ```
   nova keyboard=parana:ATKBD;_layout_=US-English display=parana:ati;
   _size_=15-inch pointer=parana:MS-S;_device_=/dev/tty00
   ```

3. Remover do registro de host_name da estação a ser instalada.

   ```
   keyboard=parana:ATKBD;_layout_=US-English display=parana: ati;
   _size_=15-inch pointer=parana:MS-S;_device_=/dev/tty00
   ```

4. Copiar este registro no arquivo `/etc./bootparams` do servidor SPARC.

   Pode-se definir estas informações para uma família de estações que estão na mesma sub-rede, através de um simples (*):

   ```
   * keyboard=parana:ATKBD;_layout_=US-English display=parana: ati;_size_=15-inch
   :pointer=parana:MS-S;_device_=/dev/tty00
   ```

A.6 Como criar um servidor de *install*

Este procedimento criar um servidor de install do Solaris, um único servidor atende toda a rede e suas sub-redes.

1. Fazer login como root no servidor de install.
2. Disponibilizar o leitor de cd-rom para o servidor de install
3. Usar o comando `setup_install_server` para copiar o conteúdo do CD Solaris para o disco local do servidor de install, com o comando: `# ./setup_install_server install_dir_path`, onde Install_dir_path especifica o diretório onde o CD Solaris será copiado. Por exemplo: `./setup_install_server /export/install`

Deve-se executar o procedimento "Como criar um servidor boot nasub-rede" se o servidor de install não está na mesma sub-rede dos install clients.

A.7 Como configurar o servidor de *boot* e de *install client*

Este procedimento configurar o servidor de boot e de install client para um sub-rede

1. No servidor de install, usar o comando add_install_client para configurar o servidor de boot e de install client, no diretório que o CD Solaris foi copiado, no formato `# ./add_install_client [-c server:jumpstart_dir_path ] host_name karch`, onde –c especifica o diretório `JumpStart` para instalação do tipo **Custom JumpStart**, server é o nome do *host* do servidor de profile onde o diretório de *JumpStart* está localizado. `Jumpstart_dir_path` é o *path* absoluto do diretório `JumpStart`; host_name é o nome do *host* do *install client* (não o nome do host do servidor de install). O nome do *host* deve estar no servidor NIS ou NIS+ [43] para o comando ser executar com sucesso; e *karch* é a arquitetura do *kernel* da estação a ser instalada, os valores válidos são sun4e, sun4d, sun4c, sun4m, and i86pc.

   Por exemplo: `./add_install_client -c parana:/jumpstart teen sun4c`

   O comando configura um servidor de install para boot e de install para um *install client* chamado `teen`. `Teen` é uma *SPARCstation IPC*, que tem a arquitetura de *kernel* sun4c. A opção -c especifica o servidor de *profile* chamado `parana`, e o *path* absoluto para o diretório `JumpStart` é `/jumpstart`.

Depois de executar o comando `add_install_client`, você deve usar o comando `kill` para parar o *daemon* `rpc.bootparamd` no servidor de boot, e reinicializar manualmente o *daemon* com o comando `/usr/sbin/rpc.bootparamd`, para reler as os registros de *bootparams* para o *install client.*

A.8 Como remover o *install client* no servidor *boot*

Este procedimento descreve como remover um *install client* no servidor de boot na mesma sub-rede do *install client*:

1. Fazer o *login* como *root* no servidor.
2. Usar o comando `rm_install_client` no formato `# ./rm_install_client host_name`.

A.9 Como configurar *Custom Jumpstart* para *x86*

Por *default*, um estação x86 executa o processo de instalação de forma interativa, entretanto, pode-se configurar a estação x86 para **Custom JumpStart** torne-se uma opção *default* de instalação.

1. Fazer *login* como *root* no servidor de install.
2. Mudar para o diretório `cd_image/export/exec/kvm/1386.186pc.Solaris_2.4/etc`., onde cd_image é o *path* do CD Solaris no disco local do servidor de install.

   Por exemplo: `cd /export/install/export/exec/kvm/sparc.sun4c.Solaris-2.4/sbin`
3. Alterar o arquivo `bootrc`, removendo a linha no fim do arquivo:

   ```
   echo '<<< starting interactive installation >>>'
   ```

   e inserir as linhas:

   ```
   setprop boot-args install
   echo '<<< starting custom jumpstart installation >>>'
   ```

A.10 Como configurar o tipo de terminal para *install client SPARC*

Este procedimento configura o tipo de terminal para o *install client* SPARC, devendo ser repetido para as diferentes arquiteturas de *kernel* do *install client.*

1 Fazer login como root no servidor de install.

2 Mudar o diretório para `cd_image/export/exec/kvm/sparc.karch.Solaris_2.x/lsbin`, onde cd_image é o *path* do CD Solaris no disco local do servidor de install.

Por exemplo: cd /export/install/export/exec/kvm/sparc.sun4c.Solaris-2.4/sbin

3 Alterar o arquivo `sysconfig` e inserir os seguintes registros no inicio do arquivo.

```
TERM=termtype
export TERM
```

A variável `termtype` é o tipo de terminal usado na instalação. Os tipos de terminais válidos são encontrados no diretório `/usr/share/lib/terminfo`.

Por exemplo:

```
TERM=vt100
export TERM
```

A.11 Como criar um servidor de *boot* na sub-rede

Este procedimento cria um servidor de boot na sub-rede que é necessário quando o *install client* e servidor de install estão em sub-rede diferentes.

1 Na sub-rede do *install client*, escolher uma estação para ser o servidor de boot e fazer *login* como *root*.

Esta estação deve estar registrada no servidor NIS ou NIS+ [45].

2 Mudar o diretório para a localização do CD Solaris.

3 Usar o comando `setup_install_server` para configurar o servidor de boot na sub-rede, no formato `# ./setup_install_server -b boot_dir_path karch`, onde –b especifica que a estação será configurada como servidor de boot; `boot_dir_path` especifica onde as informações da arquitetura do *kernel* serão copiadas; e *karch* especifica a arquitetura de *kernel* de um *install client*, os valores válidos incluem sun4e, sun4d, sun4c, sun4m, e i86pc

Comando `setup_install_server` copia as informações da arquitetura do *kernel* para o disco local e deve ser executado para cada tipo de arquitetura de *kernel* de install client.

A.12 Como criar um servidor de *profile*

Este procedimento cria um servidor de profile.

1 Fazer login como *root* no servidor onde ficará o diretório `JumpStart`.

2 Criar o diretório `JumpStart` no servidor com o comando `# mkdir jumpstart_dir_path`, onde `jumpstart_dir_path` é o path absoluto para o diretório `Jumpstart`.

Por exemplo: `mkdir /jumpstart`

3 Alterar o arquivo `/etc./dfs/dfstab` inserindo os seguintes registros.

```
share -F nfs -o ro, anon=0 jumpstart_dir_path
```

4 Reinicializar manualmente os *daemons* `nfsd` e `mountd`, para que o *install client* possa fazer o *mount* via NFS do diretório `jumpstart` durante a instalação.

5 Compartilhar o diretório de `JumpStart` com o comando `#shareall`

6 Montar o CD Solaris (se necessário) e mude para o diretório do `auto_install_sample` no CD.

7 Copia os arquivos de exemplos do `Custam JumpStart` do cd-rom para o diretório `JumpStart` no disco local com o comando `# cp -r auto_install_sample/* jumpstart_dir_path`, onde `jumpstart_dir_path` é o *path* absoluto para o diretório `JumpStart`.

Por exemplo: `cp -r auto-install_sample/* /jumpstart`

A.13 Como habilitar todos os *install clients* para acessar o servidor de *profile*

Este procedimento habilita todos os *install clients* da rede para acessar o servidor de profile.

1. No servidor de install ou de boot, fazer *login* como *root.*
2. Alterar o arquivo `/etc./bootparams` e inserir o seguinte registro:

   ```
   * install_config=server:jumpstart_dir_path
   ```

   onde * especifica todos os `install clients`; server é o *host* do servidor de profile onde está localizado o diretório `JumpStart`; e `jumpstart_dir_path` é o *path* absoluto para o diretório `jumpstart`.

   Por exemplo: `* install_config=parana:/jumpstart`

A.14 Como criar um *diskette* de *profile* para estação *SPARC*

Este procedimento cria um diskette de *profile* para estação SPARC.

1. Fazer login como *root* na estação SPARC que tem um drive para diskette e o CD Solaris disponível.
2. inserir um diskette no drive.
3. Formatar o diskette com o comando `# fdformat -U`.
4. Criar um sistema de arquivo UFS no diskette com o comando `# newfs /vol/dev/aliases/floppy0`.
5. Montar o CD Solaris (se necessário) e mudar o diretório `auto_install_sample` no cd.
6. Copia os arquivos de exemplos do **Custam JumpStart** do CD Solaris para o diretório *root* do diskette (diretório do `Jumpstart`).

A.15 Como criar um *diskette* de *profile* para estação *x86*

Este procedimento cria um diskette de *profile* para estação x86.

1. Fazer *login* como *root* na estação x86 ou SPARC que tenha um drive de diskette e o CD Solaris disponível.
2. inserir o diskette de boot do Solaris no drive.
3. Copiar o diskette de boot do Solaris para o disco local da estação com o comando # `dd if=/vol/dev/aliases/floppy0 of=boot_ímage`, onde `boot_image` é o nome do arquivo onde a imagem do diskette de boot será copiada.
   Por exemplo: `dd if=/vol/dev/aliases/floppy0 of=boot_save`
4. Fazer eject no diskette de *boot* Solaris do drive com o comando # `eject floppy`
5. Inserir um diskette virgem (sem dados) no drive.
6. Formatar o diskette com o comando # `fdformat -d -U`
7. Copiar o arquivo de imagem do diskette Solaris de boot do disco local da estação para o diskette formatado, com o comando # `dd if=boot_image of=/vol/dev/aliases/floppy0`.

A variável `boot_image` deverá ser a mesma do passo 3.

A.16 Como testar um *profile*

Este procedimento testa um arquivo de `profile`.

1. Fazer login como *root* no servidor de profile.
2. Configurar `SYS_MEMSIZE` para o tamanho especifico de memória em Mbytes da estação.

```
# SYS_MEMSIZE=memory_size
# export SYS_MEMSIZE
```

   Por exemplo:

```
SYS_MEMSIZE=15
export SYS_MEMSIZE
```

3. Mudar para o diretório `Jumpstart` onde estão os *profiles*, com o comando `# cd jumpstart_dir_path`.
4. Usar o comando `pfInstall` para testar o *profile* com o formato de `# /usr/sbin/install.d/pfinstall -D | -d disk_config [-c path ] profile`, onde -D faz o `pfinstall` usar o disco local da estação para testar o *profile*; -d `disk_config` faz o `pfinstall` usar o arquivo de `disk configuration`; -c `path` é o *path* do CD Solaris; e `Profile` é o nome do arquivo `profile`.

Você deverá executar `pfinstall` sobre uma estação executando a mesma versão de Solaris que será instalada pelo *profile*. Para garantir resultados corretos, o comando `pfinstall` pode ser executado do CD Solaris do diretório `/export/exec/arch.Solaris_2.4/sbin/install.d`, onde `arch` define a arquitetura do *kernel*.

5. Verificar os resultados do comando `pfInstall`.

A.17 Como criar arquivo de configuração do disco de uma estação *SPARC*

Este procedimento cria um arquivo de configuração do disco local de uma estação SPARC.

1. Fazer login como root na estação com disco local.
2. Usar o comando `df` para determinar o nome do device do disco local da estação
3. Salvar a saída do comando `prtvtoc`, para criar o arquivo de configuração de disco, no formato # `prtvtoc /dev/rdsk/cwtxdys2 > disk_config`, onde `/dev/rdsk/cwtxdys2` é o nome do `device` do disco local. Não especifique o número destino para discos do tipo `IDE`; disk_config é o nome do arquivo de configuração do disco.
4. Copiar o arquivo de configuração do disco para o diretório `JumpStart`.
   Por exemplo: `prtvtoc /dev/rdsk/c0t3d0s2 > 104mb.dcf`

A.18 Como criar arquivo de configuração do disco de uma estação *x86*

Este procedimento cria um arquivo de configuração para o disco local na estação x86.

5. Fazer *login* como *root* na estação com disco local.
6. Usar o comando `df` para determinar o nome do *device* do disco local da estação
7. Salvar a saída do comando `prtvtoc`, para criar o arquivo de configuração de disco, no formato # `prtvtoc /dev/rdsk/cwtxdys2 > disk_config`, onde `/dev/rdsk/cwtxdys2` é o nome do *device* do disco local. Não especifique o número destino para discos do tipo IDE; `disk_config` é o nome do arquivo de configuração do disco.
8. Copiar o arquivo de configuração do disco para o diretório JumpStart.

   Por exemplo: `prtvtoc /dev/rdsk/c0t0d0s0 > 500mb.dcf`

A.19 Como fazer o *boot* de um *install client SPARC*

Este procedimento é descreve como fazer o boot de um install client SPARC.

1. Ter certeza que *install client* está configurado para instalação do tipo **Custom JumpStart**.
2. Se está usando um diskette de *profile* para realizar a instalação do tipo **Custom JumpStart**, insira o diskette de *profile* SPARC no drive.
3. Se você está com boot a partir de um cd-rom local, insira o CD Solaris no drive de cd-rom local.
4. Se Solaris ou SunOS estiver executando, encerrar suas aplicações e executar os comandos:

   ```
   $ su
   # halt
   ```

   Para reiniciar a estação, execute o comando `stop-A`. Se a tela mostra o *prompt* ">" ao invés do *prompt* "ok", entre com "n" e pressione *Return*.
5. Para iniciar o processo de instalação via rede usar o comando `boot net - install`. Este comando pode variar em função do tipo hardware Sparc verifique [44].
6. Esperar até o processo de boot terminar. Se existe arquivo de `profile` para esta estação, esta instalação será automática.
7. Espere o software de instalação terminar o processo no *install client*. Este processo pode levar entre 15 minutos e 2 horas, dependendo da configuração a ser instalada.

A.20 Como fazer o *boot* de um *install client x86*

Este procedimento descreve como fazer o boot de um install client x86.

1. Ter certeza que os periféricos da estação *install client* estão configurados.
2. Inserir o diskette de boot do Solaris no drive. Se existe um diskette de *profile* desta estação para **Custom JumpStart**, inserir este do diskette de profile x86 no drive ao invés do diskette de boot do Solaris.
3. Fazer o boot do *install client* a partir do cd-rom local, inserir o CD Solaris no drive de d-rom. Ter certeza que configurou o *install client* para uma instalação do tipo **Custom JumpStart**.
4. Se Solaris ou SunOS estiver executando, encerrar suas aplicações e executar os comandos:

   ```
   $ su
   # halt
   ```

   Para reiniciar a estação, execute o comando `stop-A`. Se a tela mostra o *prompt* ">" ao invés do *prompt* "ok", entre com "n" e pressione *Return*.
5. Escolher o tipo de boot: por diskette, do drive de cd-rom local ou do servidor boot na rede.
6. Escolher a opção de instalação do tipo **Custom JumpStart**.

Para automatizar esta parte da instalação, deve configurar como *default* o tipo de instalação **Custom JumpStart**.

7. Espere o software de instalação terminar este processo no *install client*. Este processo pode levar entre 15 minutos e 2 horas, dependendo da configuração a ser instalada.

Apêndice B

Exemplos de scripts para Custom Jumpstart

Neste apêndice estão exemplificados exemplos de scripts para serem executados no inicio ou termino do método Custom Jumpstart.

B.1 Modelo para Start

```
#------------------------------------------------------------------------
#!/bin/sh
#
#       $Id: start,v 1.5 1995/02/17 13:49:49 casper Exp casper $
#
#       who     when    mods
#
#       jw      1/2/93 start
#       gp      3/7/93 removed setting of yp domainname
#

# We're read by the suninstall script. Time to set some parameters.

echo install_host = `uname -n`
echo ypdomain = `domainname`
echo 'timezone set to MET'
TZ=MET export TZ
umask 022

echo installing format.dat
cp /tmp/install_config/install/format.dat /tmp/root/etc
echo done

echo "SI_ARCH             $SI_ARCH"
echo "SI_BEGIN            $SI_BEGIN"
echo "SI_CLASS            $SI_CLASS"
echo "SI_CONFIG_DIR       $SI_CONFIG_DIR"
echo "SI_CONFIG_PROG      $SI_CONFIG_PROG"
echo "SI_CUSTOM_PROBES_FILE  $SI_CUSTOM_PROBES_FILE"
echo "SI_DISKLIST         $SI_DISKLIST"
echo "SI_DISKSIZES        $SI_DISKSIZES"
echo "SI_DOMAINNAME       $SI_DOMAINNAME"
echo "SI_FINISH           $SI_FINISH"
echo "SI_HOSTADDRESS      $SI_HOSTADDRESS"
echo "SI_HOSTID           $SI_HOSTID"
echo "SI_HOSTNAME         $SI_HOSTNAME"
echo "SI_INSTALLED        $SI_INSTALLED"
echo "SI_INST_OS          $SI_INST_OS"
echo "SI_INST_VER         $SI_INST_VER"
echo "SI_KARCH            $SI_KARCH"
echo "SI_MEMSIZE          $SI_MEMSIZE"
echo "SI_MODEL            $SI_MODEL"
echo "SI_NETWORK          $SI_NETWORK"
echo "SI_NUMDISKS         $SI_NUMDISKS"
echo "SI_OSNAME           $SI_OSNAME"
echo "SI_PROFILE          $SI_PROFILE"
echo "SI_ROOTDISK         $SI_ROOTDISK"
echo "SI_ROOTDISKSIZE     $SI_ROOTDISKSIZE"
```

```
echo "SI_SYS_STATE          $SI_SYS_STATE"
echo "SI_TOTALDISK          $SI_TOTALDISK"
echo "SI_TYPE             $SI_TYPE"
echo "SI_USERCHOICE         $SI_USERCHOICE"
echo "SI_USERSWAP           $SI_USERSWAP"

#
# Change newfs parameters. If you copy your cd to disk, you can
# rename newfs to newsfs.bin (in export/exec/sparc.Solaris_2.?/lib/fs/ufs)
# and copy the newfs script from instrall_tree/bin to ufs or continue
# to use the code below.
#

if [ ! -x /usr/lib/fs/ufs/newfs.bin ]
then
(
   cd /
   find usr/lib/fs/ufs -print | cpio -pdm /tmp
   cd /tmp/usr/lib/fs/ufs &&
   mv newfs newfs.bin &&
   cp -p $SI_CONFIG_DIR/bin/newfs . &&
   mount -F lofs /tmp/usr/lib/fs/ufs /usr/lib/fs/ufs
)
fi
#---------------------------------------------------------------------------
```

B.2 Modelo para No-Match

```
#------------------------------------------------------------------------
#!/bin/sh
#
#       None of the installrules did match. Don't install. Halt the system
#
#
echo "SI_ARCH             $SI_ARCH"
echo "SI_DISKLIST         $SI_DISKLIST"
echo "SI_DISKSIZES        $SI_DISKSIZES"
echo "SI_DOMAINNAME       $SI_DOMAINNAME"
echo "SI_HOSTADDRESS      $SI_HOSTADDRESS"
echo "SI_HOSTID           $SI_HOSTID"
echo "SI_HOSTNAME         $SI_HOSTNAME"
echo "SI_INSTALLED        $SI_INSTALLED"
echo "SI_INST_OS          $SI_INST_OS"
echo "SI_INST_VER         $SI_INST_VER"
echo "SI_NETWORK          $SI_NETWORK"
echo "SI_OSNAME           $SI_OSNAME"
echo "SI_TYPE             $SI_TYPE"
echo "SI_USERCHOICE       $SI_USERCHOICE"
echo "SI_USERSWAP         $SI_USERSWAP"

echo install_host = `uname -n`

echo "SI_BEGIN            $SI_BEGIN"
echo "SI_CLASS            $SI_CLASS"
echo "SI_CONFIG_DIR       $SI_CONFIG_DIR"
echo "SI_CONFIG_PROG      $SI_CONFIG_PROG"
echo "SI_CUSTOM_PROBES_FILE  $SI_CUSTOM_PROBES_FILE"
echo "SI_FINISH           $SI_FINISH"
echo "SI_KARCH            $SI_KARCH"
echo "SI_MEMSIZE          $SI_MEMSIZE"
echo "SI_MODEL            $SI_MODEL"
echo "SI_NUMDISKS         $SI_NUMDISKS"
echo "SI_PROFILE          $SI_PROFILE"
echo "SI_ROOTDISK         $SI_ROOTDISK"
echo "SI_ROOTDISKSIZE     $SI_ROOTDISKSIZE"
echo "SI_SYS_STATE        $SI_SYS_STATE"
echo "SI_TOTALDISK        $SI_TOTALDISK"
echo 'None of the installrules did match.'
echo 'No installation done'
echo 'halting system'
echo by the way system was a:
prtconf -p | grep '^Node'
echo halting system ";-)"
halt
#------------------------------------------------------------------------
```

B.3 Modelo para Finish

```
#-----------------------------------------------------------------------
#!/bin/sh
#
# $Id: finish,v 1.14 1996/08/09 15:36:50 casper Exp casper $
#
#       finish: install site specific files after auto-installation
#
# Authors:
#
#       Jan Wortelboer  (janw)
#       Gert Poletiek   (gert)
#       Casper Dik      (casper)
#

# We're read directly by the suninstall script. Execute all in subshell
# We don't want to influence our parent.

(

PATH=/sbin:/usr/sbin:/usr/bin:/bin:/usr/ucb:/usr/lib:/etc:.
export PATH

#
#  mountpoint of / for the new system
#
newroot=/a
export newroot

#
# set a nice default umask
#
umask 022

#
#  location of files to install
#
install=${SI_CONFIG_DIR}/install
scripts=scripts
files=files
classes=

#
# Switch filesystems to fast
#
$SI_CONFIG_DIR/bin/fastfs -a fast

domainname="`domainname`"
```

```
hostname="`uname -n`"

#
# Borrowed from /etc/rcS, needed for the standard patches script
# but added here because some other scripts may want it.
#

readvfstab() {
   _readvfstab "$@" < ${newroot}/etc/vfstab
}

_readvfstab() {
      while read special fsckdev mountp fstype fsckpass automnt mntopts
      do
            case ${special} in
            '#'* | '')     # Ignore comments, empty lines
                        continue ;;
            '-')        # Ignore no-action lines
                        continue
            esac

            if [ "${mountp}" = "$1" ]
            then
                  return 0
            fi
      done
      unset read special fsckdev mountp fstype fsckpass automnt mntopts
         return 1
}

#
#  Mount the last fs found by readvfstab (plus options)
#
mountfs ()
{
   if [ "x$mntopts" = x- ]; then mntopts= ; fi
   if [ -z "$fstype" ]
   then
         echo "******** can't mount filessytem, no successfull readvfstab"
         return 1
   fi
   msg=
   if [ "$fstype" = cachefs ]
   then
         msg=' (cachfs, mounted as r/o nfs)'
         fstype=nfs
         mntopts=ro
   elif [ $# = 1 ]
   then
         if [ -z "$mntopts" ]
         then mntopts=$1
```

```
        else mntopts=$mntopts,$1
        fi
    fi
    echo "Mounting $special on ${newroot}$mountp, -o${mntopts} $msg"
    /usr/sbin/mount ${mntopts:+-o$mntopts} -F $fstype -m $special ${newroot}$mountp
}

#
# If you change this, don't forget to update install_check.
#
#  findfiles locates a file in the directory ${SI_CONFIG_DIR}/install
#  and returns all matches (the hostname, one match for each class
#  (either class.domainname or class) the domainname.
#  If there are no matches, it returns default.
#
#  findfile returns only the first match
#
# Search order is:
#       ${install}/<file name>   if it's a file.
#       ${install}/<file name>/<system name>
# Then for each class to which a machine belongs:
#       ${install}/<file name>/<class>.<yp domain name>
#       ${install}/<file name>/<class>
# And finally:
#       ${install}/<file name>/<yp domain name>
#       ${install}/<file name>/default  (only returned when no other matches)
#

findfile ()
{
    findfiles -1 "$1"
}

findfiles ()
{
    match=
    if [ x"$1" = x-1 ]
    then
        shift;
        one=true
    else
        one=
    fi
    if [ -f ${install}/$1 ]
    then
        echo ${install}/$1
        return
    elif [ -f ${install}/$1/$hostname ]
    then
        echo ${install}/$1/$hostname
        [ -n "$one" ] && return
```

```
        match=true
    fi
    for class in $classes
    do
        if [ -f ${install}/$1/"$class.$domainname" ]
        then
            echo ${install}/$1/"$class.$domainname"
            [ -n "$one" ] && return
            match=true
        elif [ -f ${install}/$1/$class ]
        then
            echo ${install}/$1/$class
            [ -n "$one" ] && return
            match=true
        fi
    done
    if [ -f ${install}/$1/"$domainname" ]
    then
        echo ${install}/$1/"$domainname"
    elif [ -z "$match" ]
    then
        echo ${install}/$1/default
    fi
}

findconf ()
{
    if [ -f ${install}/$1/common ]
    then
        echo ${install}/$1/common
    fi
    set -- `findfiles $1`
    # We must return at least one file.
    if [ ! -r "$1" ]
    then
        echo /dev/null
    else
        echo $@
    fi
}

# Strip comments and blank lines from conf files.
readconf ()
{
    conf=`findconf $1`
    [ -n "$confverbose" ] && echo "$1      =" $conf 1>&2
    egrep -h -v '^#|^[      ]*$' $conf
}

cd $install
confverbose=true
```

```
classfile=`findfile class`
if [ ! -f "$classfile" ]
then
   classfile=/dev/null
fi
classes=`cat "$classfile"`
export classes
for f in ${install}/class/S[0-9]*.sh
do
   if [ -x $f ]
   then
         classes="$classes `. $f`"
   fi
done
>${newroot}/etc/INSTALL_CLASS
chmod 644 ${newroot}/etc/INSTALL_CLASS
for class in $classes
do
   echo $class >> ${newroot}/etc/INSTALL_CLASS
done

#
#        remove files in the rm.conf file
#
readconf rm.conf | while read opt target
do
         if [ -f "${newroot}$target" -o \
              -h "${newroot}$target" -o \
              -d "${newroot}$target" ]
         then
                  echo removing ${newroot}/$target
                  dir=`dirname ${newroot}/$target`
                  fil=`basename ${newroot}/$target`
                  ( cd $dir && rm -$opt $fil )
         else
                  echo "******* can't remove $target"
         fi
done


#
#        Create directories in the mkdir.conf file
#
readconf mkdir.conf | while read owner group mode dirname
do
         echo creating $dirname
         mkdir -p ${newroot}/$dirname
         chown $owner ${newroot}/$dirname
         chgrp $group ${newroot}/$dirname
         chmod $mode ${newroot}/$dirname
```

```
done


#
#       move files in the mv.conf file
#
readconf mv.conf | while read oldname newname
do
        if [ -f "${newroot}/$oldname" -o \
             -h "${newroot}/$oldname" -o \
             -d "${newroot}/$oldname" ]
        then
                echo renaming ${newroot}/$oldname to ${newroot}/$newname
                mv ${newroot}/$oldname ${newroot}/$newname
        else
                echo "******* can't move $oldname to $newname"
        fi
done

#
#       touch files in the touch.conf file
#
readconf touch.conf | while read target owner group mode comment
do
        echo "creating empty ${newroot}/$target ($comment)"
        touch ${newroot}/$target
        chmod $mode ${newroot}/$target
        chown $owner ${newroot}/$target
        chgrp $group ${newroot}/$target
done

#
#       Copy files in the copy.conf file
#
readconf copy.conf | while read source destdir owner group mode comment
do
  src=`findfile ${files}/$source`
  target=${newroot}/${destdir}/$source
  targetdir=`dirname "$target"`
  if [ -d "$targetdir" -a -r "$src" ]
  then
        echo installing $comment
        if [ -f $target -a ! -f $target.FCS ]
        then
          # keep linking relationships.
          cp -p $target $target.FCS; chmod ug-s,a-w,og-x,-t $target.FCS
        fi
        cp -p "$src" $target
        chmod $mode $target
        chown $owner $target
        chgrp $group $target
```

```
  else
        echo "******* can't install $destdir/$source ($comment)"
  fi
done

#
#       create symbolic links in the slink.conf file
#
readconf slink.conf | while read filename linkname
do
        targetdir=`dirname ${newroot}/$linkname`
        if [ -d $targetdir ]
        then
          echo sym-linking $linkname
          ln -s $filename ${newroot}/$linkname
        else
          echo "******* can't symlink $filename to $linkname"
        fi
done

#
#       Create hard links in the link.conf file
#
readconf link.conf | while read filename linkname
do
        targetdir=`dirname ${newroot}/$linkname`
        if [ -d $targetdir -a -f "${newroot}/$filename" ]
        then
                echo linking $linkname
                ln ${newroot}/$filename ${newroot}/$linkname
        else
                echo "******* can't link $filename to $linkname"
        fi
done

#
#       append to files in the append.conf file
#
readconf append.conf | while read file target
do
        if [ -f "${newroot}/$target" ]
        then
                file=`findfile ${files}/$file`
                echo appending $file to ${newroot}/$target
                cat $file >> ${newroot}/$target
        else
                echo "******* can't append to $target"
        fi
done

# Run the scripts last. So you can do everything you want there.
```

```
#
#       execute scripts to do things that cannot be done with the .conf files
#
readconf scripts.conf | while read script rest
do
        script=`findfile ${scripts}/$script`
        echo running $script
        . $script < /dev/null
done

#
# Switch filesystems back to slow
#
$SI_CONFIG_DIR/bin/fastfs -a slow

echo installation complete

# Subshell end
)
#-------------------------------------------------------------------------------
```

B.4 Modelo para remoção de uma lista de arquivos

```
#--------------------------------------------------------------------------
#
# opt   remove-target
#
# Following three don't exist after our normal install
#rf      /usr/local
#f       /etc/mail/sendmail.fc
#f       /var/spool/cron/crontabs/uucp
# This breaks patch installation.
#f       /usr/lib/sendmail.mx
rf       /var/nis
rf       /var/mail
f        /etc/rc2.d/S73autofs
```

Ou outro exemplo:

```
#!/sbin/sh
# Core file left from previous updates.
# Install_check is more careful now.
rm -f /var/spool/mqueue/core
# No autopatch
rm -f /etc/rc2.d/S74patch
#--------------------------------------------------------------------------
```

B.5 Modelo para atualização de patch

```
#-------------------------------------------------------------------------
#!/bin/sh

PERL=$SI_CONFIG_DIR/bin/perl
patches=/opt/install

#
# This system's filesystems are mounted at this point.
# The other filesystems need only be mounted read-only,
# at they are NFS filesystems that are owned by the server.
#
getpaths ()
{
  for f
  do
        echo $f
        dir=`dirname $f`
        case $dir in
        /) ;;
        /*) getpaths $dir;;
        esac
  done
}

echo "Trying to mount /usr, /usr/share and /usr/kvm, $patches"

for mp in `getpaths /usr/kvm /usr/share $patches|sort -u`
do
  if readvfstab $mp
  then
        mountfs ro
  fi
done

echo "mounting ${newroot}/tmp"
/usr/sbin/chroot ${newroot} /usr/sbin/mount -m -F tmpfs swap /tmp

# This item moved to the start of the finish script.
#echo "Switching filesystems to fast"
#$SI_CONFIG_DIR/bin/fastfs -a fast
if [ ! -d ${newroot}${patches} ]
then
  echo Failed to mount patch directory
  echo "Type control-D to continue"
  /sbin/sulogin > /dev/console < /dev/console 2>&1
fi
echo "running patch script"
```

```
status=1
if [ -x ${newroot}${patches}/fastpatch ]
then
   echo ""
   echo "WARNING: Using experimental fastpatch program (if you have $PERL)"
   echo "if this fails, we'll fallback to do-patch"
   echo ""
   $PERL ${newroot}${patches}/fastpatch -R ${newroot} \
         `/usr/sbin/chroot ${newroot} ${patches}/do-patch -c -d 2>/dev/null`
   status=$?
   /usr/sbin/chroot ${newroot} ${patches}/fix-modes
fi
[ $status = 0 ] || /usr/sbin/chroot ${newroot} ${patches}/do-patch -n -c
#echo "Switching filesystems to slow"
#$SI_CONFIG_DIR/bin/fastfs -a slow
sync
#-------------------------------------------------------------------------
```

B.6 Modelo para instalação de diretório

```
#------------------------------------------------------------------------
#!/bin/sh
#
# Script to expand single files to the <directory>/default
# for use in the installation scripts or the other way around.
#
# Casper Dik (casper@fwi.uva.nl)
#

if [ x"$1" = x-d ]
then
   dirs=true
   shift
fi
if [ x"$1" = x-f ]
then
   files=true
   shift
fi
if [ $# = 0 ]
then
   echo Usage: "$0 [-d|-f] file ..." 1>&2
   exit 1
fi
status=0
for f
do
   if [ -f $f ]
   then
         if [ -z "$files" ]
         then
            case "`file $f`" in
            *ELF*SPARC*) def=sparc;;
            *ELF*LSB*) def=i386;;
            *) def=default;;
            esac
            echo "converting $f to $f/$def"
            mkdir $f.d &&
            mv $f $f.d/$def &&
            mv $f.d $f
         else
            echo "$f is already a file"
         fi
   elif [ -d $f ]
   then
         count=`ls -A $f|wc -l`
         if [ $count -ne 1 ]
```

```
        then
          echo ${f}: not just one file. 1>&2
          status=1
        else
          def=`ls -A $f`
          if [ -z "$dirs" ]
          then
                echo "converting $f/$def to $f"
                mv $f $f.d &&
                mv $f.d/$def $f &&
                rmdir $f.d
          else
                echo "$f is already a directory"
          fi
        fi
   else
        echo ${f}: not a file or directory. 1>&2
        status=1
   fi
done
exit $status
#-------------------------------------------------------------------------
```

B.7 Modelo para correção de instalação

```
#---------------------------------------------------------------------
#
# Correct the ethernet modes (from 644 to 600)
#
installf SUNWcsd /devices/pseudo/clone@0:le c 11 40 0600 root sys
installf SUNWcsd /devices/pseudo/clone@0:ie c 11 63 0600 root sys
installf -f SUNWcsd
#---------------------------------------------------------------------
```

B.8 Modelo para gerar um vfstab

```
#-------------------------------------------------------------------------
#
# generate vfstab from existing vfstab and the updated vfstab
#


# The vfstab to append
fs=`findfile files/vfstab`

old=/etc/vfstab
new=/etc/vfstab.new

# Fix /tmp and /var with sed, awk to know when to stop.
sed    -e 's/^\(swap.*\)-$/\1nosuid/' \
          -e 's:^\(/dev/.*[          ]/var[    ].*\)-$:\1nosuid:' \
            < $old |
nawk 'BEGIN { flag=1 }
   /^#\/dev\/dsk\// { print $0; continue; }
   $3 == "/usr" || $3 == "/usr/kvm" { print $0; continue; }
   /^$/ { if (flag) { print $0; continue;}  }
   /^#/ { if (flag) { print $0; continue;}  }
    /^\/dev\/dsk\// { flag=0; print $0 ;}
   { if (flag) print $0; }
   ' > $new

cat $fs >> $new
chown root $new
chmod 644 $new
if cmp -s $old $new
then
   rm -f $new
else
   cp -p $old $old.old
   if [ ! -f $old.FCS ]; then mv $old $old.FCS ; fi
   mv $new $old
   echo New $old installed, old saved in $old.old
fi
#-------------------------------------------------------------------------
```

B.9 Modelo para script de install

```
#------------------------------------------------------------------------
#!/bin/sh
#
# $Id: auto_install,v 1.6 1995/03/07 14:17:06 casper Exp casper $
#
# Autoupgrade script. To be executed when a running system needs
# to be upgraded
#
# A system with security-mode set to none can be rebooted easily with
# ``reboot "net - install w".
#
# A system with security-mode full or command cannot be rebooted
# this way, to work around this, the following steps are taken:
#
#               * security-mode is switched to command
#               * diag-switch? is set to true
#               - diag-device will be set to net
#               - change the diag boot args to install w
#
#       actions marked with * must be undone in the install procedure.
#
# Casper Dik (casper@fwi.uva.nl)


# The diag file that is used.
bootfile='- install w'

PATH=/bin:/usr/sbin:/usr/bin:/usr/etc
export PATH

seteeprom()
{
  eeprom "$1=$2" || {
        echo "can't set EEPROM parameter '$1' to '$2'" 1>&2
        exit 1
  }
  echo "EEPROM parameter '$1' set to '$2'"
}

geteeprom()
{
  expr "`eeprom $1`" : "$1='\(.*\)'
}

#
# We can't auto-reboot of security mode is full, switch to command in
# that case. We can't switch to none, because we have to re-enter
```

```
# the password and we want to automate it all.
#
mode=`geteeprom security-mode`
if [ "$mode" = full ]
then
   seteeprom security-mode command
fi
#
# At this point, security-mode is either command or none.
# We can't pass a boot device and file to reboot if we have security mode
# set to command. We use the following trick: we define a diag-device
# and diag-file to boot ``net - install" and set diag-switch? to true.
# We undo the setting of diag-switch? in the install procedure.
# We leave the diag-file and diag-device setting for future installs.
# The EEPROM should be written as little as possible.
#
if [ $mode != none ]
then
   if [ "`geteeprom diag-device`" != net ]
   then
         seteeprom diag-device net
   fi
   if [ "`geteeprom diag-device`" != net ]
   then
         # Use alternate method.
         if [ -f /etc/hostname.le0 ]; then ether=le ; else ether=ie ; fi
         if [ "`geteeprom boot-from-diag`" != "$ether()$bootfile" ]
         then
            seteeprom boot-from-diag "$ether()$bootfile"
         fi
   elif [ "`geteeprom diag-file`" != "$bootfile" ]
   then
         seteeprom diag-file "$bootfile"
   fi
   seteeprom diag-switch? true
   reboot
else
   # Normal reboot install.
   reboot  "net $bootfile"
fi
#-----------------------------------------------------------------------
```

B.10 Modelo para configuração de Ambiente

```
#------------------------------------------------------------------------------
#!/bin/sh
#
# $Id: do-patch,v 1.26 1996/09/23 16:25:10 casper Exp casper $
#
# Automatic patch application script for Solaris 2.x.
#
# Patch.tar.Z files should be put in <path>/patches/`uname -r`s,
# where <path> is the directory where do-patch lives.
#
# The patches you want to have applied should be in patchlist.common.
# For servers, this script should be called as ``do-patch.server",
# in that case patches are taken from ``patchlist.server".
#
# Server are also all machines that locally mount /opt.
# That assumption may be wrong for some sites.
# (See also the -c and -s options)
#
# If a patchlist.`uname -n` exist, that file is used instead.
# If a patchlist.`uname -n`.add exist, that file is used as in addition to
# the standard files.
#
# Patches are unpacked in /tmp, unless <patchdir>/`uname -r`/unpacked/<patch>
# exists.
#
# Kernel architecture specific patches can be listed in patchlist.`uname -m`
#
# For some patches it was necessary to unhide what is in /usr/share.
# /usr/share is unmounted before applying patches and remounted
# afterwards. /usr/share must be unmounted, lofs mounted or in /etc/vfstab.
#
# By default, patches are only saved on servers, not on clients.
# When a previous version of a patch was installed and the files were
# saved, the patch is backed out.
#
# Usage: do-patch [-s] [-c] [-k] [-n] [-d] [-- installpatch args]"
#
# -k - don't execute install actions (default action is reboot)
# -n - don't reboot
# -d - list patches that need installing [debug]
# -c - install patches from patchlist.common, don't save old files [client]
# -s - install patches from patchlist.server, save old files [server]
#
# Before installing a patch, the patch.pre script is executed.
# After installing a patch, the patch.post script is executed.
#
# A log of a patch installation is left in /var/sadm/patchlog.<patchid>
```

```
#
# Casper Dik (casper@fwi.uva.nl)
#

umask 022
# Path needs to include perl's directory for rm-patchdata
PATH=/usr/sbin:/usr/bin:/etc:/sbin:/usr/local/bin
export PATH
installdir=`cd \`dirname $0\`; pwd`
#installdir=/opt/install
pdir=$installdir/patches/`uname -r`
if [ -d "$pdir.`uname -p`" ]
then
   pdir=$pdir.`uname -p`
fi

while getopts dkncs c
do
   case "$c" in
   d)  noaction=true
         not=true;;
   k)  noaction=true
         reboot=false;;
   c)  if [ "${type-client}" != client ]
         then
            echo "`basename $0`: options -c and -s are mutually exclusive" 1>&2
            exit 1
         fi
         type=client;;
   s)  if [ "${type-server}" != server ]
         then
            echo "`basename $0`: options -c and -s are mutually exclusive" 1>&2
            exit 1
         fi
      type=server;;
   n)  reboot=false;;
   \?) echo "Usage: `basename $0` [-csknd] [-- installpatch args]" 1>&2
         exit 1;;
   esac
done

shift `expr $OPTIND - 1`


# Backward compatibility (if type is not set on the command line,
# check whether do-patch is called as do-patch.server or whether
# /opt is a local filesystem)
if [ -z "$type" ] && {
         [ "`grep '^SERVER$' /etc/INSTALL_CLASS 2>/dev/null`" = SERVER ] ||
         [ "`basename $0`" = do-patch.server ] ||
         [ "`awk '$3 == "/opt" { print $4 }' /etc/vfstab`" = ufs ]
```

```
  }
then
  type=server
else
  : ${type:=client}
fi

if [ $type = server ]
then
  plist=patchlist.server
else
  flags=-d
  plist=patchlist.common
fi

xplist=
if [ -f /etc/INSTALL_CLASS ]
then
  for class in `cat /etc/INSTALL_CLASS` `uname -p`
  do
      if [ -f $pdir/patchlist.$class ]
      then
        xplist="$xplist patchlist.$class"
      fi
  done
  if [ -n "$xplist" ]
  then
      plist="$xplist"
  fi
fi

wildcard='$pdir/patchlist.[A-Z]*'
pkglist="`eval echo $wildcard`"
if [ "$wildcard" != "$pkglist" ]
then
  for f in $pkglist
  do
      pkg=`expr $f : "$pdir/patchlist.\(.*\)"`
      if pkginfo -q $pkg
      then
        plist="$plist $f"
      fi
  done
fi

# Use bootname to get FQDN?
hname="`uname -n`"
if [ -f $pdir/patchlist.$hname ]
then
  plist=patchlist.$hname
fi
```

```
# Additional patches for a host.
if [ -f $pdir/patchlist.$hname.add ]
then
   plist="$plist patchlist.$hname.add"
fi

if [ -d "$pdir" ]
then
        cd $pdir
else
        echo "No patches?!? You must be kidding."
        exit 0
fi
tmpf=/tmp/patches.$$

karch=`uname -m`

karchp=patchlist.$karch
if [ ! -f $karchp ] ; then karchp= ; fi
(
   cd /var/sadm/pkg;
   # In 5.5 the patches changed to "progressive"; each pkginfo filke
   # now contains possible multiple entries on a PATCHLIST line.
   # The lines are split by replacing whitespace with newlines.
   egrep -h '^(SUNW_PATCHID|PATCHLIST)=.' */pkginfo |
        sed -e 's/.*=//' -e 's/[    ][          ]*/\
/g' | sort -u > $tmpf
)
patches=`awk '{print $1}' $plist $karchp | fgrep -h -v -f $tmpf`
rm -f $tmpf

trap 'usershare' 0 1 15

nousershare()
{
   [ -n "$not" -o $type = server -o -n "$beenthere" ] && return
   loshare=`awk '/usr.share.*lofs/ { print $1 }' /etc/mnttab`
   umount /usr/share && remount=true
   beenthere=true
}

usershare()
{
   [ "$remount" != true ] && return
   if [ -z "$loshare" ]
   then
        mount /usr/share
   else
        mount -F lofs "$loshare" /usr/share
   fi
}
```

```
for p in $patches
do
   rm=""
   nousershare
   [ -z "$not" ] && echo "Installing patch #$p ...\c"
   if [ -d unpacked/$p ]
   then
         dir=$pdir/unpacked/$p
   elif [ -f $p.tar.Z -o -f alt/$p.tar.Z ]
   then
         if [ -f $p.tar.Z ]; then tarz=$p.tar.Z; else tarz=alt/$p.tar.Z; fi
         if [ -z "$not" ]
         then
            echo " unpacking ...\c"
            (zcat $tarz | (cd /tmp && tar xfB -))>/dev/null 2>&1
         fi
         if [ -n "$not" -o -d /tmp/$p ]
         then
            dir=/tmp/$p
            rm="$dir"
         else
            echo "Can't unpack patch $p - failed." 1>&2
            continue;
         fi
   else
         echo ""
         echo "Can't find patch $p - failed." 1>&2
         continue;
   fi
   if [ "$not" = true ]
   then
         # Generate listing on stdout, frills on stderr.
         echo "Would install patch #\\c" 1>&2
         echo "$p"
         continue;
   fi
   if [ -z "$noaction" -a -f "$p.pre" ]
   then
         echo "starting ...\c"
         . ./"$p.pre"
   fi
   if [ -d /var/sadm/patch ]
   then
         (
         cd /var/sadm/patch
         rev=`expr "$p" : '\(.*\)-'`
         if [ "${rev}*" != "`echo ${rev}*`" ]
         then
            old=
            rm -f /var/sadm/patchlog.${rev}*
```

```
        for oldp in `ls -td ${rev}*`
        do
                if [ -x "$oldp"/backoutpatch ]
                then
                  echo "trying to back out $oldp ..."
                  if [ -f $oldp/save.tar.gz ]
                  then
                        (
                        cd $oldp;
                        /usr/local/gnu/bin/gunzip < save.tar.gz |
                        tar xfBp -
                        rm -f save.tar.gz
                        )
                  fi
                  "$oldp"/backoutpatch "$oldp"
                  if [ $? != 4 -a $? != 0 ]
                  then
                        echo "Backoutpatch failed - aborting" 2>&1
                        exit 1
                  fi
                fi
        done
      fi
      ) || exit 1
      # Now remove patches also obsoleted by this patch or patches
      # w/o files saved. Should also work for saved patches.
      [ -x $installdir/fastpatch ] && $installdir/fastpatch -o -D $dir
fi
echo " installing ...\c"
if $dir/installpatch $flags "$@" $dir > /var/sadm/patchlog.$p 2>&1
then
      if [ -z "$noaction" ]
      then
        if [ -f "$p.post" ]
        then
              echo "finishing ...\c"
              . ./"$p.post"
        else
              # default action
              : ${reboot:=true}
        fi
      fi
      echo " done."
      # Log is kept in /var/sadm/patch/$p/log
      # This log is only interesting in case of a failure.
      rm -f /var/sadm/patchlog.$p
else
      echo " failed."
      mailx -s "Application of $p failed" root < /var/sadm/patchlog.$p
fi
if [ "$rm" != "" ]
```

```
    then
        rm -rf $rm
    fi
done

# Fwi specific change file modes and update script.
if [ -z "$not" -a -x $installdir/fix-modes ]
then
    $installdir/fix-modes
fi
if [ -z "$not" -a $type = client -a -x $installdir/rm-patchdata ]
then
    $installdir/rm-patchdata -r > /dev/null 2>&1
    # $installdir/fastpatch -obr > /dev/null 2>&1
fi
if [ "$reboot" = true ]
then
    echo Rebooting
    telinit 6
fi
#-------------------------------------------------------------------------------
```

B.11 Modelo para remover patch

```
#------------------------------------------------------------------------------
#!/bin/env perl
#
# $Id: rm-patchdata.pl,v 1.4 1994/11/03 13:18:28 casper Exp casper $
#
# Only keep data of the last patch.
# Will only remove patches which aren't saved, unless
# -f specified. Don't remove the last patch installed.
# Will only tell what will be done, unless -r specified.
# Other patches specified on the command line will be removed as
# well.
#
# This program is being subsumed by fastpatch.pl.
#
# Casper Dik (casper@fwi.uva.nl)
#

require 'getopts.pl';

$0 =~ s:.*/::;
if (!&Getopts('fr')) {
   print STDERR "Usage: $0 [-f] [-r] [ patch ... ]\n";
   exit 1;
}

for (@ARGV) {
   s/-.*//; # Remove revision
   $obs{$_} = "";
}

select(STDOUT); $| = 1;
select(STDERR); $| = 1;

for $f (</var/sadm/pkg/*/pkginfo>) {
   open(STDIN,"<$f") || die "$f: $!\n";
   ($pkg = $f) =~ s:^/var/sadm/pkg/(.*)/pkginfo$:$1:;
   while (<STDIN>) {
         chop;
         if (/^SUNW_PATCHID=/) {
            $patch = $';
            ($base,$rev) = split('-', $patch);
            $revs{$base} = $rev
                  if (!defined($revs{$base}) || $revs{$base} < $rev);
            if (defined($pkg{$patch})) {
                  $pkg{$patch} .= " " . $pkg;
            } else {
                  $pkg{$patch} = $pkg;
```

```
            }
        } elsif (/^SUNW_OBSOLETES=/) {
            for $p (split(',',$')) {
                ($base,$rev) = split('-', $p);
                $obs{$base} = "";
            }
        }

    }
}

for $patch (keys(%pkg)) {
    # Don't obsolete patches that have old files saved, unless -a specified.
    ($base, $rev) =  split('-', $patch);
    if (($rev < $revs{$base} || defined($obs{$base}))
                && ($opt_f || ! -f "/var/sadm/patch/$patch/.oldfilessaved")) {
        $obsolete{$patch} = "";
    }
}
open(PATCHADMIN,">/tmp/admin.$$");

print PATCHADMIN <<EOF ;
mail=
instance=unique
partial=nocheck
runlevel=nocheck
idepend=quit
rdepend=quit
space=quit
setuid=nocheck
conflict=nocheck
action=nocheck
basedir=default
EOF

close(PATCHADMIN);

for $patch (keys(%obsolete)) {
    # Remove the patch packages, followed by the patch directory.
    if (defined($opt_r)) {
        print "Removing: $patch\n";
        system "pkgrm -a /tmp/admin.$$ -n $pkg{$patch}";
        system "rm -rf /var/sadm/patch/$patch";
    } else {
        print "#$patch would be removed with the following commands:\n";
        print "pkgrm -a /tmp/admin.$$ -n $pkg{$patch}\n";
        print "rm -rf /var/sadm/patch/$patch\n";
    }
}
unlink ("/tmp/admin.$$");
#--------------------------------------------------------------------------
```

B.12 Modelo para makefile Solaris

```
#-------------------------------------------------------------------------
#
#  This is a Sun makefile. GNU make won't work.
#
DEST=/opt/install
#

CSRC= bootname.c rsleep.c modes.c checkattr.c fastfs.c
PLSRC = rm-patchdata.pl fastpatch.pl
TARGETS= $(CSRC:.c=)  $(PLSRC:.pl=)
HDRS= exceptions.h
SRCS=$(CSRC) $(PLSRC) $(HDRS)
SCRIPTS=fix-modes install_check auto_install do-patch
FTPDEST=/home/ftp/pub/solaris/auto-install
READMES=README.*
IREADMES=README ChangeLog
DISTRIB=Makefile $(SRCS) $(SCRIPTS) $(READMES)
TAR=$(FTPDEST)/update.tar
ITAR=$(FTPDEST)/install.tar
IFILES:sh=find itree -type f -print
.KEEP_STATE:
.SUFFIXES: .pl

.pl:
        rm -f $@
        cat $< > $@
        chmod -w,+x $@

all: $(TARGETS)

rsleep bootname := LDLIBS = -lnsl

install: all
        cp -p $(SCRIPTS) $(TARGETS) $(DEST)
        @if [ ! -d $(DEST)/patches ]; then mkdir $(DEST)/patches; fi


clean:
        rm -f $(TARGETS) *.o

dist: $(ITAR).gz $(TAR).gz $(IREADMES:%=$(FTPDEST)/%)
        chdir fmodes; $(MAKE) dist

$(TAR).gz: $(DISTRIB)
          tar cf $(TAR) $(DISTRIB)
          gzip -f $(TAR)
```

```
$(ITAR).gz: $(IFILES)
        tar cf $(ITAR) -C itree .
        gzip -f $(ITAR)

$(FTPDEST)/%: %
        cp -p $< $@
        chmod a+r $@

#$(FTPDEST)/README: README
#       cp -p README $(FTPDEST)/README
#       chmod a+r $(FTPDEST)/README
#-------------------------------------------------------------------------
```

B.13 Modelo para verificação de instalação

```
#-------------------------------------------------------------------------
#!/sbin/sh
#
# $Id: install_check,v 1.40 1995/07/15 14:36:11 casper Exp casper $
#
# Script to update a host form the install tree on the install
# server. This script is designed to be run from cron at night.
#
# Note to people about this script: don't print any messages to
# stdout. Stderr and stdout are used as record of modifications
# and failures. Any message to stdout will cause a reboot.
# Any message to stderr, will prevent it and will cause
# email to root.
#
#  This script can be run from crontab with the following options
#
#    -r <seconds>       sleep a random time less than n seconds
#                       (so all clients can be identical but won't
#                       run all jobs at the same time)
#    -f                 force execution (when users are present)
#    -n                 don't update
#    -l                 list all pathnames checked on stdout.
#    -k           don't reboot or kill
#    -s                 save diff output to $tmp/update.diff
#    -c                 extra careful, don't run when users present,
#                       don't even create a delayed file.
#
# Options to mask deficiency in cron.
#
#    -d <weekday>       run on <weekday> only, default is Mon.
#    -w                 weekly (execute every <weekday> morning)
#    -m                 monthly (first <weekday> morning of every month)
#
# Casper Dik (casper@fwi.uva.nl)

#
# set a nice default umask
#
umask 022

basedir="`dirname $0`"
basedir="`cd $basedir;pwd`"
#
# The directory basedir should contain the following programs:
# rsleep, bootname, checkattr, auto_install
#
PATH=/usr/sbin:/sbin:/etc:/bin:"$basedir"
```

```
export PATH

userprocs ()
# Processes that will be killed before a reboot, also used to test
# the presence of users on the system.
{
   ls -ln /proc | awk '$3 > 100 { print $9 }'
}

#
# All temporary files go in /var/sadm.
#
tmp=/var/sadm
#
# Parse arguments
#
weekday=Mon
time=0
[ -t 0 ] && dryrun=true

run=true
while getopts vcsklfmnwr:d: c
do
   case $c in
   v) verbose=true;;
   f) dryrun=; force=true;;
   m) run=; month=true;;
   n) dryrun=true;;
   w) run=; week=true;;
   r) time=$OPTARG;;
   d) weekday=$OPTARG
         case "$weekday" in
         Mon | Tue | Wed | Thu | Fri | Sat | Sun ) ;;
         *) echo "${0}: $weekday isn't a valid day" 1>&2 ; exit 1;;
         esac
   ;;
   k) noreboot=true;;
   l) list=true;;
   s) diff=true; exec 5>$tmp/update.diff ;;
   c) care=true;;
   \?)  echo "Usage: $0 [-d day] [-vfslkn] [-m|w] [-r secs]" 1>&2; exit 2 ;;
   esac
done

shift `expr $OPTIND - 1`

if [ $# != 0 ]
then
   echo "Usage: $0 [-d day] [-csklfn] [-m|w] [-r secs]" 1>&2
   exit 1
fi
```

```
[ -n "$month" ] && [ "`date +%d`" -le 7 ] && run=true
[ -n "$week" ] && [ "`date +%a`" = "$weekday" ] && run=true

tstamp=/var/sadm/last_update
dstamp=/var/sadm/update_delayed
rstamp=/var/sadm/must_reboot
[ -f $dstamp -a -z "$care" ] && force=true
[ -n "$run" -a -z "$dryrun$force" ] && [ -n "`userprocs`" ] && dryrun=yes
# Nothing to do.
[ -z "$run" -a -z "$dryrun" ] && exit 0

dontfix ()
{
   [ -n "$dryrun" ]
}

fatal ()
{
   if [ -s $tmp/update.stderr ]
   then
         cat $tmp/update.stderr 1>&4
   fi
   error "$@" 2>&4
   exit 1
}

error ()
{
   echo "*******" "$@" 1>&2
}

msg ()
{
   echo "$@" 1>&3
}

#
# Compute which pathname prefixes are not local to the system.
# This code breaks this script for diskless clients.
# It is only used for mkdir and copy since we create mountpoints
# (which when test are imported from the server) and some hidden
# files (needed when /opt cannot be mounted).
#
applic=
for path in `awk '$3 != "ufs" { print $2 }' < /etc/mnttab`
do
   if [ -z "$applic" ]
   then
         applic="$path|$path/*"
   else
```

```
            applic="$applic|$path/*|$path"
    fi
done
if [ -n "$applic" ]
then
    applic='case $1 in "'$applic'") return 0;; *) return 1;; esac'
else
    applic='return 1'
fi

notapplicable ()
{
    eval "$applic"
}

#
# Do locking
#
undo="rm -f $tmp/LOCK.$$"

rmnologin=

trap 'eval $undo; eval $rmnologin' 0
trap 'exit 1' 1 2 3 4 5 6 7 15

touch $tmp/LOCK.$$
ln -n $tmp/LOCK.$$ $tmp/update_lock || {
            echo "*** install_check locked" 1>&2
            exit 1
}
rm -f  $tmp/LOCK.$$
undo="rm -f $tmp/update_lock $tmp/update.stdout $tmp/update.stderr $tmp/patchlog"

#
# Redirect stdout, stderr. Old stdout is preserved on 3, old stderr on 4.
#

exec 3>&1 4>&2 > $tmp/update.stdout 2> $tmp/update.stderr
if [ -n "$verbose" ]
then
    pids=`
            tail -f $tmp/update.stdout 1>&3 2>/dev/null &
            pids=$!
            tail -f $tmp/update.stderr 1>&4 2>/dev/null &
            pids="$pids $!"
            touch $tmp/patchlog
            tail -f $tmp/patchlog 1>&3 2>/dev/null &
            echo $pids $!`
    undo="$undo; kill $pids"
fi
```

```
#
# First, we need the real hostname. Bootname is a program that
# first retrieves uname, followed by a gethostbyname, followed by
# a gethostbyaddr, which is what sort-of happens during net booting.
#
short_name="`uname -n`"
long_name=`bootname`

if [ -z "$long_name" ]
then
   fatal "Can't find hostname of $short_name"
fi



dontfix || {
   do=' was'
   rsleep $time
   if [ "$noreboot" != true ]
   then
         rmnologin="rm -f /etc/nologin"
         cat > /etc/nologin << EOF
NO LOGINS!

System update in progress.

EOF
   fi
}

bparms=`/sbin/bpgetfile install_config "" $long_name`

# $bparms is "  " when there's no answer.
set -- $bparms
if [ $# != 3 ]
then
   fatal "Can't find bootparam install_config for $long_name"
fi
install_server=$1
install_addr=$2
install_config=$3

if [ "$install_server" != $long_name -a "$install_server" != $short_name ]
then
   if mount -r ${install_server}:$install_config /mnt
   then
         undo="cd /;umount /mnt ; $undo"
         SI_CONFIG_DIR=/mnt
   else
         fatal "Can't mount ${install_server}:$install_config on /mnt"
   fi
```

```
else
   SI_CONFIG_DIR=$install_config
fi

export SI_CONFIG_DIR
iroot=$SI_CONFIG_DIR

if [ ! -x $iroot/bin/version ]
then
   fatal "install tree version unknown."
fi

instrel="`$iroot/bin/version`"
currel="`uname -r`"

if [ "$instrel" != "$currel" ]
then
   if [ -n "$instrel" -a -n "$currel" ]
   then
         # Check for a real upgrade, not a downgrade
         set -- `IFS=.; echo $currel`
         for i in `IFS=.; echo $instrel`
         do
            if [ $i -lt $1 ]
            then
                  fatal "install tree contains older OS"
            fi
            if [ $i -gt $1 ]
            then
                  # auto_install
                  fatal "want to auto_install $instrel"
            fi
            shift; if [ $# -eq 0 ]; then break; fi
         done
         if [ $# -eq 0 ]
         then
            # auto_install
            fatal "want to auto_install $instrel"
         else
            fatal "install tree contains older OS"
         fi
   fi
   fatal "install tree version mismatch, ${instrel:-unknown} != $currel"
fi

#
#  location of files to install
#
install=${SI_CONFIG_DIR}/install
scripts=scripts
files=files
```

```
domainname="`domainname`"
hostname=$long_name
#
# This code is copied from scripts/finish. It should be indentical.
#
#  findfiles locates a file in the directory ${SI_CONFIG_DIR}/install
#  and returns all matches (the hostname, one match for each class
#  (either class.domainname or class) the domainname.
#  If there are no matches, it returns default.
#
#  findfile returns only the first match
#
# Search order is:
#       ${install}/<file name>   if it's a file.
#       ${install}/<file name>/<system name>
# Then for each class to which a machine belongs:
#       ${install}/<file name>/<class>.<yp domain name>
#       ${install}/<file name>/<class>
# And finally:
#       ${install}/<file name>/<yp domain name>
#       ${install}/<file name>/default  (only returned when no other matches)
#

findfile ()
{
  findfiles -1 "$1"
}

findfiles ()
{
  match=
  if [ x"$1" = x-1 ]
  then
        shift;
        one=true
  else
        one=
  fi
  if [ -f ${install}/$1 ]
  then
        echo ${install}/$1
        return
  elif [ -f ${install}/$1/$hostname ]
  then
        echo ${install}/$1/$hostname
        [ -n "$one" ] && return
        match=true
  fi
  for class in $classes
  do
```

```
        if [ -f ${install}/$1/"$class.$domainname" ]
        then
           echo ${install}/$1/"$class.$domainname"
           [ -n "$one" ] && return
           match=true
        elif [ -f ${install}/$1/$class ]
        then
           echo ${install}/$1/$class
           [ -n "$one" ] && return
           match=true
        fi
    done
    if [ -f ${install}/$1/"$domainname" ]
    then
        echo ${install}/$1/"$domainname"
    elif [ -z "$match" ]
    then
        echo ${install}/$1/default
    fi
}

findconf ()
{
    if [ -f ${install}/$1/common ]
    then
        echo ${install}/$1/common
    fi
    set -- `findfiles $1`
    # We must return at least one file.
    if [ ! -r "$1" ]
    then
        echo /dev/null
    else
        echo $@
    fi
}

# Strip comments and blank lines from conf files.
readconf ()
{
    conf=`findconf $1`
    [ -n "$confverbose" ] && echo "$1      =" $conf 1>&2
    egrep -h -v '^#|^[      ]*$' $conf
}

#
# We'll only check newly installed files, links, symlinks and directories.
#

cd $install
```

```
classfile=`findfile class`
if [ ! -f "$classfile" ]
then
   classfile=/dev/null
fi
classes=`cat "$classfile"`
export classes
for f in ${install}/class/S[0-9]*.sh
do
   if [ -x $f ]
   then
         classes="$classes `. $f`"
   fi
done
>/etc/INSTALL_CLASS
chmod 644 /etc/INSTALL_CLASS
for class in $classes
do
   echo $class >> /etc/INSTALL_CLASS
done
#
#        Create directories in the mkdir.conf file
#
readconf mkdir.conf | while read owner group mode dir
do
   if notapplicable $dir; then continue; fi
   [ -z "$list" ] || msg $dir
   if [ ! -d $dir ]
   then
         echo $dir$do missing
         # To fix:
         dontfix || {
            mkdir -p $dir &&
            chown $owner $dir &&
            chgrp $group $dir &&
            chmod $mode $dir
         } || error "Can't mkdir $dir"
   elif checkattr $dir $owner $group $mode
   then
         :
   else
         chown $owner $dir &&
         chgrp $group $dir &&
         chmod $mode $dir
   fi
done

#
#    touch files in the touch.conf file
#
readconf touch.conf | while read target owner group mode comment
```

```
do
   if [ ! -f $target ]
   then
         echo $target$do missing
         dontfix || {
            touch $target &&
            chown $owner $target &&
            chgrp $group $target &&
            chmod $mode $target
         } || error "Can't create $target"
   elif checkattr $target $owner $group $mode
   then
         :
   else
         chown $owner $target &&
         chgrp $group $target &&
         chmod $mode $target
   fi
done

#
#        Copy files in the copy.conf file
#
linkcount ()
{
   set -- `ls -dn $1 2> /dev/null`
   echo ${2:-0}
}
readconf copy.conf | while read source destdir owner group mode comment
do
   file=`findfile ${files}/$source`
   [ "$destdir" = / ] && destdir=
   dest="$destdir/$source"
   if notapplicable $dest; then continue; fi
   [ -z "$list" ] || msg $dest
   if [ -d "${destdir:-/}" -a -r "$file" ]
   then
         if cmp -s "$file" "$dest"
         then
            if checkattr "$dest" $owner $group $mode
            then
                  :
            else
                  chmod $mode $dest &&
                  chown $owner $dest &&
                  chgrp $group $dest
            fi
         else
            echo $dest "($comment)"$do missing or changed
            if [ -f $dest -a -n "$diff" ]
            then
```

```
                echo "diff -c $file $dest" 1>&5
                diff -c $file $dest 1>&5 2>&5
            fi
            dontfix || {
                # Only remove a file if the link cont is 1,
                # we do this because we're lazy enough not to
                # have link.conf 100% correct.
                nlinks=`linkcount $dest`
                if [ -f $dest -a ! -f $dest.FCS ]
                then
                  if [ "$nlinks" = 1 ]
                  then
                        mv $dest $dest.FCS
                        nlinks=0
                  else
                        cp -p $dest $dest.FCS
                  fi
                  chmod ug-s,a-w,og-x,-t $dest.FCS
                fi
                if [ "$nlinks" = 1 ]
                then
                  rm -f $dest
                fi
                cp -p $file $dest &&
                chmod $mode $dest &&
                chown $owner $dest &&
                chgrp $group $dest
            } || error "Can't install $dest"
        fi
    else
        error "destination ${destdir:-/}, or source $file doesn't exist"
    fi
done

#
#       create symbolic links in the slink.conf file
#
readlink ()
{
    (ls -l $1 | awk '{ print $11 }') 2>/dev/null
}

readconf slink.conf | while read filename linkname
do
    if notapplicable $linkname; then continue; fi
    [ -z "$list" ] || msg $linkname
    if [ ! -h $linkname -o "`readlink $linkname`" != $filename ]
    then
        echo symlink $linkname to $filename$do missing or changed
        # To fix:
        dontfix || {
```

```
            rm -f $linkname &&
            ln -s $filename $linkname
        } || error "Can't link $filename $linkname"
    fi
done

#
#       Create hard links in the link.conf file
#
niequal ()
{
    set -- `ls -i $1 $2 2>/dev/null`
    [ "$1" != "$3" -o -z "$1" ]
}

readconf link.conf | while read filename linkname
do
    if notapplicable $linkname; then continue; fi
    [ -z "$list" ] || msg $linkname
    if niequal "$filename" "$linkname"
    then
        echo hardlink $linkname to $filename$do missing or changed
        # To fix:
        dontfix || {
            rm -f $linkname &&
            ln $filename $linkname
        } || error "Can't link $filename $linkname"
    fi
done

#
# Update scripts, when simple file changes aren't enough.
#
updatedir=$SI_CONFIG_DIR/updates
if [ -d $updatedir ]
then
    if [ ! -f $tstamp ]
    then
        updaters=`ls -tr $updatedir/*`
    else
        updaters=`find $updatedir -newer $tstamp -type f -print`
        if [ -n "$updaters" ]
        then
            updaters=`ls -tr $updaters`
        fi
    fi
    for i in $updaters
    do
        if dontfix
        then
            echo Must update with $i
```

```
        else
            . $i
            cp -p $i $tstamp
        fi
    done
fi

#
# And now for the patches.
#
#

do-patch -n ${dryrun:+-d} > $tmp/patchlog 2>&1


# Succeeded patches to stdout (include patches that call backout patch)
grep -v 'failed.$' $tmp/patchlog
# Failed patches to stderr
grep 'failed.$' $tmp/patchlog 1>&2

# At this point we redirect stdin and stderr to what they used to be
# and close the $tmp/update.* files.

exec 1>&3 2>&4 3>&- 4>&- 5>&-

if dontfix
then
    if [ -n "$run" -a "$dryrun" = yes ]
    then
        if [ -s $tmp/update.stdout ]
        then
            # Force update next time.
            touch $dstamp
        fi
    else
        # Show output (on stderr, 'cuz we want to list on stdout)
        cat $tmp/update.stdout $tmp/update.stderr 1>&2
    fi
else
    if [ -s $tmp/update.stdout ]
    then
        # Make a stamp that we really must reboot next time.
        if [ -n "$noreboot" ]
        then
            touch $rstamp $dstamp
        fi
        (date +"stdout of %C";
        cat $tmp/update.stdout; echo ----
        ) >> /var/sadm/update.log
    elif [ ! -f $rstamp ]
    then
```

```
        # From previous install_check -k.
        noreboot=true
    fi
    if [ -s $tmp/update.stderr ]
    then
        if [ -z "$verbose" ]
        then
            echo The following errors occured during update of ${long_name}:
            cat $tmp/update.stderr
        fi
        (date +"stderr of %C";
        cat $tmp/update.stderr; echo ----
        ) >> /var/sadm/update.log
        noreboot=true
    fi
fi

[ -n "$noreboot" ] || dontfix || {
        if [ -n "`userprocs`" ]
        then
                # Indent tabs, not spaces.
                wall <<- EOF > /dev/null 2>&1
                The system will be rebooted shortly by auto-update [$$].
                EOF
                sleep 120
        fi
        # Indent tabs, not spaces.
        wall <<- EOF > /dev/null 2>&1
        The system will be rebooted now by auto-update.
        EOF
        sigs='TERM HUP KILL'
        for sig in $sigs
        do
          # Kill non system processes.
          procs=`userprocs`
          if [ -z "$procs" ]
          then
                break;
          fi
          for proc in $procs
          do
                kill -$sig $proc > /dev/null 2>&1
          done
          sleep 15
        done
        rm -f $dstamp
        telinit 6
        rmnologin=
}
#---------------------------------------------------------------------------
```

B.14 Modelo para configurar segurança de arquivos

```
#------------------------------------------------------------------------
#!/bin/sh
#
# $Id: fix-modes,v 1.11 1996/09/23 09:12:50 casper Exp casper $
#
# Fix-modes script.
#
# Warning: no locking of the contents file is attempted.
#
# Casper Dik (casper@fwi.uva.nl)
#
PATH=/usr/local/etc:`dirname $0`:$PATH
export PATH
# Only tested for 5.2-5.5.1, check the modes program for each new revision
# and adjust script accordingly.
case "`uname -r`" in
5.[2345]*) ;;
*)
        echo "Only supported on Solaris 2.2 thru 2.5.1" 1>&2
        exit 1;;
esac

tpf=/tmp/fixmodes$$
trap 'rm -rf $tpf' 0
trap 'exit 1' 1 2 15
umask 077

if mkdir /tmp/fixmodes$$
then
   :
else
   echo "can't create temporary directory" 1>&2
   exit 1
fi

umask 022
changes=$tpf/changes

modes > $changes || {
        echo "modes failed, aborting fix-modes" 1>&2
        rm -f /tmp/contents
        exit 1
}
if [ -s $changes ]
then
   find /var/sadm -xdev \( -perm -020 -o -perm -002 \) -print >> $changes
   xargs chmod og-w < $changes
```

```
    touch /var/adm/messages
    chmod og-w /var/adm/messages
    set -e
    # Always save old contents file. The very first one becomes increasingly
    # out of date. Previously, we saved the very first contents file.
    rm -f /var/sadm/install/contents.org.Z
    mv /var/sadm/install/contents /var/sadm/install/contents.org
    compress /var/sadm/install/contents.org
    mv /tmp/contents /var/sadm/install/contents
    chmod 644 /var/sadm/install/contents
else
    rm -f /tmp/contents
fi
# Our install scripts created these with owner root.
if [ ! -f /var/lp/logs/lpNet ]
then
    touch /var/lp/logs/lpNet /var/lp/logs/lpsched
fi
chown lp /var/lp/logs/lpNet /var/lp/logs/lpsched
#-------------------------------------------------------------------------------
```

B.15 Modelo para instalação de patch

```
#------------------------------------------------------------------------
#!/usr/bin/env perl
#
# $Id: fastpatch.pl,v 1.21 1996/05/31 11:50:11 casper Exp casper $
#
# A faster install patch script than the one Sun has given us.
#
# Objectives: make a (non-saving) installpatch script that leaves the
# system in a state like the standard installpatch procedure does.
# But does it much quicker.
#
# This script should only be used on initial install: those installs
# that can be re-done quickly using the original installpatch.
#
# Explanation of options:
#
#       -d      debug (show what you would do).
#       -r      remove outdated patchdata that existed before the
#               current patches are nstalled from the system.
#       -R dir  chroot to dir. (Other arguments interpreted relatve to dir
#               as well, used for Jumpstart installs with -R /a).
#       -D dir  Use patch info from dir (directory *must* end in valid
#               patchid), used to obsolete old patches before new ones
#               are installed.
#       -p dir  Look for patch.tar.Z files or unpacked patches in dir.
#       -o      Don't install patches, only remove obsoleted patches.
#       -i      Install patches w/ installpatch.
#       -n      If there are no applicable patches *don't* use installpatch.
#       -I      Ignore backoutpatch failures.
#       -b      Don't use backoutpatch when applicable
#       -c      Don't use "chroot()", instead try passing "-R rootdir"
#
# Casper Dik (casper@fwi.uva.nl)
#

#
# Site specific constants.
#
# Where to get patches, "/`uname -r`" is appended.
umask(022);
($patchdir = $0) =~ s:[^/]*$:./patches:;
chop($patchdir=`cd $patchdir;pwd`);

$adminfile = "/tmp/admin.$$";

$0 =~ s:.*/::;
```

```
# We want to be able to have just perl in a bootstrapping env.
# no perl lib dir.
#require 'getopts.pl';
#
#if (!&Getopts('dr')) {
#    print STDERR "Usage: $0 [-d] [-r] [ patch ... ]\n";
#    exit 1;
#}

$opt_r = 0;
$errors = 0;
@extradirs = ();
#
# Determine where to get patches from.
#
chop($osrev=`uname -r`);
$patchdir .= "/" . $osrev;
@patchdirs = ("/tmp", $patchdir, "$patchdir/unpacked", "$patchdir/alt");

while ($#ARGV > -1 && ($_ = $ARGV[0]) =~ /^-/) {
    if (&simple_opt('d')) {
    } elsif (&simple_opt('r')) {
    } elsif (&simple_opt('I')) {
    } elsif (&simple_opt('b')) {
    } elsif (&simple_opt('o')) {
    } elsif (&simple_opt('n')) {
    } elsif (&simple_opt('i')) {
    } elsif (&simple_opt('c')) {
    } elsif (/^-R/) {
        $rootdir = &arg_opt('R');
    } elsif (/^-p/) {
        push(@patchdirs,&arg_opt('p'));
    } elsif (/^-D/) {
        push(@extradirs,&arg_opt('D'));
    } else {
        die "Usage: $0 [-crIboni] [-D patchdir] [-R rootdir ] [ patch ... ]\n";
    }
    shift if ($ARGV[0] =~ /^-$/);
}

#$opt_d = 1;

# For .pre and .post scripts.
$ENV{'pdir'}=$patchdir;
$ENV{'PATH'}="/usr/sbin:/usr/bin:/etc:/sbin:/usr/local/bin";

select(STDOUT); $| = 1;
select(STDERR); $| = 1;

#
```

```
# Chroot now. -- should use -R
#
$nochroot = defined($opt_c);
$root = $rarg = "";
if (defined($rootdir)) {
   if ($nochroot) {
         $rarg = "-R $rootdir";
         $root = $rootdir;
         # This should catch all missing -R options.
         $ENV{'PKG_INSTALL_ROOT'} = $rootdir;
   } else {
         chdir($rootdir) || die "$0: can't chdir to \"$rootdir\": $!\n";
         chroot($rootdir) || die "$0: can't chroot to \"$rootdir\": $!\n";
         grep($_ =~ s:^$rootdir::,@patchdirs);
         $patchdir =~ s:^$rootdir::;
   }
}

#
# Don't remove already outdated files unless -r is specfied.
# First update internal patch database.
#
&find_and_make_obsolete("$root/var/sadm/pkg", !$opt_r);

foreach $d (@extradirs) {
   ($current_patch = $d) =~ s:^.*/::;
   &find_and_make_obsolete($d, 0);
}

mkdir ("$root/var/sadm/patch", 0755)
   if (! -d "$root/var/sadm/patch" && !defined($opt_d));

for (@ARGV) {

   ($base,$rev) = split('-',$_);

   # Patch older than one installed, or installed already.
   next if (defined($revs{$base}) && $rev <= $revs{$base});

   # Patch obsoleted
   if (defined($obs{$base})) {
         warn "$0: $_ is obsolete\n";
         next;
   }

   undef $pdir;

   foreach $d (@patchdirs) {
         if (-d "$d/$_") {
            $pdir = "$d/$_";
            last;
```

```
        }
}

if (!defined $pdir) {
        undef $ptar;
        foreach $d (@patchdirs) {
           $f = "$d/$_.tar.Z";
           if (-f "$f") {
                $ptar=$f;
                last;
           }
        }
        warn ("$0: cannot find patch $_\n"),next unless defined($ptar);
        $pdir="/tmp/$_";
        system("zcat $ptar | (cd /tmp; tar xfB -) 2> /dev/null");
        warn ("$0: cannot find patch $_\n"), next unless -d $pdir;
        $rmit=$pdir;
}

print "Processing patch $_\n";
#
# Remove the patches obsoleted by the new patch.
#
$pname = $current_patch = $_;
&find_and_make_obsolete($pdir, 0);

# If it is a new patch type, skip it.
if ($newpatch{$current_patch}) {
        warn "Patch \"$pname\" uses new installpatch -- can't fastpatch\n";
}

# Skip patch installation, if asked. For obsoleting
next if ($opt_o);

#
# Install the new patch.
#
@patch_packages = ();
@from_patch = split('\s+',$pkg{$current_patch});

foreach $p (@from_patch) {
        push(@patch_packages,$p) if (&basedir($current_patch,$p));
}

$ptarget = "$root/var/sadm/patch/$current_patch";
#
# If we can't find the packages, use installpatch.
#
if ($#patch_packages == -1 || defined($opt_i) ||
          defined($newpatch{$current_patch})) {
        if (defined($opt_n)) {
```

```
            warn "$0: $current_patch: skipped\n";
            next;
        }
        warn "$0: $current_patch: no applicable packages found, trying installpatch\n"
            unless defined($opt_i) || defined($newpatch{$current_patch});
        if (defined($opt_d)) {
            print "$pdir/installpatch $rarg $pdir\n";
        } else {
            $exit = system "$pdir/installpatch -u -d $rarg $pdir";
            # No applicable patches (exit 8), success or client system.
            if ($exit/256 == 8) {
                warn "$0: installpatch found no applicable packages\n";
            } elsif ($exit != 0) {
                warn "$0: installpatch failed for $current_patch ($exit)\n";
                $errors++;
            }
        }
        next;
}

#
# Using pkgadd.
#
if (defined($opt_d)) {
        print "#Would add patch with following commands:\n";
} else {
        mkdir($ptarget, 0755);
        system("cp $pdir/README.$current_patch $ptarget");
        system "sh -c '. $patchdir/$pname.pre'"
            if (-f "$patchdir/$pname.pre");
}

print "Patching packages: @patch_packages\n";

foreach $p (@patch_packages) {
        if (!defined($opt_d)) {
            &make_pkgadd_adminfile(&basedir($current_patch,$p));
            mkdir("$ptarget/$p", 0755);
            system("cp $pdir/$p/pkginfo $pdir/$p/pkgmap $ptarget/$p");
            if (system("pkgadd $rarg -S -a $adminfile -n -d $pdir $p")) {
                warn "$0: pkgadd failure for pkg $p of $current_patch\n";
                $errors ++;
                last;
            }
        } else {
            print "$p has basedir ", &basedir($current_patch,$p), "\n";
            print "pkgadd $rarg -S -a $adminfile -n -d $pdir $p\n";
        }
}

if (!defined($opt_d)) {
```

```
            system "sh -c '. $patchdir/$pname.post'"
               if (-f "$patchdir/$pname.post");
      }
   } continue {
      system("rm -rf $rmit")
            if (defined($rmit));
      undef $rmit;
   }

   unlink ($adminfile);
   exit($errors != 0);

   #
   # Subroutines.
   #

   sub find_and_make_obsolete {
      local($patch);
      &find_obsolete($_[0]);

      for $patch (keys(%pkg)) {
            ($base, $rev) = split('-', $patch);
            &obsolete($patch, $_[1])
               if ($rev < $revs{$base} || defined($obs{$base}));
      }
   }

   sub obsolete {
      local($patch, $dont) = @_;
      local($exit);
      local($pdir) = "$root/var/sadm/patch/$patch";

      if (!defined($obsolete{$patch})) {
            print("obsolete: $patch\n");
            $obsolete{$patch} = 1;
            return if ($dont);
            if (!defined($opt_d)) {
               print "Removing: $patch\n";
               if (!defined($opt_b) && -f "$pdir/backoutpatch") {
                     $exit = system("$pdir/backoutpatch $rarg $patch");
                     if ($exit != 0 && !defined($opt_I)) {
                        die "$0: backoutpath failure for $patch\n";
                     }
               }
               if (-d "$pdir") {
                     die "fatal: removing base package\n" if ($pkg{$patch} !~ /\./);
                     &make_pkgrm_adminfile;
                     system "pkgrm $rarg -a $adminfile -n $pkg{$patch}" ||
                        die "$0: backoutpath failure for $patch\n";
                     system "rm -rf $pdir";
               }
```

```
        } else {
            print "#$patch would be removed with the following commands:\n";
            if (!defined($opt_b) && -f "$pdir/backoutpatch") {
                print "$pdir/backoutpatch $rarg $patch\n";
            } else {
                warn "fatal: remove base package\n" if ($pkg{$patch} !~ /\./);
                print "pkgrm $rarg -a $adminfile -n $pkg{$patch}\n";
                print "rm -rf $pdir\n";
            }
        }
    }
}

sub find_obsolete {
    local($pkgdir) = @_;
    local($basedir, $dir, $t, $f, $pkg, $base, $rev, $thisbase, $patch, $index);

    $current_patch =~ s/^T// # Test patch
        if (defined($current_patch));

    $prefix = defined($current_patch) ? $current_patch : "none";
    $dir = $pkgdir . "/*/pkginfo";
    for $f (<${dir}>) {
        undef $patch;
        open(PKG,"<$f") || die "$f: $!\n";
        ($pkg = $f) =~ s:^$pkgdir/(.*)/pkginfo$:$1:;
        $index = $prefix.$pkg;
        while (<PKG>) {
            chop;
            s/[\t ]*=[\t ]*/=/;
            if (/^SUNW_PATCHID=/) {
                $patch = $';
                die "$0: Patch id $patch isn't expected: $current_patch\n"
                    if (defined($current_patch) && $current_patch ne $patch);
                $thisbase = &addpatch($patch,$pkg);
            } elsif (/^SUNW_OBSOLETES=/) {
                for $p (split(',',$')) {
                    ($base,$rev) = split('-', $p);
                    # Some patches have themselves in the obsolete
                    # bit. That isn't right but there you are.
                    # This check depends on SUNW_PATCHID to come before
                    # SUNW_OBSOLETES
                    if ($base eq $thisbase) {
                        warn "$thisbase wants to obsolete self: $p\n";
                    } else {
                        $obs{$base} = "";
                    }
                }
            } elsif (/^PATCHLIST=/) {
                local(@plist) = split(/\s+/,$');
                local($p);
```

```
                foreach $p (@plist) {
                    &addpatch($p, $pkg);
                }
            } elsif (/^VERSION=/) {
                $t = $';
                # The new patches use a different installation scheme.
                unless ($t =~ s/,PATCH=.*$//) {
                    $newpatch{$prefix} = 1;
                }
                $version{$index} = $t;
            } elsif (/^ARCH=/) {
                $arch{$index} = $';
            } elsif (/^PKG=/) {
                $realpkg{$index} = $';
            } elsif (/^BASEDIR=/) {
                $basedir = $';
            }
        }
        $thisbase = "";
        $basedir{$realpkg{$index},$arch{$index},$version{$index}} = $basedir
            unless (defined($current_patch));
        close(PKG);
    }
}

sub make_pkgadd_adminfile {
open(PATCHADMIN,">$adminfile");

print PATCHADMIN <<EOF ;
mail=
instance=unique
partial=nocheck
runlevel=nocheck
idepend=nocheck
rdepend=nocheck
space=quit
setuid=nocheck
conflict=nocheck
action=nocheck
basedir=$_[0]
EOF

close(PATCHADMIN);
}

sub make_pkgrm_adminfile {
open(PATCHADMIN,">$adminfile");

print PATCHADMIN <<EOF ;
mail=
instance=unique
```

```
partial=nocheck
runlevel=nocheck
idepend=quit
rdepend=quit
space=quit
setuid=nocheck
conflict=nocheck
action=nocheck
basedir=default
EOF

close(PATCHADMIN);
}

sub simple_opt {
   if ($ARGV[0] =~ s/^-$_[0]/-/) {
         eval "\$opt_$_[0] = 1";
   } else {
         0;
   }
}

sub arg_opt {
   $ARGV[0] =~ s/-$_[0]//;
   shift @ARGV if (length($') == 0);
   defined($ARGV[0]) || die "$0: -$_[0] requires argument\n";
   shift @ARGV;
}

sub basedir {
         local($patch, $pkg) = @_;
         local ($index) = $patch.$pkg;

         $basedir{$realpkg{$index}, $arch{$index}, $version{$index}};
}

sub addpatch {
   local($patch,$pkg) = @_;
   local($base,$rev) =  split(/-/,$patch);

   ($base,$rev) = split('-', $patch);
   $revs{$base} = $rev
         if (!defined($revs{$base}) || $revs{$base} < $rev);
   if (defined($pkg{$patch})) {
         $pkg{$patch} .= " " . $pkg;
   } else {
         $pkg{$patch} = $pkg;
   }
   $base;
}
#-------------------------------------------------------------------------
```

Apêndice C

Exemplos de wrappers para aplicativos

Neste apêndice estão exemplificados exemplos de wrappers que solucionam o problema de controle de versões no ambiente Unix.

Exemplo de wrapper foi escrito em C Shell e o usuário pode escolhe a versão do aplicativo do WdX a ser executado:

```
# Wrapper: WdXV1.9 – Define ambiente de execução do aplicativo WdX versão 1.9,
# criado em 05-07-2000 por CAS-Fenix.
#
# Define Shell como sendo C Shell
!/bin/csh –f
        # Define o diretorio de execução
        cd /usr/local/stow/WdX1.9/bin
        # Define variavel de ambiente do programa WdX
        setenv APPHOME /usr/local/stow/WdX1.9/bin/CalcX
        # Executa o programa
        exec ${EXEC} "${@}"
```

Exemplo de wrapper para o aplicativo de desenho Vx para versão 2.1, e tem vários binários disponíveis.

```
# Wrapper: ExecVx2.1 – Define ambiente de execução do aplicativo Vx versão 2.1, criado
# em 14-01-2000 por CAS-Fenix.
#
# Define Shell como sendo Bourne Shell
!/bin/sh
        # Obtem o comando a ser executado atraves do programa basename,
        # que armazena o valor na variável CMD
        CMD=`basename $0
        # Obtem o diretório atual do usuário através do programa dirname,
        # que armazena o valor na variável DIR
        DIR=`dirname $0`
        # Define a variável de ambiente EXEC como null
        EXEC=
        # Loop uma lista de binários disponível para este programa Vx
        for isa in `/usr/bin/isalist`; do
          # Verifica se existe o binario para este programa
          if [-x ${DIR}/${CMD}]; then
                # Define a correta variavel de ambiente para EXEC
                EXEC=${DIR}/${isa}/${CMD}
                # Encerra loop
                break
          fi
        done
        # Verifica de EXEC não null
        if [-z "${EXEC}"]; then
          # Mensagem de não existe executavel para esta arquitetura
          echo  1>&2 "$0: não existe executavel para esta arquitetura (binário)"
          # Encerra Wrapper
          exit 1
        fi
        # Executa versão correta do binario para o programa Vx
        exec ${EXEC} "${@}"
```

Exemplo de wrapper para o aplicativo de bringover versão 1.3 que copia a um arquivo para o diretório atual para correta execução do aplicativo, e remove o arquivo no termino do wrapper.

```
# Wrapper: bringoverV1.3 – Define ambiente de execução do aplicativo bringover versão 1.3,
# criado em 09-04-2001 por CAS-Fenix.
#
# Define Shell como sendo Bourne Shell
!/bin/sh
        # Obtem o comando a ser executado atraves do programa basename,
        # que armazena o valor na variável progname
        progname=`basename $0
        # Define o nome do arquivo a ser copiado para a variável de ambiente filename
        filename="/usr/lib/speclib.a"
        # Copia o arquivo para o diretório atual
        cp –p $filename .
        # Envia mensagem que executou a copia
        echo "$filename foi copiado para o corrente diretório"
        # Verifica a existência do programa bringover
        if [-f $progname ]
          then
                eval exec /cmgr/$progname $ opts "'$@'"
          else
                echo >&2 "Desculpe, /cmgr/$progname não foi encontrado."
                #Remover arquivo copiado
                rm –p $filename
                # Encerra Wrapper
                exit 1
        fi
        #Remover arquivo copiado
        rm –p $filename
        # Encerrar wrapper
        exit
```

Exemplo de wrapper foi escrito em C Shell e o usuário pode escolhe a versão do aplicativo do CalcX a ser executado:

```
# Wrapper: CalcXV2.0 – Define ambiente de execução  do aplicativo CalcX versão 2.0, criado
# em 05-07-2000 por CAS-Fenix.
#
# Define Shell como sendo C Shell
!/bin/csh –f
main:
        # Envia mensagem para o usuario escolher a versão do aplicativo
        echo "Seleção da versão do Aplicativo CalcX"
        echo ""
        echo "1. Versão 1.5 (sem função exponencial)"
        echo "2. Versão 2.0 (com função exponencial)"
        echo "3. Versão 2.2 (beta ➔ teste)"
        echo ""
        echo –n "Digite a opção e pressione ENTER"
        # Registra a escolha
        set choice = $<
        # Verifica a escolha (se valida)
        if ( $choice !~ [1-2] ) then
                # Nao valida -> retorna para escolher novamente
                goto main
        endif
        # Define ambiente para cada escolha
        switch ($choice)
                case "1":
                        # Define ambiente para escolha da versão 1.5
                        cd /usr/local/stow/CalcX1.5/bin
                        setenv APPHOME /usr/local/stow/CalcX1.5/bin/CalcX
                        breaksw
                case "2":
                        # Define ambiente para escolha da versão 2.0
                        cd /usr/local/stow/CalcX2.0/bin
                        setenv APPHOME /usr/local/stow/CalcX2.0/bin/CalcX
                        breaksw
                case "3":
                        # Define ambiente para escolha da versão 2.2
                        cd /usr/local/stow/CalcX2.2/bin
                        setenv APPHOME /usr/local/stow/CalcX2.2/bin/CalcX
        endsw
# Executa versão escolhida
exec ${EXEC} "${@}"
```

Bibliografia

[01] Anais 13º SBRC - Simpósio Brasileiro de Redes de Computadores. Departamento de Ciência da Computação, UFMG, 1995.

[02] Anais 2º Wais - 2º Workshop sobre Administração e Integração de Sistemas. Fortaleza, CE, UFCE, 1996.

[03] Thomas S. Gregory, Shreder O. James, Orcutt E. Merrilee, Johnson C. Desiree, Simmelink T. Jeffrey, and Moore P. John. Unix Host Administration in heterogeneous distributed Computing environment. Proceedings of the USENIX Systems Administration (LISA X) Conference, pages 43-50, Chicago, Illinous, October 4, 1996.

[04] Bill Rosenblatt. Learning the KornShell, O'Reilly & Associates, Inc, 1993.

[05] Paul Anderson. Managing Program Binaries in a heterogeneos unix network. Proceedings of the USENIX Systems Administration (LISA V) Conference, pages 1-9, San Diego, California, Septmber 30, 1991.

[06] Brain L. Wong. Configuration and Capacity Planning for Solaris Servers, Sun Microsystems Press, ISBN 0-13-349952-9, 1997.

[07] Rémy Evard. An Analysis of unix system configuration. Proceedings of the USENIX Systems Administration (LISA IX) Conference, pages 179-193, San Diego, California, October 26-31, 1997.

[08] Bourne S. R.. The Unix System, Addison-Wesley Publishing Company, ISBN 0-201-13791-7, 1983.

[09] Calvin R. Gaisford. Caldera Volution – System Management and Administration Software. 2001. Disponível em: http://www.caldera.com/volution/volution.html.

[10] Paul Anderson. Towards a high-level machine configuration system. Proceedings of the USENIX Systems Administration (LISA VIII) Conference, pages 19-26, San Diego, California, September 19-23, 1994.

[11] Walter Wong, and David Markley. Carpe: The software maintainer's guide, em CMU. Computing Services, Carnegie Mellon University. 1998.

[12] Colyer Wallace, Mark Held, David Markley, and Walter Wong. Software Management in the Andrew System, AFS User's Group Proceedings. Spring 1992.

[13] Wallace Colyer, and Walter Wong. Depot: A Tool for Managing Software Environments, LISA VI Proceedings, pages 153-162, 1992

[14] Mark Held. Depot: A Tool for Managing Software Environments., em CMU. Computing Services, Carnegie Mellon University. 1997.

[15] George Becker, Mary E. S. Moris, and Kathy Slattery. Solaris Implementation: A guide for system Administration, Sun Microsystems Press, ISBN 0-13-353350-6, 1995.

[16] Ken Yap. EtherBoot – Make Boot ROMS, Projeto Gnu, 1999. Disponível em http://www.gnu.org/software/etherboot.htms.

[17] Mark Held, and Dawn Neuhart. Software Management in the Andrew Distributed UNIX System, em CMU, Computing Services, Carnegie Mellon University. 1996.

[18] Janice Winsor. Solaris System Administration Guide, Sun Microsystems Press, ISBN 1-56276-080-7, 1992.

[19] Janice Winsor. Solaris: Advanced System Administrator's Guide, Sun Microsystems Press, ISBN 1-56276-131-5, 1998.

[20] Kenneth Manheimer, Barry Warsaw, Stephen Clark, and Walter Rowe. The Depot: A Framework for Sharing Software Installation Across Organizational and Unix Platform Boundaries". LISA IV Proceedings. pages 37-46., 1990.

[21] Morris I. Bolsky, and David G. Korn. The Korn Shell - Command and Programming Language, Prentice-Hall, ISBN 0-13-516972-0, 1993.

[22] Paulo Licio de Geus. An overview of the computing facilities at, DCC-IMECC-UNICAMP, I WASH, Recife, 1994.

[23] Paul Anthony Kasper, and Alan L. McClellan. Automating Solaris Installations: a Custom Jumpstart Guide, Sunsoft Press, ISBN 0-13-312505-X, 1995.

[24] Red Hat Inc.. Red Hat Package Manager (RPM), Red Hat Publishing, 1998. Disponível em http://www.rpm.org.

[25] Lee Leonardo Amatangelo. Unleashing the power of jumpstart: a new technique for disaster recovery, cloning, or snapshotting a Solaris System. Proceedings of the USENIX Systems Administration (LISA XIV) Conference, pages 219-228, New Orleans, LA, December 3-8, 2000.

[26] Tina Dar Mohray. Job Descrptions for Sysadminis, Usenix Association, ISBN1-880446-57-X, 1993.

[27] Michael E. Shaddock., Michael C. Mitchell, and Helen E. Harrison. How to upgrade 1500 workstation on Saturday, and still have time to mow the yard on Sunday, Proceedings of the USENIX Systems Administration (LISA IX) Conference, pages 59-65, Monterey, California, Septmber 18-22, 1995.

[28] Salim Douba. Networking Unix, Sams publishing, ISBN 0-672-30584-4, 1995.

[29] R Sandberg. The Sun Network File System: Design, Implementation and Experience, Published SunSoft Press, 1994.

[30] M. Satyanarayanan, , J. H. Howard; D. A. Nichols; N. Sidebotham, and A. Z. Spector. The ITC Distributed File System: Principles and Design. Proceedings of the 10th ACM Symposium on Operating System Principles. 1985.

[31] Bjorn Satdeva. Methods for maintaining one source tree in a heterogeneous environment. Proceedings of the USENIX Systems Administration (LISA VII) Conference, pages 57-65, Monterey, California, November 1-5, 1993.

[32] Jason Heiss. Enterprise Rollouts with Jumpstart. Proceedings of the USENIX Systems Administration (LISA XIII) Conference, Seattle, Washington, November 7-12, 1999.

[33] C. Walter Wong. Local Disk Depot – Customizing the Software Environment. Proceedings of the USENIX Systems Administration (LISA VII) Conference, pages 49-53, Monterey, CA, November 1-5, 1993.

[34] P. John Rouillard, and Martin B. Richard. Depot-Lite: a mechanism for managing software. Proceedings of the USENIX Systems Administration (LISA VIII) Conference, pages 83-91, San Diego, CA, September 19-23, 1994.

[35] Michael E. Shaddock, Mike C. Mitchell, and, Helen E. Harrison. Pong: a flexible network services monitoring system. Proceedings of the USENIX Systems Administration (LISA VIII) Conference, pages 167-173, San Diego, California, Septmber 19-23, 1994.

[36] Todd Miller, Christopher Stirlen, e Nemeth Evi. Satool – a system administrator's cockpit, na implementation. Proceedings of the USENIX Systems Administration (LISA VII) Conference, pages 119-129, Monterey, CA, November 1-5, 1993.

[37] R. N. Sidebotham. Volumes: The Andrew File System Data Structuring Primitive. Technical Report CMU-ITC-053. Information Technology Center, Carnegie Mellon University. 1986.

[38] Michael T. Stolarchuk. Faster AFS, AFS, User's Group Proceedings. Spring 1992.

[39] Carl Staelin. mkpkg: a software packaging tool. Proceedings of the USENIX Systems Administration (LISA XII) Conference, pages 243-252, Boston, MA, December 6-11, 1998.

[40] B. Glickstein. Stow Manual, Zanshin Software Inc., 1997. Disponível em http://www.gnu.org/software/stow.

[41] Sun Microsystems. Network and File Servers: a performance tuning guide, SunSoft, 1990. Disponível em http://docs.sun.com.

[42] Sun Microsystems. SunOS 5.1 Administering NFS and RFS, SunSoft, 1992. Disponível em http://docs.sun.com.

[43] Sun Microsystems. SunOS 5.1 Administering NIS and DNS, SunSoft, 1992. Disponível em http://docs.sun.com.

[44] Sun Microsystems. SPARC: Architecture Manual, version 9, Prentice-Hall, ISBN-0-13-099227-5, 1994. Disponível em http://docs.sun.com.

[45] Sun Microsystems. The Network Information Service, in System and Network administration, 1995. Disponível em http://docs.sun.com.

[46] Carl Hauser. Speeding up Unix login by caching the initial environment. Proceedings of the USENIX Systems Administration (LISA VII) Conference, pages 117-124, San Diegon, CA, September 19-23, 1994.

[47] IBM. An Introduction to Tivoli Enterprise, SG24-5494-00, 2001. Disponível em http://www.tivoli.com.

[48] Computer Associates. Unicenter, 2001. Disponível em http://www.cai.com.

[49] Tobias Oetiker. SEPP – Software installation and sharing system. Proceedings of the USENIX Systems Administration (LISA XII) Conference, pages 253-259, Boston, MA, December 6-11, 1998.

[50] David Vanryzin. Flexible Administration for AFS,. Proceedings of the Fall AFS Users Group. 1992

[51] John Lockard, and Jason Larke. Synctree for single point installation, upgrades, and OS patches. Proceedings of the USENIX Systems Administration (LISA X) Conference, pages 261-193, Boston, MA, December 6-11, 1998.

[52] Larry Wall, and Randal L. Schwartz. Programming perl. O'Reilly and Associates, Inc. 1995.