RemX - Uma Ferramenta para Execução de Aplicações Remotas voltada para a Integração de Ambientes Multi-Plataformas

Roberto Cesar Serra Cordeiro[*]

Rodolfo Jardim de Azevedo[**]

João Paulo Andrade Almeida[**]

Departamento de Informática
Universidade Federal do Espírito Santo
Av. Fernando Ferrari S/N, Campus Goiabeiras, CT/DI
29060-900, Vitória, ES

rcesar@inf.ufes.br, rodolfo@inf.ufes.br, jpaulo@inf.ufes.br

Resumo

A diversidade de plataformas computacionais em um ambiente acadêmico tende a isolar grupos de usuários em seus ambientes operacionais, não permitindo que estes usufruam de outros recursos disponíveis em outros sistemas. Por outro lado, a migração de usuários de um ambiente para outro é problemática, pois estes apresentam certa resistência ao aprendizado de um novo conjunto de aplicativos. O objetivo do **RemX** é permitir que um usuário possa executar os aplicativos de seu ambiente operacional em qualquer outro ambiente que ofereça X-Windows, com a vantagem de também poder executar os aplicativos locais, tudo isso de forma transparente. O **RemX** também pode fazer o balanceamento de carga atribuindo a execução da aplicação à máquina com melhor condição de executá-la.

Abstract

The diversity of computer platforms in an academic environment tends to isolate user groups in their operating environments, preventing them from making use of the resources available in other systems. By the other hand, the migration of users from an environment to another is usually problematic because they offer resistance in learning a new set of applications. The aim of **RemX** is to allow a user to run applications from his operating environment in any other using X-Windows, with the advantage of also running local applications, in a transparent way to this user. The **RemX** can also perform load balancing, assigning the execution of remote applications to the less loaded workstation.

---

* Professor assistente do Departamento de Informática da UFES
** Bolsista PET da Engenharia de Computação da UFES

1 Introdução

Um ambiente acadêmico tende a ter uma grande diversidade de plataformas computacionais (*hardware* e *software*) devido a vários fatores, tais como: rápido surgimento de novas tecnologias, doação dos órgãos fomentadores de pesquisa, doação de convênios, mudança de fornecedores em diferentes processos de licitação, etc.

As estações de trabalho com arquitetura RISC, apesar de virem equipadas com um sistema operacional *UNIX-like*, apresentam diferenças bastante significativas em suas interfaces gráficas, assim como no conjunto de comandos de seus diversos *shells*. As Interfaces gráficas diferenciam-se significativamente de acordo com o seu padrão, *Motif* ou *OPEN LOOK*. Além disso, cada fabricante desenvolve os seus próprios aplicativos para edição de texto, correio eletrônico, ambiente de desenvolvimento de programa, etc., muitas vezes bem diferentes entre si.

A aquisição de novas plataformas computacionais tende a isolar os usuários em três grupos distintos: os que resistem a aprender a nova interface e continuam usando a plataforma que dominam; os que migram direto para a nova plataforma, abandonando totalmente a anterior; os que passam a usar as duas plataforma simultaneamente, usufruindo de uma nova tecnologia mas ainda se valendo do ambiente no qual já possui experiência ou programas desenvolvidos.

O isolamento dos usuários nestes grupos causa certos transtornos quanto a utilização das máquinas em um laboratório. Por exemplo, se todas as máquinas de uma mesma plataforma estão ocupadas, um usuário deste mesmo grupo não terá um posto de trabalho disponível, mesmo que haja várias máquinas das outras plataformas ociosas.

Existem várias soluções para integrar estes dois sistemas. Poderia ser dado acesso a todos os usuários em todas as plataformas, ou então colocar todas as máquinas no mesmo domínio de NIS[1]. Ainda mais, cada usuário poderia simplesmente exportar o seu ambiente para a máquina na qual estivesse trabalhando. Todas estas soluções apresentam seus inconvenientes que serão abordados adiante.

O **RemX** é uma ferramenta gráfica que permite ao usuário selecionar qual aplicação ele deseja executar e em qual plataforma. O **RemX** se encarrega de se conectar a uma das estações desta plataforma, executar este aplicativo e exportar sua saída gráfica para o vídeo da máquina em que este usuário está trabalhando, tudo isso de forma totalmente transparente. Melhor ainda, o **RemX** consulta as máquinas da plataforma escolhida, se desejado, e passa a execução do aplicativo para a máquina que estiver menos carregada naquele momento, fazendo assim um balanceamento de carga.

2 Soluções para Integração de Ambientes UNIX

Uma das soluções usadas para integração de sistemas é o NIS que permite o compartilhamento de informações entre estações de trabalho inseridas em um domínio. Uma estação é configurada como sendo o NIS MASTER e será responsável por manter as informações que se desejam compartilhar. Cada estação do domínio NIS que queira obter as informações compartilhadas deverá requisitá-las ao NIS SERVER.

[1] NIS - Network Information System

Uma das grandes facilidades do NIS (e a mais usada) é poder compartilhar os arquivos de autenticação de usuários. Assim, quando um usuário é cadastrado, suas informações são inseridas apenas uma vez no NIS MASTER. Este usuário poderá então usar qualquer máquina deste domínio NIS, haja visto que todas tem acesso ao arquivo de autenticação de usuários. O usuário estará cadastrado no domínio NIS e não apenas numa máquina específica.

De forma a criar um ambiente de trabalho único em todas as máquinas de um domínio NIS, é preciso colocar disponível aos usuários o seu diretório de trabalho (*home directory*) e os aplicativos que ele usa normalmente. Melhor falando, tanto o diretório do usuário quanto o diretório dos aplicativos mais usados precisam ser vistos de qualquer máquina do domínio NIS. De forma a fazer estes diretórios disponíveis é necessário o uso do sistema de arquivos de rede (NFS[2]), como mostra a figura 2.1.

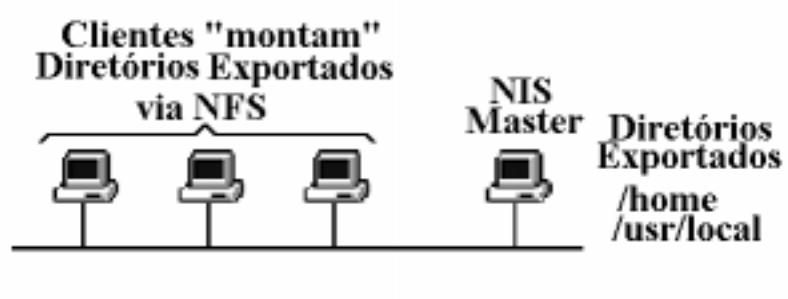

Figura 2.1. Domínio NIS. Um usuário neste domínio pode trabalhar em qualquer estação, tendo sempre disponível o seu diretório *home*

O NFS usa o conceito de "montagem" do sistema de arquivos do UNIX, onde um disco pode ser particionado em vários *file systems* e cada um deles é montado nos seus respectivos pontos de montagem, formando uma única árvore de diretórios. Com o NFS, no entanto, um *file system* pode estar numa máquina remota. Todo acesso ao *file system* remoto é tratado pelo NFS dando a transparência ao usuário como se o *file system* fosse local.

O usuário ao usar uma máquina qualquer num domínio NIS estará executando todos os seus processos nesta máquina, apesar de seu diretório *home* e diretório de aplicativos poderem estar em outra máquina. Portanto o NIS+NFS não se apresenta como uma boa solução quando existem máquinas de plataformas diferentes no domínio, como mostrado na figura 2.2.

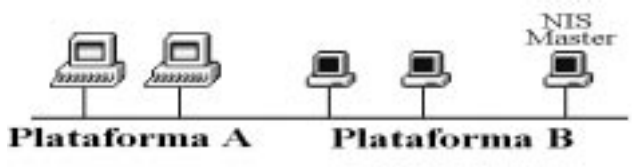

Figura 2.2. Domínio NIS com máquinas de plataformas diferentes

Suponha que um usuário desenvolveu seus programas e os compilou numa máquina da plataforma A, deixando os executáveis em seu diretório *home*. Se este usuário usar uma máquina da plataforma B ele terá os seus programas disponíveis, só que não poderá executá-los, visto que o seu código é incompatível com o processador e sistema operacional da plataforma B. Outro problema sério desta solução é que a interface com o usuário será diferente nas diversas plataformas, mesmo que sejam executados os mesmos tipos de aplicativos. Por exemplo, uma gerenciador de arquivos da plataforma A, apesar de poder

[2] NFS - Network File System

executar todas as funções que o gerenciador de arquivos da plataforma B, certamente terá uma interface com o usuário bastante diferente deste último, o que por si só já é um sério motivo para o não uso por parte de alguns usuários.

De forma a transpor este problema, o usuário vê-se obrigado a abrir seções `telnet` de B para A e executar seus programas dentro destas seções. Esta solução funciona bem enquanto não é preciso que o aplicativo executado remotamente exiba uma saída gráfica. Se for preciso, a máquina local (da plataforma B) precisa ter uma interface X-Windows.

A interface X-Windows permite que um aplicativo executado remotamente exiba sua saída gráfica localmente. No entanto, para que isto aconteça é preciso que o usuário siga o seguinte procedimento: primeiro, dê a permissão na sua interface X-Windows para receber a saída gráfica da máquina onde ele deseja executar o aplicativo. Depois precisa abrir uma janela de comando na máquina remota (via `telnet`) configurando a mesma para exportar a sua saída para a máquina local. Por último, iniciar o aplicativo remoto. Este procedimento não é muito natural devido à complexidade da sintaxe dos comandos envolvidos, o que inibe a maioria dos usuários de usá-lo. A figura 2.3 exemplifica.

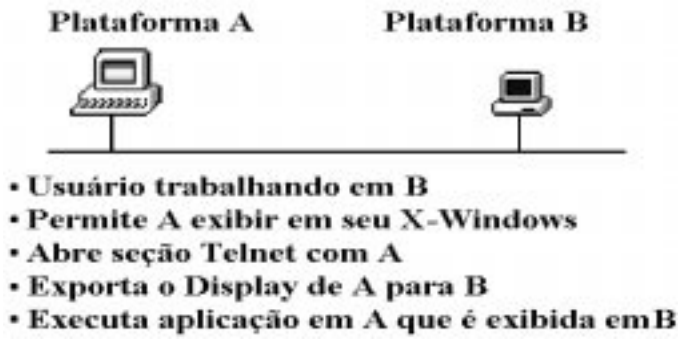

Figura 2.3. Procedimento para utilização da interface X-Windows

O processo de executar um aplicativo remotamente com a exibição gráfica na máquina local é bastante facilitada pelo uso do comando `xrsh`, presente na distribuição X11R6 do X-Windows. Através deste comando o usuário pode determinar qual o aplicativo ele deseja executar remotamente e em qual estação deverá ser executado. No entanto, o aplicativo remoto é executado com os direitos do usuário remoto cujo nome é o mesmo do usuário local. Por exemplo, se o usuário *joaquim* numa máquina da plataforma B pedir a execução de um aplicativo numa máquina da plataforma A usando o `xrsh`, este aplicativo será executado apenas se existir um usuário *joaquim* em A e se a máquina de B estiver no *host.equiv* da máquina de A, ou seja, não é feita verificação de *password* do usuário.

Assim como todos os aplicativos da família "r" (rlogin, rshel, rwho, etc), o `xrsh` apresenta uma falha grave de segurança. Se algum intruso tiver acesso ao nível de enlace de uma rede local ele poderá forjar a existência de uma máquina do domínio, criar usuários com os mesmos nomes existentes nas estações do domínio e ter acesso irrestrito a todos os arquivos destes usuários. Por este motivo vários administradores de rede desabilitam todos os aplicativos da família "r" de seus domínios.

Uma outra forma de integração seria exportar o gerenciador do *desktop* X-Windows (XDM[3]) da plataforma do usuário para a máquina onde ele estivesse trabalhando. No exemplo acima, ao conectar-se a máquina B, o usuário iniciaria o X-Windows capturando o XDM da máquina

[3] XDM - X Display Manager

da plataforma A, transformando a máquina B num mero terminal-X. Esta solução, além de difícil utilização pois depende de características de cada servidor X-Windows, apresenta também as seguintes desvantagens: primeiro, todos os aplicativos seriam executados na máquina remota, o que poderia sobrecarregá-la se vários usuários estivessem fazendo o mesmo; segundo, a grande quantidade de informação gráfica poderia saturar a rede de comunicação; terceiro, o usuário não teria acesso direto as aplicações na própria estação em que estivesse trabalhando; por último, seria uma sub-utilização de uma estação de trabalho que apenas estaria executando o servidor X-Windows.

A opção adotada no **RemX** foi desenvolver uma ferramenta que permita que o usuário execute aplicações em qualquer máquina, onde a saída desta aplicação será na interface X-Windows da máquina que ele esteja trabalhando. Para cada aplicação iniciada, o **RemX** se conecta automaticamente com a máquina remota, exporta a saída para o X-Windows local e ativa a execução da aplicação. A grande vantagem do **RemX** é que tudo isso se torna inteiramente transparente para o usuário, podendo este executar aplicativos de qualquer máquina, inclusive a máquina local.

De forma a autenticar o pedido de execução remota, o **RemX** requisita o *login* e senha do usuário remoto, podendo inclusive executar aplicações de usuários diferentes na mesma máquina remota ou em máquinas distintas. De forma a tornar segura a transferência de informação de autenticação do usuário, o **RemX** utiliza um mecanismo de criptografia similar ao RSA [7] nos pacotes que contenham estas informações.

O **RemX** também permite ao usuário optar pela execução de uma aplicação em uma máquina específica ou numa dentre um conjunto de máquinas de uma mesma plataforma. Neste último caso o **RemX** escolhe a máquina com menor carga naquele momento para executar o aplicativo, distribuindo assim, a carga de trabalho entre as estações.

3 Arquitetura do RemX

Por definição, um ambiente **RemX** é composto por máquinas de plataformas diferentes (ou não), onde cada uma destas executa dois *daemons*, o *remxd* e o *cargad*. Estas máquinas também possuem disponível o *remx-cliente* que é iniciado pelo usuário sempre que este desejar. Dentro de um ambiente **RemX**, um conjunto de máquinas da mesma plataforma capazes de autenticar o mesmo grupo de usuários compõem um domínio **RemX**. Cada domínio é normalmente (mas não necessariamente) um domínio NIS. A figura 3.1 mostra um ambiente **RemX** com dois domínios diferentes.

O **RemX** usa uma arquitetura cliente-servidor [1] baseada no protocolo TCP/IP. A implementação das rotinas de comunicação do **RemX** foi feita usando a interface *socket*[1].

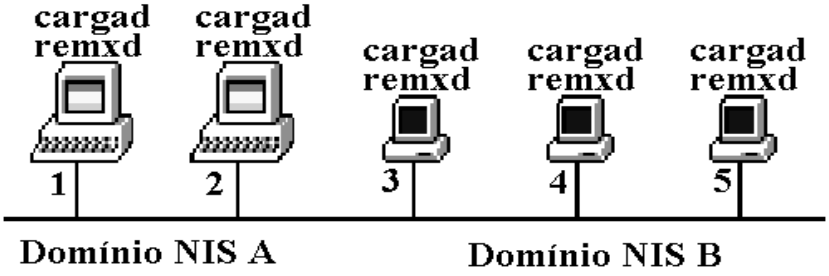

Figura 3.1. Domínio **RemX**, seus *deamons* e domínios

O objetivo do *cargad* é obter a carga da estação. Esta carga é obtida pela chamada do comando `uptime`. O *cargad* aceita um pedido via UDP, obtém a carga da estação e devolve ao chamador esta informação. Em linguagem procedural o *cargad* funciona da seguinte forma:

```
INICIO
   ENQUANTO VERDADE FAÇA
   (* Loop infinito        *)
      Espera conexão UDP;
      uptime;
      envia resultado do uptime à estação chamadora;
   FIM ENQUANTO
FIM PROCEDIMENTO
```

O objetivo do *remxd* é executar uma linha de comando passada pelo *remx-cliente*. Ele também autentica o usuário que está pedindo a execução do aplicativo e redireciona sua saída padrão e de erros (*stdout* e *stderr*) para a console do *remx-cliente*. Todas as informações necessárias a execução de uma aplicação (nome do usuário, senha, linha de comando, máquina que fez o pedido, porta TCP do console) são fornecidas pelo cliente e transmitidas criptografadas por questão de segurança. O *remxd* usa o protocolo TCP e pode aceitar conexões de vários clientes concorrentemente (controlado pelo *inetd*[4]). Em linguagem procedural o *remxd* funciona da seguinte forma:

```
INICIO
   (* Já existe conexão TCP aberta pelo remx-cliente         *)
   Gera chaves de criptografia;
   Envia chave pública para o cliente;
   Recebe do cliente solicitação de execução;
   Descriptografa solicitação;
   Efetua conexão TCP com console do remx-cliente;
   Redireciona saídas (stdout e stderr) para console;
   Autentica usuário na estação;
   SE usuário autenticado
      Então  Troca permissões de root para usuário autenticado;
             Executa comando com permissão do usuário;
      Senão  Manda para console informação de acesso negado;
   FIM SE
   Fecha conexão TCP aberta pelo remx-cliente;
   (* A conexão TCP com o console permanece aberta           *)
FIM PROCEDIMENTO
```

O *remx-cliente* é iniciado pelo usuário quando este deseja executar aplicações remotas. Após sua inicialização o *remx-cliente* exibe uma janela de comandos e uma janela de console. Quando o usuário pede uma execução de um aplicativo remoto, é requisitado o *userid* e senha do usuário remoto. Caso o usuário deseje executar um aplicativo em um domínio, o *remx-cliente* procura pela estação com menor carga dentro deste domínio e passa a execução para ela.

O *remx-cliente* possui um módulo servidor que aceita as conexões TCP das máquinas remotas e exibe na janela do console as informações que receber dentro desta conexão. É criado um *socket* para o servidor alocando a primeira porta TCP livre a partir da 20000.

[4] *inetd* - Super servidor do *UNIX* que permite a execução de um *daemon* servidor para cada conexão.

De forma a controlar várias conexões simultâneas com o servidor do console, é usada uma tabela com os *sockets* disponíveis para esta porta TCP. Para cada pedido de conexão com o servidor do console é alocado um novo *socket* e este é inserido na tabela. Para saber qual *socket* recebeu mensagem é usado a chamada de sistema `select`, que deixa ativo na tabela apenas os *sockets* que estão com informação recebida. O servidor então lê a informação destes *sockets* e exibe na tela.

Em linguagem procedural o *remx-cliente* funciona da seguinte forma:

```
INICIO
 Abre as Janelas de Menu de Aplicativos e Console;
 Aloca porta TCP para o módulo servidor do console;
 (* Aloca primeira porta TCP livre a partir de 20000   *)
 (* Qualquer mensagem recebida via esta conexão é      *)
 (* direcionada para o console                         *)
 Inicializa tabela de sockets;
 Inclui o socket do servidor na tabela;
 ENQUANTO usuário não pedir para terminar FAÇA
 (* Enquanto houver janela Tk                          *)
    Verifica se existe pedido nalgum dos sockets da tabela;
    SE houver pedido para socket do servidor
    (* Requisição de nova conexão                      *)
       ENTÃO  Cria um socket para esta conexão;
              Inclui socket criado na tabela;
    FIM SE

    SE pedido for para um socket que não do servidor
    (* Alguma mensagem foi recebida neste socket       *)
       ENTÃO  Lê mensagem;
              Exibe mensagem na tela do console;
    FIM SE

    (* Controle da Interface Gráfica                   *)
    SE existe evento Tk
       ENTÃO Executa evento Tk;
    FIM SE
FIM PROCEDIMENTO
```

O evento Tk que foi implementado para o *remx-cliente* tem a seguinte estrutura procedural:

```
INICIO
  Aguarda seleção do usuário;
  SE usuário escolheu uma aplicação para um domínio
     Então  Conecta todas os cargad de todas as estações do
                                                    domínio;
            (* cargad é conectado via porta UDP 628     *)
            Aguarda respostas dos cargad's ou
                      time-out de 3 segundos;
            Seleciona a estação com menor carga;
  FIM SE
  Conecta com remxd da estação selecionada;
  (* conexão feita via porta TCP 627                    *)
  Aguarda envio da chave de criptografia pública;
  Compõe linha de comando remoto e gera o pacote para
                              enviar para remxd;
  (* este pacote contém userid, senha, nome da         *)
  (* máquina cliente, e a própria linha de comando     *)
  Habilita X-Windows a receber saída da máquina remota;
  (* habilitação feita pelo comando xhost              *)
  Criptografa o pacote e o envia a remxd;
  Fecha conexão TCP com remxd;
FIM DE PROCEDIMENTO
```

Tanto a transmissão da chave pública quanto da solicitação de execução remota são feitas usando-se o protocolo XDR[5] [1]. Com isso garante-se total interoperabilidade do sistema independente das arquiteturas envolvidas.

4 Interface Gráfica

O **RemX** provê uma interface gráfica em X-Windows, desenvolvida em Tcl/Tk. As maiores vantagens da utilização desta linguagem/toolkit são a portabilidade, e a possibilidade de integração com "C" (para, por exemplo permitir a programação *socket*). Foi utilizada a versão do Tcl 7.4 e do Tk 4.0 com uma compilação e instalação muito simples nas máquinas DEC (OSF/1 3.0), SUN (SunOS 4.1.3) e LINUX disponíveis.

Figura 4.1 - Lançador de Programas

A interface consiste basicamente em um lançador (*launcher)* de programas remotos, com gerenciador de aplicativos, gerenciador de domínios **RemX**, e um console para onde serão redirecionadas as saídas padrão e de erro. A figura 4.1 mostra o lançador de programas.

O lançador de programas mantém a sua configuração salva em arquivos escondidos no diretório do usuário. Ao se cadastrarem novos usuários no ambiente **RemX**, os arquivos de configuração padrão (criados pelo administrador) são copiados para seu diretório. Cada usuário pode então

[5] XDR - External Data Representation [1], é um padrão de representação para troca de informação entre máquinas de arquiteturas diferentes.

modificar a lista de aplicativos e domínios, tanto quanto suas propriedades. O lançador também permite que se esconda/mostre a janela do console (figura 4.2).

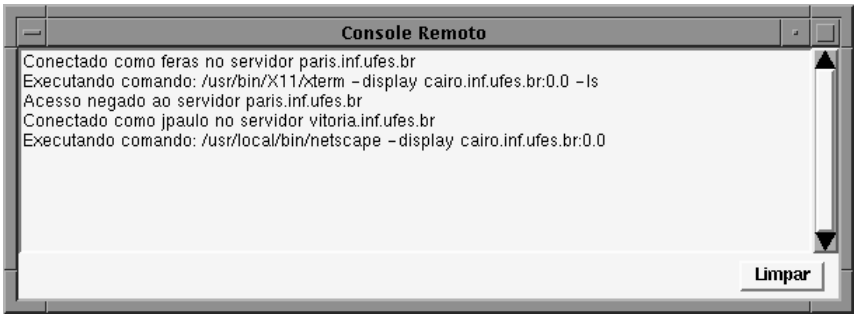

Figura 4.2 - Console do **RemX**

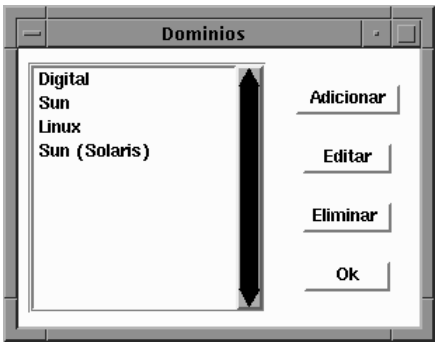

Figura 4.3 - Lista de Domínios

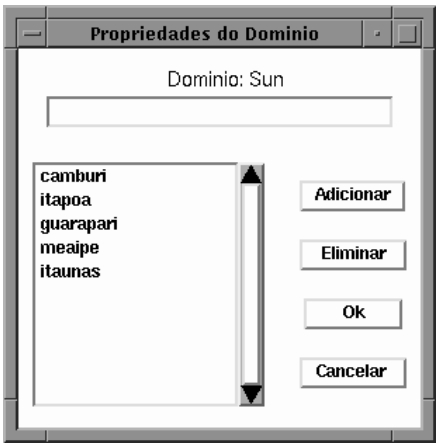

Figura 4.4 - Máquinas de um Domínio

Na caixa de diálogo **Propriedades da Aplicação**, configura-se a linha de comando, o usuário e a máquina na qual será executada. Se o usuário não deseja uma máquina especifica, ele pode optar por deixar o **RemX** selecionar a melhor máquina daquele domínio para a execução da aplicação. A vantagem de se colocar o nome do usuário nas propriedades do aplicativo é que, na hora da execução, será requerida ao usuário apenas a sua senha. Caso esse campo seja deixado em branco, o nome do usuário também será requisitado quando da execução.

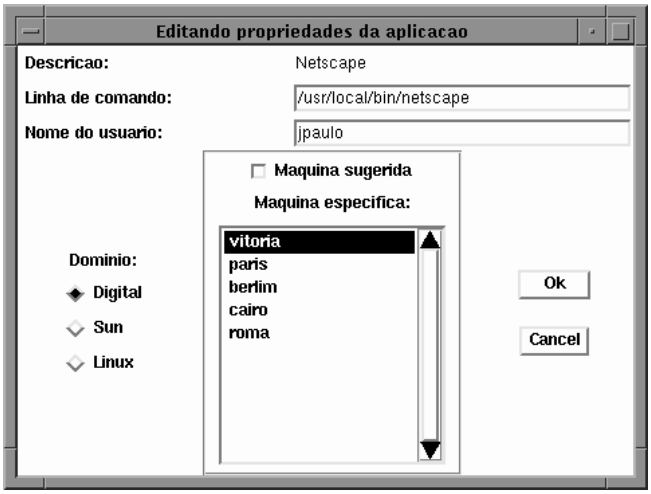

Figura 4.5 - Propriedades da Aplicação

A senha será requerida somente uma vez para aplicativos do mesmo domínio executando sob autenticação de um mesmo usuário, para cada sessão **RemX** (as senhas não são gravadas em disco). A figura 4.6 mostra um *desktop* OSF/1 (Motif) rodando aplicações locais e SunOS, através do **RemX**.

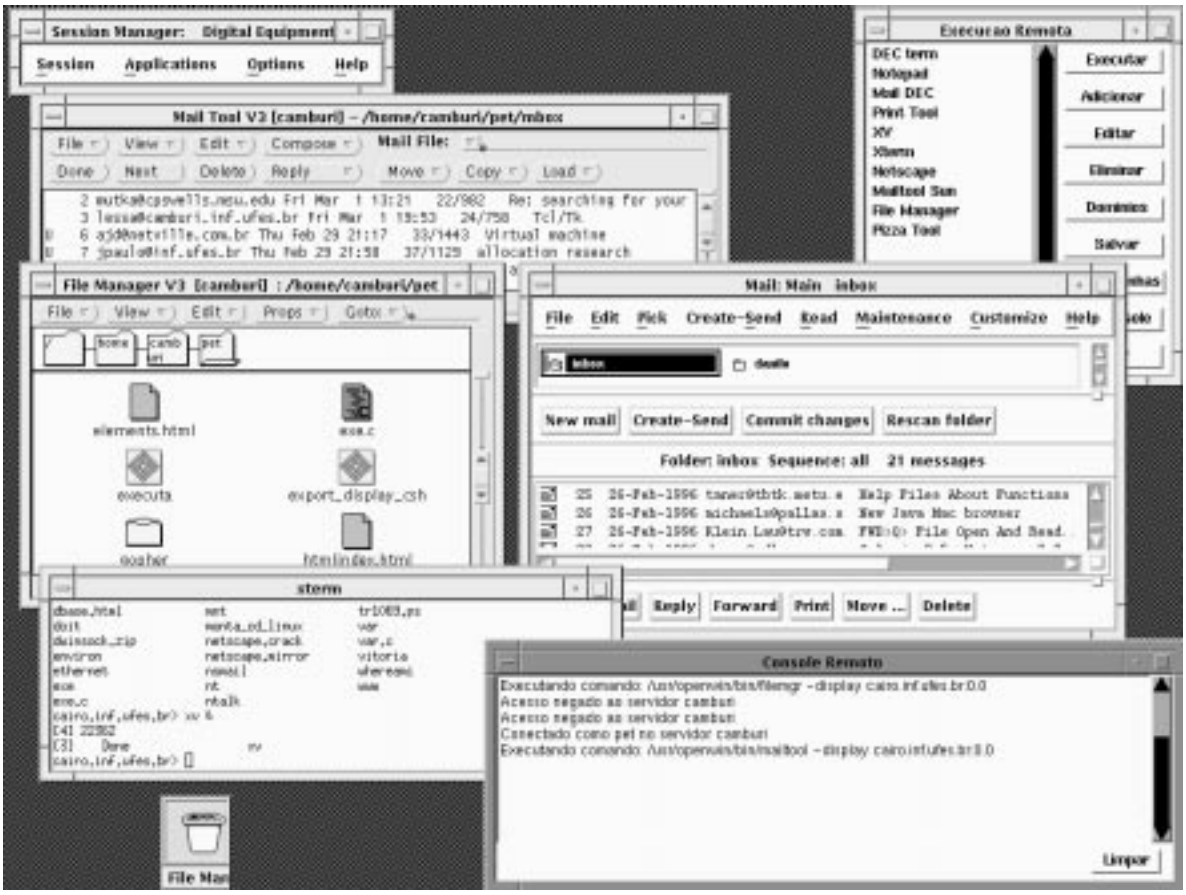

Figura 4.6 - *Desktop* OSF/1 (Motif) exibindo aplicações locais e remotas (SunOS).

5 Transações Seguras no RemX

O **RemX** poderia fazer uso do protocolo `rexec` para efetuar a execução remota, porém, esta solução apresentaria problemas básicos de segurança. A transmissão das informações de autenticação do usuário (inclusive a senha) no `rexec` é feita sem segurança. Isto faz com que a simples interceptação dos pacotes de comunicação permita a invasão das contas dos usuários de aplicações remotas.

O **RemX** contorna este problema utilizando a criptografia nos pacotes de autenticação. Foi escolhido um algoritmo [7] similar ao RSA, definido por Rivest, Shamir e Adleman [6], que utiliza uma fórmula matemática para codificação dos dados. O algoritmo opera com um esquema de duas chaves: uma pública e uma privada. Os dados criptografados com a pública só podem ser restaurados utilizando a chave privada.

Toda vez que o servidor *remxd* recebe uma conexão, ele gera um novo conjunto de chaves e envia a pública para o cliente. A partir deste ponto, o cliente só enviará ao servidor informações criptografadas.

Quanto à segurança do RSA, há uma dependência direta do número de bits utilizados nas chaves, que pode ir até 1024. Vale lembrar que, por motivos de velocidade no processo de criptografia/geração de chaves, foi usado como *default* um nível de segurança que utiliza chaves de 128 bits. Para ter-se um idéia de como é difícil quebrar chaves deste tamanho, foram necessários 8 dias, 120 workstations e 2 supercomputadores para violar o sistema de segurança do Netscape [8], que utiliza apenas 40 bits. Para cada bit extra da chave, este tempo dobra, ou seja, uma estimativa para violação do sistema de segurança do **RemX**, seria de $6{,}78 \times 10^{24}$ anos utilizando o mesmo poder de processamento, ou seja, virtualmente inquebrável.

6 Balanceamento de Carga

Quando um domínio do ambiente **RemX** é também um domínio NIS, torna-se indiferente ao usuário qual das máquina executará a aplicação (partindo do pressuposto que todas as máquinas deste domínio tenham a mesma configuração). O usuário pode ser autenticado em qualquer máquina e seu diretório *home* estará disponível.

De forma a balancear a carga de trabalho das máquinas em um domínio, o **RemX** dá a opção de executar a aplicação na máquina que estiver com menor carga. Para tal, o **RemX** consulta todas as máquina do domínio e passa a execução da tarefa àquela com menor carga.

O processo de avaliação da carga empregado no **RemX** é bastante simples. Ele apenas verifica a média de quantos processos estiveram na fila de execução. Para obter este valor usou-se o comando UNIX `uptime`. Este comando fornece o seguinte resultado:

```
19:42 up 4 days, 6:29, 8 users, load average: 5.20, 2.03, 0.80
```

No caso do **RemX** foi usado o valor central do `load average`, que corresponde ao número médio de processos nos últimos 30 segundos.

Vale salientar que este parâmetro de carga só pode ser considerado caso as máquinas do domínio possuam configurações muito semelhantes. No caso de máquinas heterogêneas outros métodos de medida de carga deverão ser usados.

Por exemplo, se um usuário recebe o seu correio eletrônico no domínio NIS DEC OSF/1 e está usando uma estação SUN no mesmo ambiente **RemX**, ele precisa executar o aplicativo `dxmail` para ler o seu correio eletrônico. Como o domínio DEC OSF/1 é também um domínio NIS, para o usuário é indiferente em qual máquina o `dxmail` será executado. Caso o usuário queira, o **RemX** executará o aplicativo na máquina de menor carga do domínio.

7 Conclusão

O **RemX** mostrou ser uma ferramenta bastante útil num ambiente muti-plataforma. A experiência do Laboratório de Informática da UFES foi muito bem sucedida com uma boa aceitação por parte dos usuários. Neste laboratório estão presentes as plataformas Sun SunOS 4.1.3, DEC OSF/1 3.0 e LINUX 1.2.13.

Com uma configuração *default* bem feita pelo administrador do ambiente, os usuários praticamente não precisam alterar as definições do **RemX**. A interface com o usuário também é bastante amigável de forma que muito pouco tempo é gasto no treinamento.

Como a interface é a mesma independente da plataforma, os usuários passaram a usar mais as máquinas das plataformas que não as suas originais. Este comportamento trouxe uma melhor utilização dos postos de trabalho disponíveis no laboratório, bem como a criação de um processo de migração mais suave.

Contudo, para que o **RemX** funcione adequadamente é preciso que todos os usuários sejam cadastrados em todas as máquinas do ambiente, ou seja, em todos os domínios NIS. O próximo passo no projeto de integração de sistemas desenvolvido na UFES será uma ferramenta para o cadastramento automático de usuários em todos os domínios NIS de um ambiente **RemX**.

Por se tratar de uma ferramenta multi-plataforma, o **RemX** foi implementado de forma a ser facilmente portável. Foram feitos testes em três plataformas diferentes sem maiores dificuldades e poucas modificações. Isso nos leva a crer que a inclusão de novos domínios de diferentes plataformas no ambiente **RemX** se dará sem dificuldades.

Do ponto de vista acadêmico este projeto foi também bastante proveitoso. A experiência adquirida em programação *socket*, programação X-Windows Tcl/Tk e programação em ambiente UNIX abre possibilidades para vários outros projetos na área.

Maiores informações sobre este projeto, incluindo códigos fonte dos programas implementados podem ser encontrados na URL `http://www.inf.ufes.br/~pet/projetos`

8 Bibliografia

[1] Comer, Douglas E. & Stevens, David L, *Internetworking with TCP/IP - Client-Server Programming and Applications BSD Socket Version - Volume III*. Prentice-Hall International.

[2] Mui, Linda & Pearce, Eric, *X Windows System Administrator's Guide - Volume Eight*. O'Reilly & Associates, Inc.

[3] Ousterhout, John K., *Tcl and the Tk Toolkit*. Addison-Wesley.

[4] Bond, Andy, "Load Sharing in a Distributed Environment"*Proceedings of UniForum NZ '92.*

[5] Manual do SunOS 4.1.3 - *Network Programming Guide* - Sun Microsystems.

[6] R.L. Rivest, A. Shamir, and L. Adleman. "A method for obtaining digital signatures and public-key cryptosystems". *Communications of the ACM*, 21(2):120--126, February 1978.

[7] Paasivirta, Risto, "Minimalistic implementation of RSA public key algorithm", *URL: http://www.jyu.fi/~paasivir/crypt*

[8] Tabibian, O. Ryan. "Net security under scrutiny". *PC Magazine*, Oct. 24, 1995 v.14 n.18 p.29.

[9] Tanenbaum, Andrew S. - *Modern Operating Systems.* Prentice-Hall International.