# Aplicações de Gerência Extensíveis

Ana Lúcia de Moura
Edison Ishikawa
Michele Espíndula Lima
Noemi Rodriguez

Departamento de Informática — PUC/Rio
Rua M. S. Vicente 225, Gávea
22453-900 — Rio de Janeiro — RJ
*ana,edison,michele,noemi@inf.puc-rio.br*

PUC-Rio Inf.MCC12/98 March, 1998

*Abstract:* This paper proposes an environment for the development of network management applications. This environment is based on Lua, which is an interpreted language, and on a set of libraries. The paper gives special attention to the LuaMan library, which was developed for the specific needs of network management. Applications written on this platform can be dynamically extended; this allows new functionality to be added without the need of recompilation. The paper also describes a *web based* management application which was built using the proposed environment.

*Keywords:* network management, interpreted language, extensibility

*Resumo:* Este trabalho propõe um ambiente de desenvolvimento de aplicações de gerência de redes composto por Lua, uma linguagem interpretada, e por um conjunto de bibliotecas. Em particular, a biblioteca LuaMan, construída especificamente para oferecer funcionalidades relativas a gerência, é descrita com detalhes. Aplicações construídas neste ambiente podem ser dinamicamente estendidas, agregando novas funcionalidades sem necessidade de recompilação. O trabalho apresenta também uma aplicação de gerência baseada em *web* construída com uso da plataforma proposta.

*Palavras chave:* gerência de redes, linguagem interpretada, extensibilidade

# 1 Introdução

O crescimento do tamanho e complexidade das redes de computadores, assim como do uso de serviços e aplicações em redes, criou um espectro de necessidades de gerência muito mais amplo do que o previsto há alguns anos atrás. A solução tradicional de aplicações de gerência específicas é claramente inadequada, uma vez que um mesmo administrador hoje tipicamente gerencia dezenas de serviços e aplicações. A extensibilidade de plataformas de gerência passou a ser um imperativo pela necessidade de se construir ambientes integrados de gerência, aos quais o administrador possa agregar a funcionalidade de controle de novos dispositivos e serviços a medida que estes surgirem.

Além disto, cada instalação e cada grupo de usuários impõe requisitos diferentes em relação à funcionalidade oferecida pela aplicação de gerência. Em alguns ambientes, o administrador pode estar interessado em garantir a disponibilidade dos serviços, enquanto em outros o interesse maior pode ser no tempo de resposta, ou ainda na geração de informação necessária para contabilizar gastos de recursos. Uma única aplicação de gerência não pode embutir as janelas e estatísticas relevantes para todos os tipos de uso.

Uma opção que é oferecida por várias plataformas de gerência é a provisão de uma biblioteca para desenvolvimento de aplicações de gerência. Tipicamente, uma linguagem como C ou C++ é utilizada para construir uma aplicação "sob medida" para um determinado contexto. Esta solução exige considerável experiência de programação por parte do desenvolvedor, e implica em um investimento que em geral só pode ser feito por instituições de certo porte. Uma aplicação como esta muitas vezes deve ser feita para satisfazer as necessidades dos vários subgrupos da instituição interessada. Ajustes finos da aplicação, como fontes e cores de exibição da informação, podem muitas vezes ser feitas através de menus. No entanto, a adição de novas funcionalidades ou até mesmo a retirada de exibição de informação irrelevante muitas vezes só podem ser feitas com a recompilação da aplicação.

Ao invés de menus, pode-se lançar mão de uma *linguagem de extensão*, aumentando em muito o potencial de reconfiguração da aplicação. Linguagens de extensão são, tipicamente, linguagens de programação interpretadas que permitem o desenvolvimento de *scripts* de configuração da aplicação. Com uma linguagem de programação como ferramenta de configuração, pode-se utilizar condicionais e repetições para decidir, por exemplo, que conjunto de janelas deve ser exibido em determinada situação.

O uso de linguagens de extensão está se tornando bastante difundido. Muitas aplicações atuais utilizam uma estrutura que consiste de um núcleo, escrito numa linguagem compilada, contendo a porção "estável" do programa, e uma parte configurável, escrita em uma linguagem interpretada. Frequentemente, essa segunda parte descreve a interface da aplicação com o usuário, enquanto o núcleo contém a maior parte do processamento.

No contexto de gerência, gostaríamos de dar ao usuário a chance de controlar não apenas a interface, isto é, *como* a informação é exibida/controlada, mas também a parte mais interna da aplicação, que neste caso diz respeito a *quais* informações são monitoradas e como essas informações são manipuladas. Por isso, optamos por uma plataforma para desenvolvimento de aplicações de gerência onde a aplicação é totalmente desenvolvida em uma linguagem interpretada. A linguagem escolhida foi Lua [IFC96, IFC97], uma linguagem de configuração desenvolvida na PUC-Rio.

Para que uma linguagem de programação possa ser utilizada tanto na construção de uma aplicação de gerência como em extensões que agreguem nova funcionalidade a uma aplicação já existente, é necessário que ela ofereça acesso ao SNMP e a outros serviços básicos na área de gerência, como ICMP e DNS. A biblioteca LuaMan, descrita na seção 3.1, foi desenvolvida para este fim. Uma série de outras bibliotecas já existentes para Lua, descritas na seção 3.2, fornecem facilidades para construção de interfaces gráficas, construção de páginas WWW dinâmicas, acesso a bases de dados e acesso a objetos CORBA.

O ambiente de desenvolvimento e extensão de aplicações de gerência resultante pode ser usada por usuários/programadores com diferentes níveis de sofisticação. Por ter uma sintaxe bastante simples, Lua pode ser utilizada por usuários pouco experientes para tarefas fáceis, como mudança de fontes e tamanhos de janelas, e ao mesmo tempo, pode ser uma ferramenta de programação bastante poderosa para administradores mais experientes. Apresentamos neste trabalho uma aplicação específica que utiliza essa biblioteca – uma ferramenta de gerência *baseada em web*, isto é, onde a tarefa de gerência pode ser realizada a partir de qualquer browser WWW. Como discutido na seção 4, essa abordagem apresenta uma série de vantagens, entre elas a ampla disponibilidade de browsers, que permite que a tarefa de gerência possa ser realizada a partir de qualquer plataforma.

## 2 A Linguagem de Extensão Lua

Lua é uma linguagem de extensão de propósito geral que integra facilidades para descrição de dados, reflexividade e construções com sintaxe simples. Entre os aspectos 'convencionais' de Lua, estão seu caráter procedural, com estruturas de controle padrão (*while, if, ...*), definições de funções com parâmetros e variáveis locais, etc. Entre suas características menos convencionais, Lua oferece arrays associativos (*tabelas*) como um mecanismo geral de estruturação de dados, e funções como valores de primeira classe.

Tipos são verificados dinamicamente em Lua. Uma variável pode armazenar um valor de qualquer tipo. Sempre que uma operação é ativada, os tipos de seus argumentos são verificados. Além dos tipos básicos `number` (floats) e `string`, e do tipo `function`, Lua oferece mais três tipos de dados: `nil`, com um único valor também chamado `nil`, cuja propriedade básica é ser diferente de qualquer outro valor da linguagem; `userdata`, que representa ponteiros em C; e `table`.

O tipo `table` implementa arrays associativos, ou seja, arrays que podem ser indexados por qualquer valor da linguagem (com exceção de `nil`). Estruturas podem ser implementadas por tabelas, usando-se os nomes de campos como índices. Lua dá suporte a essa representação, tratando `a.nome` como açúcar sintático para `a["nome"]`. Arrays associativos constituem uma ferramenta de programação bastante poderosa; muitos algoritmos se tornam triviais com seu uso [Ben88]. Estruturas de dados mais comuns, como arrays convencionais, conjuntos, bags e tabelas de símbolos, podem ser facilmente implementadas com tabelas.

A figura 1 mostra um exemplo de uso de tabelas para representar um conjunto. A função `next`, pré-definida em Lua, permite que se percorra uma tabela. Essa função recebe dois parâmetros: a tabela e um índice. Quando este índice é diferente de `nil`, a função retorna um próximo índice e o valor correspondente; quando o índice passado é igual a `nil`, a função retorna um "primeiro" índice e o valor correspondente (ou seja, o percurso da tabela é iniciado). Essa função ilustra também uma outra característica de Lua: a possibilidade de uma função retornar mais de um valor (o que permite que toda passagem de parâmetros seja por valor).

Funções em Lua são valores de primeira classe, ou seja, podem ser manipuladas como qualquer outro valor da linguagem. Uma definição de função corresponde à criação de um valor do tipo `function` e à atribuição desse valor a uma variável global. Como qualquer outro valor, esse valor pode ser armazenado em variáveis, passado como parâmetro, ou retornado como resultado de uma outra função. Em particular, campos de tabelas podem armazenar funções. Essa propriedade permite a implementação de alguns mecanismos interessantes de orientação a objetos, cujo uso é simplificado com um pouco de açúcar sintático. Por exemplo, a sintaxe:

```
receiver:foo(params)
```

é equivalente a

```
conj = {}
conj["banana"] = 1
conj[3] = 1
...
-- teste de pertinencia
if conj["tangerina"] then write("tangerina esta' no conjunto") end
...
-- impressao de todos os elementos do conjunto
local indice, elem = next(conj,nil)
while i do
  write(conj[indice])
  indice, elem = next(conj,indice) -- avanca no percurso da tabela
end
```

Figura 1: Exemplo de uso de tabela

```
receiver.foo(receiver, params)
```

Ou seja, a função armazenada no campo `foo` do objeto `receiver` é chamada, tendo como primeiro parâmetro o próprio `receiver` (que faz assim o papel do `this` em C++).

Parte da natureza reflexiva de Lua é fornecida por *fallbacks*. Fallbacks permitem que o programador modifique o comportamento de certas construções da linguagem, em especial para situações de erro. Fallbacks em Lua têm implementação eficiente e muitas vezes são usadas para emular facilidades inexistentes na linguagem original, como herança e sobrecarga [IFC96]. Como um exemplo, a expressão apresentada acima,

```
receiver:foo(params)
```

geraria um erro de execução caso `foo` não fosse um campo da tabela `receiver`. O programa pode interceptar esse erro e procurar uma descrição de `foo` em outro local (por exemplo, em outra tabela), simulando um tipo de herança. Em [IFC96] o leitor pode encontrar um tratamento mais completo das facilidades de orientação a objetos, que está fora do escopo deste trabalho.

Lua é implementada por um interpretador escrito em ANSI C, e executa atualmente em plataformas que variam de DOS a CRAY.

# 3 Plataforma de Desenvolvimento

O ambiente de desenvolvimento que se propõe neste trabalho é resultante da integração da linguagem Lua, descrita na seção anterior, com uma série de bibliotecas que oferecem serviços importantes para o contexto de gerência de redes.

A seção 3.1 descreve a biblioteca Luaman, desenvolvida especificamente para suprir necessidades ligadas a gerência de redes. Na seção 3.2, várias outras bibliotecas relevantes, já disponíveis anteriormente para Lua, são brevemente apresentadas.

## 3.1 LuaMan

A construção de aplicações de gerência de redes requer o acesso a diversos serviços relevantes a esse contexto. Uma necessidade facilmente identificável é o acesso ao protocolo SNMP, para permitir que as aplicações de gerência interajam com os agentes SNMP disponíveis na rede gerenciada, e, assim, manipular as informações de gerenciamento mantidas por esses agentes em suas MIBs [Sta96]. Contudo, também o acesso ao protocolo

ICMP e a informações mantidas pelo Domain Name System (DNS) e armazenadas em bases de dados de redes locais constituem serviços básicos de grande utilidade no desenvolvimento de aplicações e ferramentas de gerência.

De forma a possibilitar o uso da linguagem Lua na construção de aplicações de gerência, uma biblioteca foi desenvolvida, estendendo a linguagem através da implementação de uma API de gerenciamento Lua. Na definição desta API buscou-se oferecer uma abstração de nível mais alto que o normalmente oferecido pelas plataformas de gerência, encapsulando-se, sempre que possível, a complexidade dos serviços e protocolos acessados. Além disso, o acesso aos diversos serviços e protocolos relevantes a aplicações de gerência de redes – e não apenas ao protocolo SNMP – é integrado numa única biblioteca. Nesse ponto, este trabalho é bastante semelhante ao descrito em [SL95], que foi usado como base para várias idéias da biblioteca.

A implementação atual de LuaMan, escrita em C, é baseada na biblioteca de gerência SNMP desenvolvida em Carnegie Mellon, CMU SNMP/SNMPv2. A biblioteca LuaMan pode ser portada para diversas plataformas, e está instalada, atualmente, nos ambientes Linux, SunOS, Solaris, IRIX e AIX.

Apresentamos a seguir os vários conjuntos de primitivas disponibilizados pela biblioteca LuaMan. A documentação completa da API de gerenciamento está disponível em [ML97]. Para uma discussão mais detalhada sobre a biblioteca, ver [Lim98].

### Primitivas SNMP

Este conjunto de primitivas oferece o acesso às operações básicas do protocolo SNMP, permitindo sua invocação nos modos síncrono e assíncrono. Atualmente apenas a versão do protocolo SNMPv1 é suportada. A próxima versão da biblioteca LuaMan, em desenvolvimento, incluirá o suporte às versões SNMPv2C e SNMPv2U (USEC). A figura 2 apresenta a sintaxe das primitivas do conjunto SNMP.

```
session, status = snmp_open{<lista de parâmetros de configuração>}

vbList, status, index = snmp_get(session, vars)
vbList, status, index = snmp_getnext(session, vars)
vbList, status, index = snmp_set(session, vars)

reqID, status, index = snmp_asynch_get(session, vars, [callback])
reqID, status, index = snmp_asynch_getnext(session, vars, [callback])
reqID, status, index = snmp_asynch_set(session, vars, [callback])

status = snmp_close(session)
```

Figura 2: Primitivas SNMP

Através da primitiva `snmp_open`, um construtor de tabelas, o usuário da API de gerenciamento cria uma *sessão SNMP*, definindo os parâmetros de configuração necessários à invocação das operações do protocolo sobre o agente identificado. A sintaxe dessa primitiva é simplificada com o uso do açúcar sintático `snmp_open{...}`, que permite a construção de uma nova tabela, passada como argumento na invocação da função. As opções de configuração de uma sessão SNMP incluem o endereço IP do agente associado, a versão do protocolo a ser utilizada, e as informações necessárias para a autenticação junto ao agente. Os parâmetros opcionais `callback` e `trap` permitem a definição de rotinas *default* para o tratamento das respostas a operações assíncronas e para o tratamento de notificações (*traps*) originadas pelo agente associado à sessão. Uma sessão SNMP é finalizada através da primitiva `snmp_close`.

As operações básicas do protocolo são invocadas, em modo síncrono, através das primitivas `snmp_get`, `snmp_getnext` e `snmp_set` e, em modo assíncrono, através das primitivas `snmp_asynch_get`, `snmp_asynch_getnext` e `snmp_asynch_set`. Novamente, a sintaxe das primitivas é simplificada com o uso das características da linguagem. Neste caso, utilizou-se a possibilidade de uma função retornar múltiplos valores.

A invocação síncrona das operações fornece como retorno a lista de variáveis solicitadas, o *status* correspondente à execução da operação e, quando aplicável, o índice da variável causadora da falha na execução da operação. A identificação das variáveis requisitadas pode ser fornecida através de seu nome (*MIB label*) ou `OID` (o identificador do objeto em *dotted notation*), ou ainda através de tabelas do tipo *varbind*. Uma tabela *varbind* descreve uma variável através de três atributos : `oid`, `type` e `value`. Cada elemento da lista de variáveis retornada é uma tabela do tipo *varbind*. Os tipos de dados SNMP são mapeados para os tipos básicos de Lua (`string`, `number`, `table` e `nil`).

A invocação assíncrona das operações fornece como retorno imediato uma identificação para a requisição, e, em caso de erro identificado localmente, o *status* correspondente e o índice da variável causadora do erro. O parâmetro opcional `callback` permite a definição de uma rotina de tratamento específica para a resposta à operação solicitada. A rotina de *callback* receberá como parâmetros a lista de variáveis solicitadas, o status relativo à operação, o índice da variável causadora de um possível erro e as identificações da requisição e sessão SNMP correspondentes

No exemplo apresentado na figura 3 implementa-se uma função que percorre uma subárvore MIB de um agente SNMP.

```
function walk(host,commStr,subtree)
  s=snmp_open{peer=host,community=commStr}
  root = mib_oid(subtree)
  if root = nil then
    print("snmpwalk: subarvore invalida ("..subtree..")")
    return
  end
  vBind = {oid=root}
  repeat
    vBind = snmp_getnext(s,vBind)
    if vBind then
       if not strfind(vBind.oid,root) then
          mibEnd = 1
       else
          print(sprint_var(vBind))
       end
    end
  until (not vBind) or mibEnd
  snmp_close(s)
end
```

Figura 3: MIB walk

### Primitivas MIB

O conjunto de primitivas MIB permite a carga e consulta a definições de MIBs SNMP, sendo especialmente interessante para o desenvolvimento de aplicações como *MIB Browsers*. A figura 4 apresenta a sintaxe dessas primitivas.

```
status, errString = mib_load(filename)

name, status = mib_name(var)
name, status = mib_fullname(var)
oid, status  = mib_oid(var)

descr, status  = mib_description(var)
access, status = mib_access(var)
type, status   = mib_type(var)
enums, status  = mib_enums(var)

successors, status = mib_successor(var)
parent, status     = mib_parent(var)
```

Figura 4: Primitivas MIB

A primitiva `mib_load` realiza a carga das definições de MIBs contidas no arquivo passado como parâmetro. As sub-árvores correspondentes a essas definições são adicionadas à árvore MIB global. Caso algum erro seja detectado durante a análise das definições, um `string` contendo a descrição do erro e o número da linha no arquivo onde o erro foi encontrado será retornado.

As primitivas `mib_name` e `mib_fullname` permitem a obtenção, respectivamente, do nome e do nome completo (*full name*) de uma variável da MIB, dada sua identificação (OID) em *dotted notation*. A primitiva `mib_oid` realiza a conversão do nome de uma variável em sua identificação.

As primitivas `mib_description`, `mib_access` e `mib_type` retornam a descrição textual, o tipo de acesso, e o tipo de dado associado a uma variável, respectivamente. Através da primitiva `mib_enums` é possível obter a lista de `strings` associados, através de uma *enumeração*, aos valores que uma variável poderá assumir.

A primitiva `mib_successor` retorna a lista de sucessores imediatos de uma variável na MIB global. Finalmente, `mib_parent` fornece o antecessor da variável identificada por seu nome ou OID.

### Primitivas ICMP

O protocolo ICMP [Pos81] provê mecanismos para notificação de condições de erro e transferência de informações entre máquinas. Como todo dispositivo que implementa o protocolo IP suporta também as operações básicas do protocolo ICMP, o acesso a essas operações permite a realização de procedimentos de monitoração de dispositivos que não suportam o protocolo SNMP. Além disso, o acesso ao protocolo ICMP permite a implementação dos algoritmos normalmente utilizados em ferramentas que realizam a descoberta da estrutura de redes IP [SL93]. A figura 5 apresenta a sintaxe do conjunto de primitivas ICMP.

A primitiva `icmp_echo` permite que se verifique a acessibilidade de um determinado dispositivo através do envio de uma requisição de *echo*. Essa facilidade é muito utilizada em ferramentas de monitoração básica, como, por exemplo, o utilitário `ping`.

```
tripTime, status = icmp_echo(host)
mask, status = icmp_mask(host)
timeDiff, tripTime, status = icmp_time(host)
host, tripTime, status = icmp_trace(host,ttl)

hosts, netIP , status = icmp_netecho(addr,mask)
```

Figura 5: Primitivas ICMP

As primitivas `icmp_mask` e `icmp_time` fornecem, respectivamente, a máscara de rede utilizada por uma máquina e a diferença entre os relógios da máquina especificada e da máquina executando a aplicação de gerência.

A primitiva `icmp_trace` permite a construção de ferramentas para levantamento e análise de rotas, como, por exemplo, o utilitário `traceroute`. A invocação dessa primitiva provoca o envio de um pacote UDP com o TTL especificado a uma porta não utilizada da máquina alvo. O retorno dessa primitiva fornece o endereço IP da máquina que está `ttl` pontos à frente na rota para a máquina de destino e o tempo de percurso do pacote a esse ponto da rota.

A definição da primitiva `icmp_netecho` teve como objetivo facilitar a implementação de algoritmos de descoberta de estruturas de redes IP. Dados o endereço IP de uma rede e sua máscara, a invocação da primitiva `icmp_netecho` provoca o envio de requisições de *echo* a todos os possíveis endereços IP da rede especificada. Como retorno, essa primitiva fornece uma tabela contendo todos os endereços IP correspondentes a máquinas "ativas" nessa rede. O segundo retorno dessa primitiva retorna o endereço IP da rede analizada. Esse valor é útil quando o parâmetro `addr` é, na verdade, o endereço IP de uma determinada máquina e se deseja conhecer a estrutura da rede à qual essa máquina está conectada.

A figura 6 mostra um exemplo de uso da primitiva `icmp_netecho` para descoberta de uma rede IP (no caso, a rede `"139.82.95"`) com máscara `"255.255.255.192"`. Para cada endereço IP correspondente a uma máquina descoberta, o serviço DNS é consultado, através da primitiva `dns_ptr`, para obter o nome da máquina correspondente.

### Primitivas DNS

O conjunto de primitivas DNS foi definido de forma a não só oferecer a possibilidade de consulta às informações mantidas pelo Domain Name System (DNS) [Moc87] como também permitir a depuração desse serviço. A sintaxe das primitivas que compõem esse conjunto é apresentada na figura 7.

As primitivas `dns_a`, `dns_ptr` e `dns_cname` realizam as consultas consideradas básicas para o serviço DNS, ou seja, a conversão de nomes em endereços IP - e vice-versa - e a obtenção do nome oficial (*canonical name*) correspondente a um *alias*. Através da primitiva `dns_hinfo` são obtidas informações sobre o *hardware* e sistema operacional de uma máquina. As primitivas `dns_soa` e `dns_ns` permitem a obtenção de informações sobre a zona de autoridade e os servidores com autoridade para responder a consultas sobre um domínio. Finalmente, `dns_mx` fornece informações sobre os servidores de correio (*mail exchangers*) de um dado domínio.

Além de seu valor de retorno básico, as primitivas DNS oferecem retornos adicionais que informam quais os servidores de nomes responsáveis pela realização da consulta especificada e quais os servidores com autoridade para respondê-la. Esses retornos adicionais visam auxiliar o usuário experiente na depuração dos serviços DNS. É interessante notar que a sintaxe de Lua permite que retornos de função desnecessários sejam ignorados,

```
-- Descobre maquinas ativas na rede 139.82.95.0
hosts, netIP = icmp_netecho("139.82.95.0","255.255.255.192")
if hosts then
  print("Maquinas na rede "..netIP.." : \n")
  -- Percorre a tabela de hosts traduzindo enderecos para nomes
  local i,ip = next(hosts,nil)
  while i do
    hostPTR = dns_ptr(ip)
    if hostPTR then
      print(format(" %s -> %s\n",ip,hostPTR[1]))
    else
      print(format(" %s\n",ip))
    end
    i, ip = next(hosts,i)
  end
else
  print("Nao foram encontradas maquinas ativas na rede "..netIP\n")
end
```

Figura 6: Descoberta de rede IP com uso de primitivas ICMP e DNS

```
addrs, servT, authT, status = dns_a(host)
names, servT, authT, status = dns_ptr(address)
cname, servT, authT, status = dns_cname(alias)
hinfo, servT, authT, status = dns_hinfo(host)

authSrv, servT, authT, status = dns_ns(domain)
soaRR,   servT, authT, status = dns_soa(domain)
mailX,   servT, authT, status = dns_mx(domain)
```

Figura 7: Primitivas DNS

simplificando a chamada das primitivas quando apenas o resultado básico da consulta é requerido.

#### Primitivas DB

O objetivo do conjunto de primitivas DB é permitir o acesso a informações armazenadas nos arquivos de configuração que compõem a base de dados local. Através desse conjunto de primitivas é oferecido o acesso às informações contidas nos arquivos *hosts*, *services*, *protocols* e *networks* de sistemas Unix. A sintaxe desse conjunto de primitivas é apresentada na figura 8.

### 3.2 Outras Bibliotecas

Além do acesso a agentes SNMP, o acesso a objetos CORBA [Obj95] vem se afirmando como uma importante alternativa na área de gerência de serviços e telecomunicações [KC96, Ban97]. A biblioteca LuaOrb [CRI97, ICR98] fornece uma amarração entre Lua e CORBA, baseada no uso da interface de invocação dinâmica. O uso de Lua para acesso dinâmico a servidores CORBA permite que trabalhemos com um console de gerência a

```
hosts, status = db_host([host])
services, status = db_service([service | port])
protocols, status = db_protocol([protName | protNum])
nets, status = db_net([net])
```

Figura 8: Primitivas DB

partir do qual o administrador pode interrogar o repositório e fazer acessos aos objetos encontrados. O uso desta facilidade é explorado em [RCIM97], onde é apresentada uma aplicação que realiza acessos tanto a objetos CORBA como a agentes SNMP.

O console de gerência é extremamente útil para realização de testes e pequenas verificações. No entanto, uma necessidade óbvia no desenvolvimento de aplicações de gerência mais permanentes é a construção de interfaces gráficas. A biblioteca IUP [LFG+96], implementada para uma série de plataformas, permite que um programa Lua controle a criação de janelas, menus, etc.

Uma outra facilidade importante para atividades de gerência é o acesso a bases de dados. A biblioteca DBLua [dbl] permite que programas Lua façam acessos a qualquer base de dados com interface ODBC.

Uma idéia que vem ganhando importância é a da "gerência baseada em *web*", onde um browser WWW é utilizado como console de gerência. A biblioteca CGILua [HBI97] dá suporte à confecção de páginas HTML dinâmicas e manipulação de dados provenientes de formulários. Em CGILua, páginas dinâmicas podem ser construídas ou através de *scripts Lua puros*, cuja saída deve ser uma página HTML, ou através de *arquivos HTML mesclados*, onde instruções Lua e diretivas CGILua são inseridas em documentos HTML convencionais. A combinação dessas duas possibilidades resulta em um ambiente de desenvolvimento bastante poderoso, onde scripts Lua especializados podem ser acionados por arquivos mesclados que contêm toda a definição de layout gráfico. A seção seguinte discute uma aplicação de gerência que utiliza essa biblioteca.

## 4 Gerência baseada em *Web*

O uso de um browser WWW para a tarefa de gerência pode trazer vários benefícios [Ste97, Ros96]. A existência de browsers para praticamente todas as plataformas de uso corrente tira dos fabricantes o ônus de desenvolvimento multiplataforma. O problema de segurança passa a ser o problema de segurança no modelo WWW, que vem sendo amplamente estudado e para o qual algumas soluções já foram implementadas.

Para que uma aplicação de gerência baseada em WWW possa fazer mais do que simplesmente visualizar relatórios gerados por ferramentas independentes, é necessário fazer uso de mecanismos de geração dinâmica de páginas. Utilizando para isto apenas os mecanismos já padronizados, pode-se trabalhar com a arquitetura básica mostrada na Figura 9. Nesta seção descrevemos *LuaWebMan*, uma aplicação de gerência baseada nessa arquitetura, desenvolvida utilizando CGILua.

Um dos objetivos do desenvolvimento desta aplicação foi estudar até que ponto a arquitetura apresentada na Figura 9 pode substituir a arquitetura tradicional. Para isso, em primeiro lugar foi desenvolvida uma ferramenta com funções comuns a várias aplicações de gerência, como um *mib browser* e um visualizador de redes. A seguir, partiu-se para a verificação do potencial de extensibilidade de uma ferramenta baseada na arquitetura proposta.

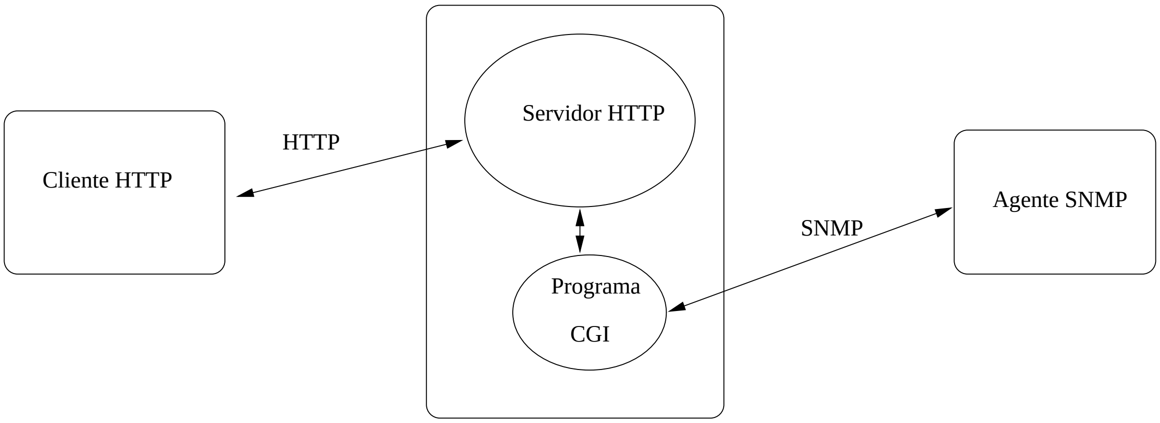


Figura 9: Arquitetura da aplicação baseada em *web*

## 4.1 Funcionalidade Básica

O desenvolvimento inicial do *LuaWebMan* teve por objetivo avaliar a facilidade de implementação de ferramentas básicas de gerência de rede em um ambiente baseado em Web. Para isso, foram escolhidas as três funções seguintes, de caráter bastante distinto entre si, e frequentemente encontradas em aplicações convencionais de gerência:

- descoberta automática dos elementos de uma rede
- visualização do gráfico do tráfego em um agente SNMP
- visualização da meta-informação da MIB (*MIB browser*)

A tela básica do ambiente *LuaWebMan* é composta por três regiões (*frames*), como mostra o esquema da figura 10. A região à esquerda da tela contém o menu de opções, a região superior é reservada para exibição de mensagens e a região central é utilizada para visualização dos equipamentos de rede gerenciados.

Além desta área básica, o *LuaWebMan* utiliza a instanciação de browsers WWW auxiliares, de forma a se disponibilizar novas áreas de trabalho como interface a outros módulos do programa.

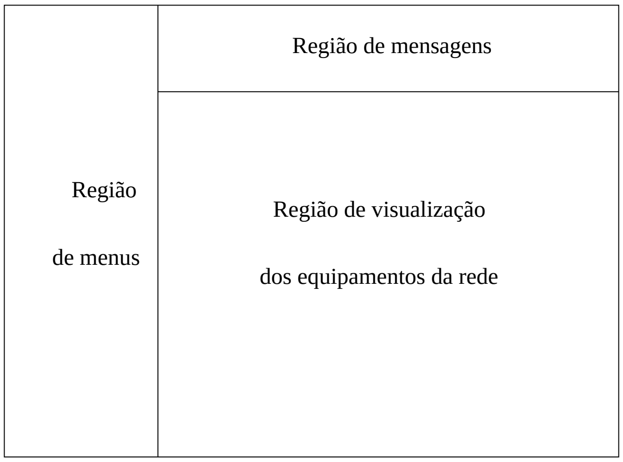


Figura 10: Divisão da interface em 3 regiões.

Como documentos HTML não permitem a definição de menus tipo "pull-down", procurou-se preservar a familiaridade do usuário com este elemento de interface através da utilização de diferentes níveis de menus. Desta forma, quando um ítem da região de menus é selecionado, o conteúdo dessa região é substituído, se for o caso, pelo conjunto de ítens que corresponde ao nível de menu imediatamente inferior. Todos os níveis de menu apresentados contêm, como uma de suas opções, o retorno ao menu de nível imediatamente superior (vide figura 11).

### Descoberta automática dos elementos da rede

Como praticamente todas as aplicações de gerência oferecem algum mecanismo para levantamento dos diversos dispositivos que compõem uma rede, optou-se por incluir no conjunto de ferramentas básicas do *LuaWebMan* uma ferramenta que faz este levantamento e permite monitorar o status dos dispositivos. A ferramenta implementada dá também suporte ao cadastramento de redes já "descobertas", com operações de inclusão, atualização e remoção. A seguir são descritas algumas das funções disponibilizadas por essa ferramenta.

A função de *descoberta* permite o levantamento de uma rede através de seu endereço IP e máscara, utilizando para isto a função icmp_netecho da biblioteca LuaMan. O script Lua correspondente a essa função é bastante parecido com o exemplo apresentado na figura 6 da seção 3.1. A rede descoberta é armazenada de forma permanente, permitindo sua visualização em futuras execuções da aplicação de gerência.

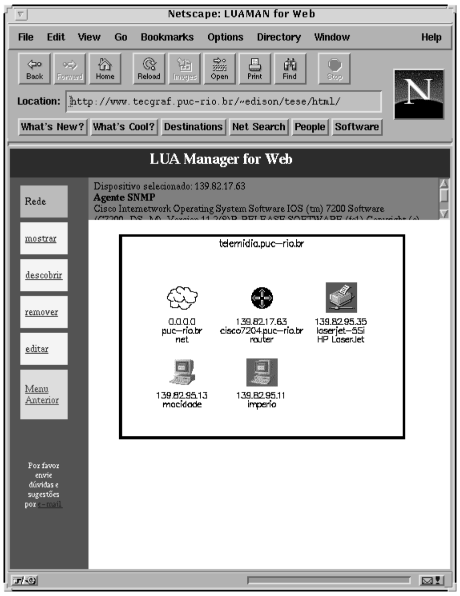


Figura 11: Visualização de uma rede.

Uma função de *exibição* permite a visualização dos equipamentos das redes já cadastradas. Esta visualização é feita segundo um paradigma análogo ao de sistemas de arquivos em árvore, ou seja, uma rede pode ser comparada a um diretório e os dispositivos aos arquivos. Desta maneira a rede é organizada em domínios e subdomínios de acordo com a nomenclatura do DNS. A rede selecionada é apresentada na região de visualização dos equipamentos da rede, como pode ser visto na figura 11. Através do "refresh" periódico da página HTML, o status dos equipamentos da rede é continuamente monitorado, sendo os mesmos exibidos com o fundo vermelho caso estejam inacessíveis e branco caso contrário. A rede visualizada é construída com o uso da biblioteca Imagemaker [Spo97] que constrói em tempo de execução a imagem no formato GIF, juntamente com o mapa clicável do HTML.

Ao se selecionar um determinado equipamento, informações a ele relacionadas serão exibidas na região de mensagens. Dentre essas informações, é indicado se foi possível contactar um agente SNMP associado ao equipamento. O equipamento selecionado pode ter suas informações cadastrais alteradas ou apagadas. Da mesma forma, pode-se incluir um novo equipamento na rede; para isso basta selecionar a rede em que se deseja incluir o novo dispositivo e cadastrá-lo.

A noção de equipamento corrente selecionado também é importante para outras funções do *LuaWebMan*. Ao se requisitar a visualização de tráfego descrita na próxima seção, por exemplo, serão as informações relativas a esse equipamento que serão exibidas.

### Visualização do gráfico do tráfego em um agente SNMP

Uma outra funcionalidade avaliada é a que permite o monitoramento de agentes SNMP através de gráficos do tráfego. Um exemplo de gráfico desse tipo foi construído como uma tabela HTML (figura 12), o que permite um maior desempenho da aplicação, uma vez que a representação de um gráfico na forma de tabela HTML ocupa uma banda passante menor do que se fosse uma imagem no formato GIF ou JPEG. Também é possível se construir gráficos utilizando a biblioteca Imagemaker; neste caso gera-se uma imagem no formato GIF, e os gráficos assim gerados são exibidos em um outro browser. Este esquema funciona analogamente ao sistema de janelas das interfaces gráficas. Como no caso de mapa de equipamentos, os gráficos produzidos pelo Imagemaker também são mapas clicáveis, isto é, as regiões do gráfico podem ser associadas a páginas HTML ou scripts CGILua.

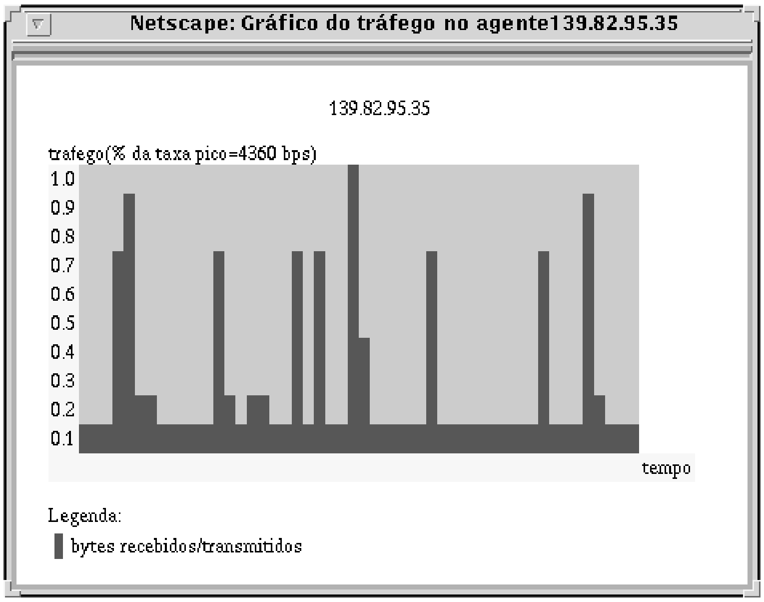


Figura 12: Gráfico do tráfego total em um agente SNMP.

### MIB Browser

Por fim, foi incluída no *LuaWebMan* a funcionalidade mais comum a aplicações de gerência, o *MIB browser*, que permite a visualização da meta informação da MIB atráves de uma árvore. A figura 13 mostra essa função em uso. É interessante observar que esta árvore também foi construída com o auxílio de uma tabela HTML.

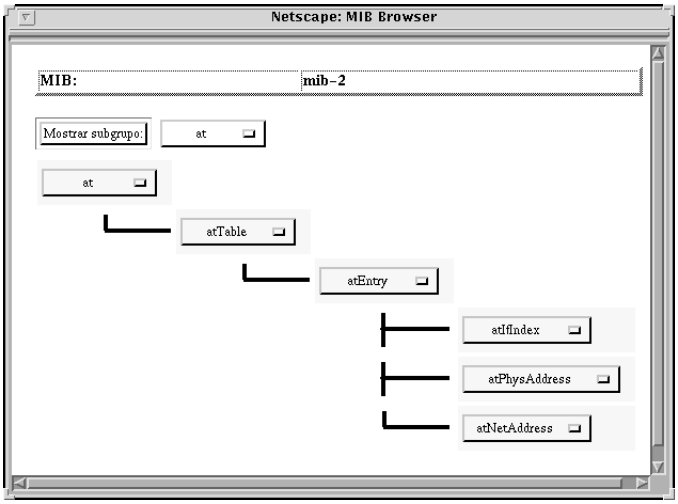


Figura 13: MIB browser.

## 4.2 Extensibilidade

Como já foi discutido, o uso de uma linguagem interpretada e de sintaxe simples como Lua traz inúmeras vantagens quanto à facilidade e potencial de extensibilidade no desenvolvimento de aplicações de gerência. A inovação do *LuaWebMan* em relação a outra plataformas de gerência extensíveis, como o *Scotty* [Sch97], é proporcionar um ambiente de desenvolvimento integrado à aplicação de gerência. Desta forma, não é necessário migrar de ambiente para desenvolver novos módulos de personalização para MIBs proprietárias ou para criar novas funções para a aplicação.

A extensão do *LuaWebMan* é realizada em duas etapas: a especificação do código que implementará a nova funcionalidade e a definição de sua interface de interação, através da associação do código especificado a um novo ítem de menu da aplicação. Por isso, as funções que permitem a extensão da ferramenta de gerência estão divididas em dois grupos: *Scripts* e *Interface*.

As funções necessárias à implementação do código são basicamente as de edição, alteração, gravação e depuração. São estas funções que o grupo *Scripts* implementa. Para elucidar o funcionamento deste ambiente, tomar-se-á como exemplo a implementação de uma função "ping", baseada no uso da primitiva `icmp_echo` da biblioteca LuaMan.

Dois arquivos são utilizados na implementação dessa função. O arquivo `ping.html`, apresentado na figura 14, descreve a interface de entrada de dados (formulário HTML) utilizada para a obtenção dos parâmetros da função. O arquivo `ping.lua`, apresentado na figura 16, contém o script Lua acionado pelo formulário. Os dois arquivos são criados com o uso da função "criar scripts" disponível no menu do grupo *Scripts*. A interface para a criação de scripts é exibida na figura 16.

Para entender o código contido em `ping.html` é necessário fazer uma pequena apresentação das facilidades de CGILua. O código mostrado exibe um exemplo do que foi chamado de *arquivo HTML mesclado*. Este tipo de arquivo contém trechos de código Lua embutidos, como comentários, em um trecho HTML "normal".

O primeiro trecho Lua

```
campos={ {txt='Ping no endereço:', type='text', name='ip', value='0.0.0.0'},
         {txt=nil, type='submit', name='submit', value='Ok'} }
```

cria uma tabela (`campos`) que descreve os campos que comporão o formulário utilizado para a entrada de dados. Cada linha dessa tabela é, por sua vez, uma outra tabela, contendo

```
<HTML>
   <HEAD><TITLE>PING</TITLE></HEAD>
   <BODY BGCOLOR="#FFFFFF">

<!--$$
campos={ {txt='Ping no endereço:', type='text', name='ip', value='0.0.0.0'},
         {txt=nil, type='submit', name='submit', value='Ok'} }
$$-->
      <FORM METHOD="post" ACTION="ping.lua">
<!--$$ LOOP start='i=1', test='campos[i]', action='i=i+1' $$-->
       <P>$|campos[i].txt|$
       <INPUT TYPE="$|campos[i].type|$ NAME="$|campos[i].name|$" VALUE="$|campos[i].value|$">
<!--$$ ENDLOOP $$-->
      </FORM>
   </BODY>
</HTML>
```

Figura 14: Arquivo ping.html.

os atributos do campo correspondente.

A construção especial

```
<!--$$LOOP start='i=1', test='campos[i]', action='i=i+1' $$-->
       [trecho-HTML]
<!--$$ENDLOOP $$-->
```

funciona de maneira análoga ao comando `for` de C. O trecho Lua descrito no campo `start` (`i=1`) é executado antes do primeiro teste da condição do *loop*. O *trecho HTML* é repetido enquanto a condição for verdadeira. O trecho Lua descrito no campo `action` (`i=i+1`) é repetido a cada iteração. O resultado da execução do código de `ping.html` é a criação do formulário exibido na figura 15.

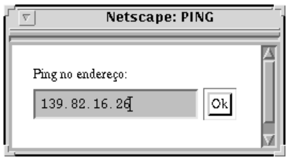


Figura 15: Formulario para entrada de dados da função ping

O script Lua acionado pelo formulário utiliza a função `icmp_echo` para verificar a acessibilidade do dispositivo cujo endereço IP é fornecido, produzindo, como resultado, uma página HTML, conforme apresentado na figura figura 17.

*LuaWebMan* provê também uma função para depuração do código implementado por seu usuário, permitindo a realização de testes dos novos scripts criados. Ao se selecionar o teste, o script é executado pelo servidor WWW e o seu resultado é exibido em um outro browser WWW, que simula uma interface de exibição. Caso não haja erros no código, o documento HTML gerado é exibido; do contrário, o interpretador CGILua gera um documento HTML com as mensagens de erro geradas pelo interpretador Lua para que se possa depurar o script.

É importante ressaltar que o script CGILua gerado no browser WWW é transmitido até o servidor WWW para lá ser executado. A forma de extensão do *LuaWebMan* inverte a direção de transmissão usual dos códigos móveis [Con97]. Códigos móveis, como Java e JavaScript, são usados para distribuir aplicações pela Web, transmitindo o código do servidor WWW em direção ao browser para sua execução. As linguagens utilizadas para

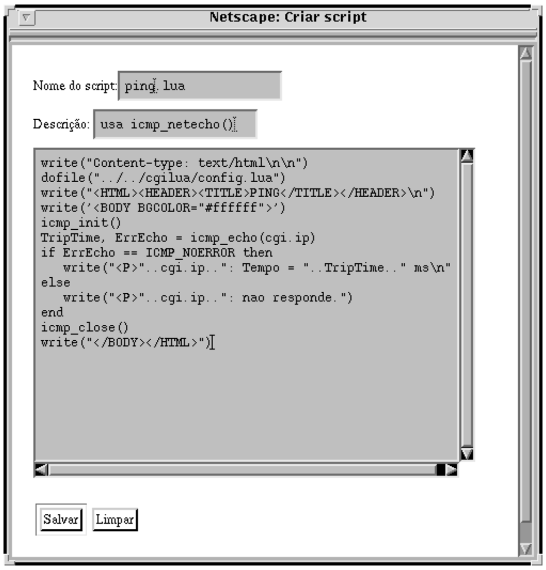


Figura 16: Arquivo ping.lua no ambiente de desenvolvimento.

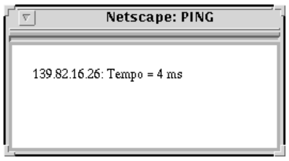


Figura 17: Resultado da execução da função criada.

implementar códigos móveis se preocupam muito com a segurança, uma vez que estes códigos serão executados em máquinas distribuídas pelas rede, e seus usuários esperam que o aplicativo distribuído não venha a provocar nenhum dano. No caso do *LuaWebMan*, o código é gerado para ser executado pelo servidor. Se por um lado, pode-se imaginar que as possibilidade de dano são ainda maiores no servidor, por outro cada servidor pode ter uma configuração local do CGILua adequada às suas necessidades de segurança. Funções potencialmente perigosas podem ser redefinidas, podendo ser substituídas por versões mais seguras [HBI97]; isso permite que os programas CGI sejam executados em um ambiente "protegido".

Voltando novamente ao exemplo, uma vez especificado o código que implementa a nova função "ping", é necessário associá-lo a um ítem de menu, de forma a disponibilizar sua utilização através da interface do *LuaWebMan*.

No grupo *Interface* são agrupadas as funções que manipulam a interface da ferramenta de gerência. Para que a interface do *LuaWebMan* possa ser reconfigurável, parte dela é gerada dinamicamente através de scripts. A parte reconfigurável foi escolhida levando-se em conta a necessidade de se reconfigurar apenas os elementos da interface que não descaracterizassem a ferramenta de gerência. As informações da parte reconfigurável são armazenadas em tabelas Lua persistentes.

Para incorporar a nova função "ping" à interface do *LuaWebMan*, o código correspondente deve ser associado a um menu (ou *grupo*) existente ou a um menu a ser criado. Suponha que o usuário optou por criar o grupo "ICMP", para agrupar as funções que façam uso deste protocolo. Neste caso, ele deverá seguir os passos mostrados na figura 18. Em primeiro lugar, ele deve utilizar a função *criar grupo*, disponível no menu principal

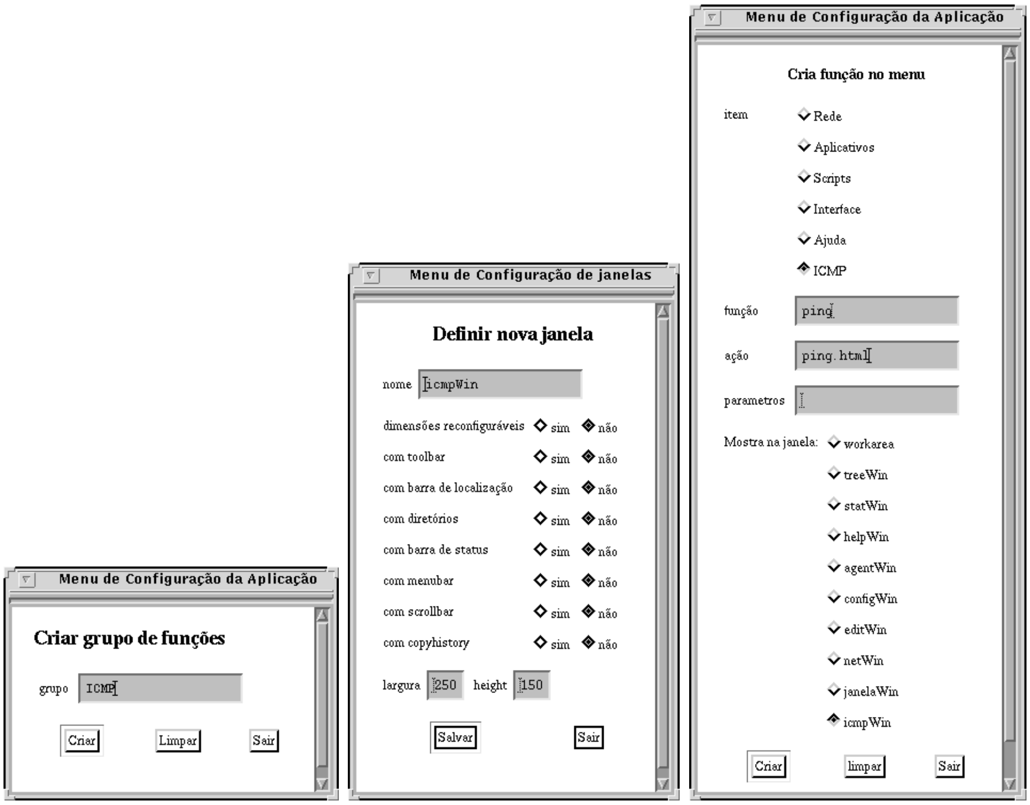


Figura 18: Passos para se criar uma nova função.

de *Interface*. Em seguida, ele deve selecionar a função *janela* para configurar a interface que irá exibir as telas geradas pela nova função. Por fim, através da opção *criar função*, o usuário associa o código criado, neste caso o código contido no arquivo `ping.html`, a um ítem do novo grupo criado.

É importante enfatizar que a operação descrita causa não só uma alteração na interface do *LuaWebMan*, que passa a conter um item "ICMP" em seu menu, como também incorpora à aplicação uma nova funcionalidade. A aplicação de gerência pode assim ir sendo dinamicamente adaptada às necessidades do administrador.

# 5 Considerações Finais

Apresentamos neste trabalho uma plataforma de desenvolvimento de aplicações de gerência extensíveis, baseada em uma linguagem de programação interpretada e de fácil aprendizado. Em [Lim98] são apresentados diversos exemplos de pequenas ferramentas típicas de gerência, desenvolvidas nesta plataforma como forma de verificar o grau de facilidade oferecido por ela. Os resultados foram considerados muito satisfatórios.

Com o desenvolvimento da ferramenta *LuaWebMan*, acreditamos, por diversos motivos, ter realizado um passo ainda mais importante de validação deste ambiente. Em primeiro lugar, o autor de *LuaWebMan* [Ish98] não foi um dos envolvidos na construção de LuaMan, representando assim o primeiro "usuário externo" da plataforma. Em segundo lugar, a área de gerência baseada em *web* ainda se encontra muito pouco desenvolvida. Segundo [Ste97], as poucas ferramentas no mercado que permitem interação do usuário com agentes SNMP em tempo real via *web* ainda apresentam um número bastante grande de limitações. [Jan96] discute as dificuldades de desenvolvimento para este ambiente. Neste contexto, o fato de se ter logrado obter não apenas a "funcionalidade padrão" de aplicações tradicionais de gerência como também a extensibilidade da ferramenta, com a possibilidade de agregação dinâmica de nova funcionalidade, nos parece especialmente importante.

**Agradecimentos** Este trabalho contou com o apoio do CNPq, através de bolsas e outros recursos disponibilizados no escopo do projeto GERENTE, da fase III do Protem. O desenvolvimento da biblioteca LuaMan e do *LuaMan for Web* foi parcialmente realizado nos laboratórios Telemídia e TeCGraf, da PUC-Rio, e no NC-RJ da RNP.

## Referências Bibliográficas

[Ban97] Bela Ban. *A Generic Management Model for CORBA, CMIP and SNMP*. PhD thesis, University of Zurich, 1997.

[Ben88] J. Bentley. *More programming pearls*. Addison-Wesley, 1988.

[Con97] D. Connolly. W3C Consortium – Mobile Code, 1997. disponível por www em `http://www.w3.org/pub/WWW/MobileCode/`.

[CRI97] R. Cerqueira, N. Rodriguez, and R. Ierusalimschy. Binding an interpreted language to corba. In *Anais do SBLP97*, pages 23–36. SBC, 1997.

[dbl] Biblioteca DBLua. `http://www.tecgraf.puc-rio.br/manuais/dblua`.

[HBI97] A.M. Hester, R. Borges, and R. Ierusalimschy. CGILua: A multi-paradigmatic tool for creating dynamic www pages. In *XI Simpósio Brasileiro de Engenharia de Software (SBES'97)*, Fortaleza, 1997. SBC.

[ICR98] Roberto Ierusalimschy, Renato Cerqueira, and Noemi Rodriguez. Using reflexivity to interface with CORBA. In *International Conference on Computer Languages 1998*, Chicago, 1998. IEEE. to appear.

[IFC96] R. Ierusalimschy, L. Figueiredo, and W. Celes. Lua - an extensible extension language. *Software: Practice and Experience*, 26(6), 1996.

[IFC97] R. Ierusalimschy, L. Figueiredo, and W. Celes. Lua—an extensible extension language. In *II Prêmio Compaq de de Estímulo à Pesquisa e Desenvolvimento em Informática*, pages 174–189. Academia Brasileira de Ciências/Instituto UNIEMP, 1997. (1o lugar na Categoria Tecnológica).

[Ish98] Edison Ishikawa. Gerência de redes baseada em *web*. Master's thesis, Depto de Informática, PUC-Rio, 1998. a ser submetida.

[Jan96] Mary Jander. Welcome to the revolution. *Data Communications*, 25(16), 1996.

[KC96] Q. Kong and G. Chen. Integrating corba and tmn environments. In *Network Operations and Management Symposium*. IEEE/IFIP, 1996.

[LFG+96] C. Levy, L. Figueiredo, M. Gattass, C. Lucena, and D. Cowan. IUP/LED: a portable user interface development tool. *Software Practice and Experience*, 26(7):737–762, 1996.

[Lim98] Michele E. Lima. LuaMan: uma plataforma para desenvolvimento de aplicações de gerenciamento extensíveis. Master's thesis, Depto de Informática, PUC-Rio, January 1998.

[ML97] A. Moura and M. Lima. A API de Gerência de Redes LuaMan, 1997. http://www.telemidia.puc-rio.br/~ana/luaman.html.

[Moc87] P. Mockapetris. Domain names - concepts and facilities, 1987. RFC 1034.

[Obj95] Object Management Group, Inc., Framingham, MA. *The Common Object Request Broker Architecture and Specification; Revision 2.0*, jul 1995.

[Pos81] J. Postel. Internet control message protocol, 1981. RFC 792.

[RCIM97] N. Rodriguez, R. Cerqueira, R. Ierusalimschy, and A. Moura. Aplicações de gerência com comportamento dinâmico. In *Anais do SFBSID'97*, pages 476–487, Fortaleza, Brasil, 1997.

[Ros96] Marshall Rose. Industry comment. *The Simple Times*, 4(3), 1996.

[Sch97] J. Schonwalder. Scotty - Tcl Extensions for Network Management Applications, 1997. http://wwwsnmp.cs.utwente.nl/~schoenw/scotty/.

[SL93] J. Schonwalder and H. Langendorfer. How to keep track of your network configuration. In *proceedings 7th Conference on Large Installation System Administration (LISA VII)*, Monterey, California, 1993.

[SL95] J. Shonwalder and L Langendorfer. Tcl extensions for network management. In *proceedings 3rd Tcl/Tk Workshop*, Toronto, Canada, 1995.

[Spo97] Flavio Spolidoro. Imagemaker for network management, 1997. Projeto Final de Curso. Depto de Informática, PUC-Rio. Descrição disponível por www em http://www.tecgraf.puc-rio.br/~spol/imagemaker/.

[Sta96] William Stallings. *SNMP, SNMPv2, and RMON: Practical Network Management*. Addison-Wesley, 1996.

[Ste97] Steve Steinke. Network management meets the web. *Network*, 12(12):44–50, 1997.