4
Implementação

Este capítulo visa relatar detalhes importantes da implementação das abordagens descritas no capítulo 3. Como prova de conceito foi utilizado o ORB OiL (*ORB in Lua*) [12], escrito em Lua [13] [14], e os elementos procuradores (*proxies*) desenvolvidos foram todos também escritos em Lua.

A seção 4.1 irá detalhar a sintaxe/semântica do arquivo XML de configuração, enquanto que as demais seções irão tratar da implementação de cada abordagem separadamente.

4.1
Arquivo de Configuração

Toda a configuração a ser realizada pela aplicação CORBA para travessia de firewalls/NAT deve ser feita através de um arquivo XML a ser escrito pelo desenvolvedor da mesma. Um dos objetivos desse arquivo é não sobrecarregar o código da aplicação com informações de infra-estrutura. No apêndice A.1 está listado o DTD deste arquivo XML.

O elemento raiz `firewall-traversal` indica uma configuração específica de travessia de firewall/NAT. Ele possui um atributo obrigatório de nome `choice` que pode conter ou o valor `omg` ou o valor `proxy`, sendo o primeiro indicativo da abordagem OMG (seção 3.1) e o segundo indicativo da abordagem Procurador TCP (seção 3.2) ou Procurador HTTP (seção 3.3).

Caso o atributo `choice` contenha o valor `omg`, o elemento `firewall-traversal` deverá conter elementos filhos indicando caminhos de entrada (`inbound-path`) e/ou saída (`outbound-path`). Cada elemento desses deve conter uma sequência de elementos filhos do tipo `element` onde, por sua vez, cada um representa uma estação no caminho entre a rede externa

e o ORB ou vice-versa. No caso do elemento `inbound-path`, os elementos filhos devem estar na ordem da estação mais externa para a estação hospedeira do ORB, inclusive. Para o caso do elemento `outbound-path` os elementos devem estar na ordem da estação hospedeira do ORB (não incluindo a mesma) até a estação mais externa. O elemento `element` possui um atributo `is-intellingent` que pode conter os valores `true` ou `false` indicando a capacidade ou não de tratamento do protocolo IIOP. `element` pode conter como filhos um ou mais elementos `endpoint` que representam pontos de acesso à estação. Este elemento contém três atributos chamados `service`, `address` e `port` que indicam o tipo de servico [5], o endereço IP (ou nome DNS) e a porta do ponto de acesso, respectivamente. O apêndice A.2 contém um exemplo dessa configuração.

Caso o atributo `choice` contenha o valor `proxy`, isso indica o uso da abordagem Procurador TCP ou da abordagem Procurador HTTP. Neste caso deve existir um elemento de nome `proxy` que contém um atributo obrigatório chamado `type`, que irá indicar o tipo de procurador escolhido podendo conter o valor `tcp` ou `http`. Outros dois atributos obrigatórios, `address` e `port`, são comuns às duas formas de procuração. O primeiro indica o endereço IP (ou nome DNS) do procurador e o segundo a sua porta. O atributo `polling-interval` é exclusivo da abordagem Procurador HTTP e deve conter um valor inteiro indicando o intervalo de tempo máximo (em milisegundos) entre duas consultas HTTP feitas pelo ORB. Um exemplo desta configuração é apresentado no apêndice A.3.

Além de escrever o arquivo de configuração XML, o desenvolvedor da aplicação precisa informar ao OiL a existência deste arquivo, definindo uma variável global do interpretador Lua chamada `_FIREWALL_TRAVERSAL_CONF_FILE`. Esta variável deve conter como valor o caminho deste arquivo para o ORB proceda com a configuração escolhida.

4.2 Abordagem OMG

Para implementar a proposta descrita na seção 3.1 são necessárias modificações tanto no ORB servidor quanto no ORB cliente a fim de habilitá-los a trabalhar com as estruturas e protocolos definidos pela OMG na especificação [5]. Além dessas modificações, é necessário desenvolver o procurador de aplicação que será responsável por dar suporte aos ORBs da rede interna

poderem atravessar firewalls/NAT. A seção 4.2.1 irá detalhar as modificações feitas no ORB enquanto que a seção 4.2.2 irá relatar aspectos relevantes na implementação do procurador de aplicação.

4.2.1
Adaptações no ORB

Para implementar a especificação da OMG é necessário primeiramente que o ORB suporte a versão 1.1 do protocolo IIOP, pois é nela que é definida a sequência de componentes rotulados (`IOP::TaggedComponent`) a ser inserida no perfil IIOP do IOR. É nessa sequência que se encontrará o componente de travessia de firewall (`TAG_FIREWALL_PATH`). Como o OiL não possuia esse suporte foi necessário portanto adaptar a geração do IOR para trabalhar com esse elemento.

No lado do ORB responsável pelo objeto servidor, a única modificação necessária foi criar o componente rotulado `TAG_FIREWALL_PATH` no perfil IIOP do IOR no momento de criação do objeto CORBA. Ao ser solicitado pela aplicação a criar um objeto CORBA, o ORB verifica se existe a opção de travessia de firewall e se a abordagem escolhida é a abordagem OMG. Em caso positivo, o ORB lê do arquivo de configuração os elementos existentes entre a rede externa e o ORB inclusive e procede com a construção do componente e a sua inserção no perfil IIOP do IOR. A partir daí, todo o processamento no ORB servidor se dá de forma normal, ou seja, a requisição CORBA é recebida e atendida como uma requisição usual sem o ORB tomar ciência de que ela chegou através da travessia de firewall.

Na parte do ORB responsável pelo cliente, foi necessário modificar o código relativo à criação da conexão com o ORB servidor no momento de uma invocação remota de método. Na versão original o ORB analisa o perfil IIOP do IOR e se conecta diretamente ao endereço/porta indicado neste campo. Na versão modificada ele deve antes verificar se existe o componente rotulado `TAG_FIREWALL_PATH` e, em caso positivo, proceder com a negociação de sessão de travessia de firewall. Conforme foi visto na seção 3.1, essa negociação consiste em: construir a entrada de contexto de serviço `FIREWALL_PATH`; enviar a mensagem GIOP NegotiateSession ao primeiro elemento entre o ORB cliente e o servidor; e aguardar a resposta positiva. Uma fez feito isso, a operação original do ORB pode prosseguir normalmente, visto que cliente e servidor devem seguir com o protocolo tradicional de invocação remota (GIOP) pela

conexão aberta durante a negociação de sessão.

A fim de se evitar a análise do IOR e do arquivo de configuração XML a cada invocação remota, é mantido em memória tanto os componentes rotulados já analisados quanto o arquivo XML de configuração.

4.2.2
Implementação do Procurador de Aplicação

Conforme dito anteriormente, o procurador de aplicação foi desenvolvido em Lua e projetado de forma a atender simultaneamente vários clientes e servidores e não bloquear em nenhuma operação de entrada/saída. Para implementar a concorrência no atendimento a clientes e servidores foram utilizadas co-rotinas, um recurso oferecido pela linguagem Lua [14] [15]. A arquitetura do procurador de aplicações é mostrada na figura 4.1, onde as setas indicam a ordem de criação das co-rotinas.

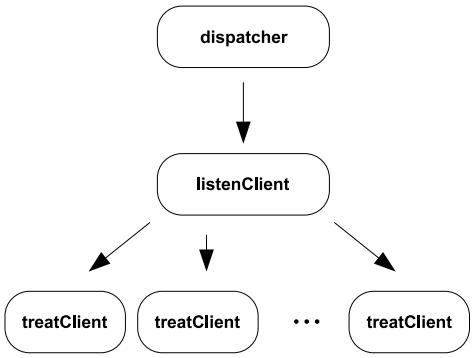

Figura 4.1: Arquitetura do Procurador de Aplicação

Na figura 4.1 cada co-rotina é representada por um retângulo de bordas arredondadas. Existe uma co-rotina principal chamada `dispatcher`, que tem como função principal escolher a próxima co-rotina a ser reiniciada. A primeira coisa que esta co-rotina faz é criar outra auxiliar, chamada `listenClient` que é responsável por aguardar novas conexões de ORBs clientes interessados em negociar uma sessão de travessia de firewall. Estas duas co-rotinas executam durante toda a existência do procurador. A cada conexão de um novo cliente, a co-rotina `listenClient` cria outra (identificada na figura por `treatClient`) que é responsável por tratar a conexão recém-estabelecida. Todas as co-rotinas retornam o controle para `dispatcher` quando bloqueiam em uma operação de entrada/saída. Neste bloqueio o controle de execução é passado para `dispatcher` junto com a conexão bloqueada, a fim de que a co-rotina

execute uma operação `select` [16], reiniciando posteriormente as co-rotinas que tenham dados a serem lidos.

Ao implementar a especificação um problema surgiu com base na omissão do texto da OMG sobre a mensagem GIOP de negociação de sessão `NegotiateSession`. A especificação indica a sua existência na versão do protocolo GIOP 1.3 e faz referência à seção aonde ela estaria definida na especificação [1]. Esta seção não existe e buscas na Internet (à época da escrita deste texto) não foram suficientes para encontrar a definição da mesma (em IDL), obtendo-se apenas mensagens em listas de discussão acerca de sua natureza bi-direcional. Tendo em vista este problema, definimos uma mensagem `NegotiateSession` simples, contendo apenas um campo para a sequência de entradas de contexto de serviço [6], visto que esta é a única informação necessária para proceder com a negociação de sessão. É nesta seqüência que a entrada de contexto de serviço `FIREWALL_PATH` será inserida. A IDL da mensagem é mostrada a seguir:

```
struct NegotiateSession_1_3 {
  IOP::ServiceContextList service_context;
}
```


4.3 Abordagem Procurador TCP

A fim de implementar a abordagem descrita na seção 3.2, é necessário tanto modificar o ORB responsável pelo objeto CORBA como desenvolver o procurador em questão. O ORB do lado cliente não requer nenhuma modificação, visto que para o mesmo esta abordagem é totalmente transparente. Nas próximas seções serão detalhados aspectos relevantes na execução destas duas tarefas.

4.3.1 Adaptações no ORB

A única modificação necessária no ORB responsável pela criação do objeto CORBA servidor é registrar este último no procurador, no momento

de sua criação. Este registro é feito através do envio da mensagem `Register` e espera da respectiva resposta `RegisterReply` (ver seção 3.2). Ao ser criado o objeto, o ORB verifica se existe a opção de travessia de firewall e se a abordagem escolhida foi a de Procurador TCP. Em caso positivo, ele procede com o registro no procurador e aguarda a resposta antes de retornar o controle à aplicação. Caso seja este o primeiro registro, o ORB também cria a conexão de dados e a inclui na lista de conexões a serem monitoradas por novas requisições. A partir de então todo o processamento se dá como se o procurador fosse o cliente das requisições que chegam por esta conexão, não exigindo nenhuma outra modificação no código do ORB.

Além disso, foi criado um método chamado `_get_ior_exported` no objeto CORBA retornado pelo ORB à aplicação. Este método é similiar ao método existente `_get_ior`, com a diferença de que o primeiro retorna o IOR criado pelo procurador e que pode ser exportado para a rede externa (esse IOR foi recebido pela mensagem `RegisterReply`). O objetivo deste método é permitir à aplicação o acesso ao seu IOR exportado e flexibilizar a publicação do mesmo.

4.3.2
Implementação do Procurador TCP

Da mesma forma como o procurador de aplicação descrito na seção 4.2.2, o procurador TCP foi desenvolvido em Lua e projetado de forma a atender simulteamente vários clientes e servidores e não bloquear em nenhuma operação de entrada/saída. Co-rotinas foram utilizadas para implementar a concorrência no atendimento a clientes e servidores e a arquitetura do procurador é mostrada na figura 4.2.

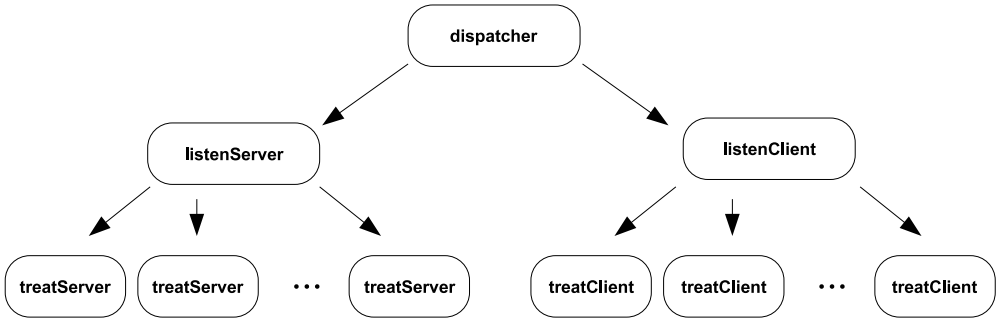

Figura 4.2: Arquitetura do Procurador TCP

Cada retângulo de bordas arredondadas mostrado na figura 4.2 repre-

senta uma co-rotina e as setas indicam a ordem de criação das mesmas. A co-rotina principal é a `dispatcher` e ela é responsável por escolher qual a próxima co-rotina a ser reiniciada. Todas as outras co-rotinas retornam o fluxo de controle para a co-rotina `dispatcher` assim que elas bloquearem em uma operação de entrada/saída. Nesse retorno a co-rotina bloqueada devolve para a co-rotina `dispatcher` a conexão na qual está aguardando a chegada de dados. Esta, por sua vez, entra em uma operação `select` [16] nestas conexões, reiniciando as co-rotinas a medida que novos dados cheguem nas conexões correspondentes.

Inicialmente a co-rotina `dispatcher` cria duas co-rotinas auxiliares identificadas por `listenServer` e `listenClient`, que são responsáveis por monitorar novas conexões de ORBs servidores e ORBs clientes, respectivamente. Essas três co-rotinas vão existir durante todo o funcionamento do procurador. Uma vez criadas, a cada estabelecimento de uma nova conexão de ORB a co-rotina correspondente será reiniciada pela `dispatcher` e criará uma co-rotina temporária para tratamento da mesma.

No caso de ORBs servidores, a co-rotina identificada por `listenServer` irá criar co-rotinas identificadas por `treatServer`, que farão ou o registro de objetos, ou a abertura de canal de requisições/respostas. Neste último caso a co-rotina criada irá abrir a conexão e aguardar por mensagens GIOP de resposta advindas do ORB servidor, que serão tratadas e encaminhadas ao ORB cliente correspondente.

No caso de ORBs clientes a co-rotina `listenClient` irá criar uma co-rotina temporária identificada por `treatClient`, que será responsável por repassar a requisição GIOP para a conexão de dados aberta com o ORB responsável pelo objeto CORBA. Depois desse encaminhamento, a conexão com o cliente é armazenada e a co-rotina é finalizada. A resposta a esta requisição eventualmente será recebida pela co-rotina reponsável pela abertura da conexão de dados com o ORB correspondente (`treatServer`) e será encaminhada ao cliente pela conexão armazenada anteriormente.

4.4 Abordagem Procurador HTTP

O processo de implementação da abordagem Procurador HTTP requereu tanto a modificação do ORB responsável pelo objeto CORBA quanto a construção do procurador desta abordagem. As próximas seções detalham este

esforço.

4.4.1
Adaptações no ORB

As modificações necessárias no ORB responsável pelo objeto CORBA são bem mais complexas do que aquelas requeridas pela abordagem Procurador TCP. Assim como nesta última, é necessário alterar o ORB para registrar o objeto CORBA junto ao procurador no momento de sua criação. A complexidade desta abordagem deve-se à monitoração por novas requisições. No caso da abordagem Procurador TCP bastava monitorar a conexão com o procurador para o recebimento de mensagens GIOP, da mesma forma como é feito com as outras conexões normalmente. No caso do ORB OiL, é suficiente adicionar esta conexão a uma lista de conexões que entram em uma operação `select` [16], todas à espera de mensagens GIOP. No caso da abordagem Procurador HTTP, as mensagens GIOP advindas da rede externa chegam ao ORB encapsuladas em pacotes HTTP através de uma ação de consulta iniciada pelo ORB servidor. É necessário portanto que a monitoração por novas mensagens GIOP seja feita intercalando a operação `select` nas conexões usuais (sem a opção de travessia), com a consulta (*polling*) HTTP ao procurador. O intervalo de consulta (`polling-interval`) definido no arquivo de configuração (ver seção 4.1) é usado para limitar a duração da operação `select`, de modo que o intervalo de consulta especificado no arquivo de configuração seja respeitado.

Outro ponto que merece destaque na implementação desta abordagem é a questão do recebimento de mais de uma requisição GIOP durante uma consulta HTTP. Conforme foi visto na seção 3.3.1, a fim de se reduzir o número de mensagens trocadas, um pacote HTTP de resposta pode trazer mais de uma requisição GIOP. Esta característica obriga o ORB a armazenar requisições para posterior tratamento, ou seja, em uma nova operação de consulta por novas requisições, deve-se antes verificar se não há alguma requisição armazenada localmente, evitando assim tanto a operação `select` quanto a consulta HTTP.

4.4.2
Implementação do Procurador HTTP

O procurador desenvolvido para esta abordagem é similar ao descrito na seção 4.3.2. Ele também foi implementado em Lua usando co-rotinas para atender concorrentemente mais de um ORB. A arquitetura é a mesma mostrada anteriormente na figura 4.2: existe uma co-rotina principal chamada `dispacher` que é responsável por escolher a próxima co-rotina a ser executada; duas co-rotinas auxiliares (`listenServer` e `listenClient`) são responsáveis por escutar novas conexões de ORBs servidores e clientes e por sua vez criam co-rotinas temporárias para tratamento de um ORB específico; por fim a co-rotina `treatServer` é responsável por tratar uma requisição de um ORB servidor (consulta por requisições, envio de resposta GIOP, ...), enquanto que a co-rotina `treatClient` é encarregada de receber uma requisição GIOP de um cliente e armazená-la para posterior envio ao ORB correspondente.