Web-Services in Perl

Alberto Manuel Brandão Simões
ambs@di.uminho.pt

1 de Junho de 2008

SOAP

- protocolo para intercâmbio de mensagens baseadas em XML;
  - sucessor do XML-RPC;
- normalmente com protocolo de transporte HTTP ou HTTPS.
- habitualmente usando um padrão RPC:

Arquitectura

- cliente envia uma mensagem com um pedido ao servidor;
- servidor envia imediatamente uma mensagem com a resposta;

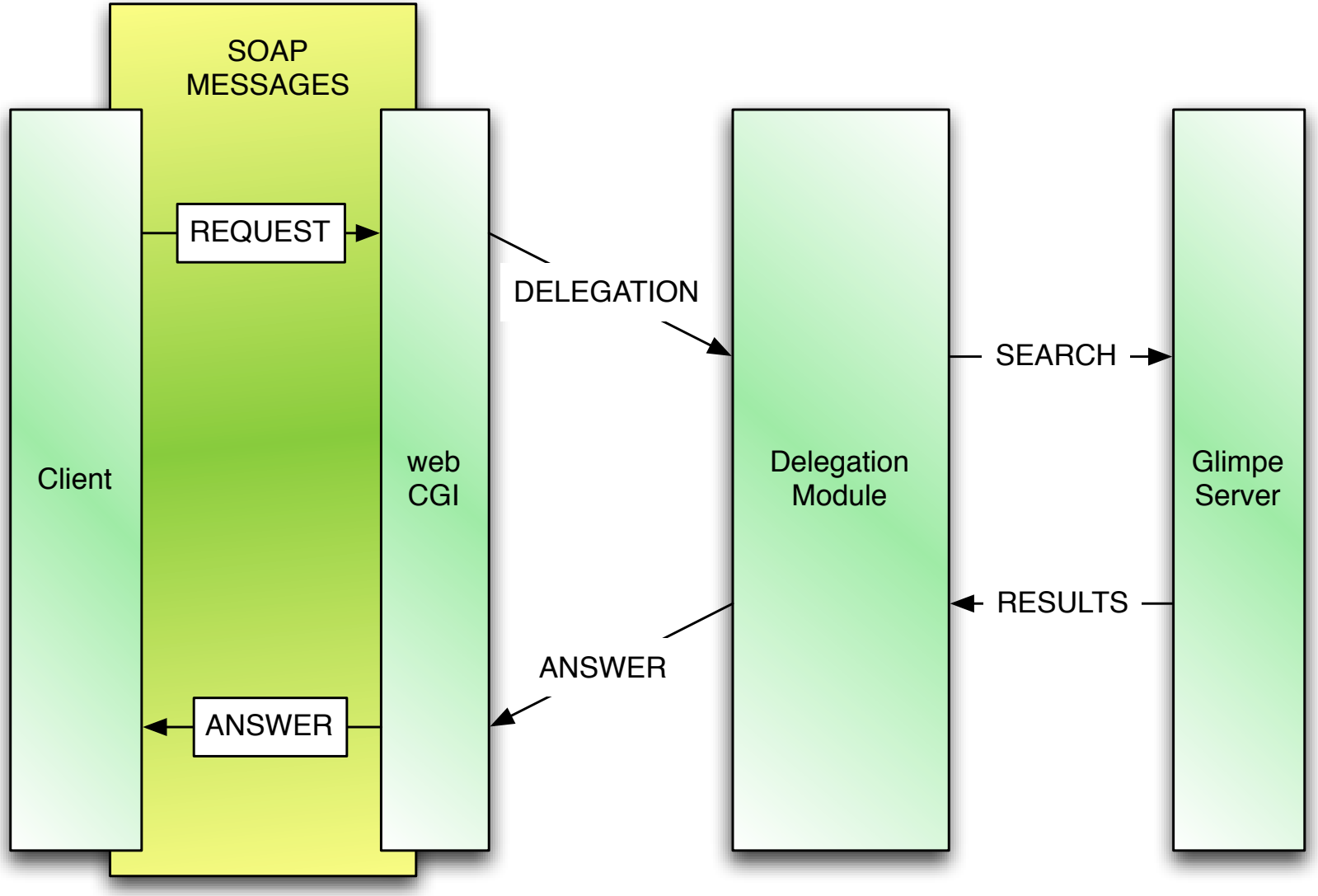
Servidor de Memórias de Tradução Distribuídas

Alberto Simões, Xavier Gómez Guinovart, and José João Almeida. Distributed translation memories implementation using webservices. Procesamiento del Lenguaje Natural, 33:89–94, July 2004.

O servidor:

- armazena unidades de tradução: $(\mathrm{S}_{\mathcal{A}} \times \mathrm{S}_{\mathcal{B}})^{\star}$;
- responde à seguinte API:
  - $\mathrm{traduzes?} : \quad \mathrm{L} \times \mathrm{L} \longrightarrow \mathbb{B}$
  - $\mathrm{traduz} : \quad \mathrm{L} \times \mathrm{L} \times \mathrm{S} \longrightarrow \mathrm{S}$
  - $\mathrm{traducoes} : \quad \mathrm{L} \times \mathrm{L} \times \mathrm{S} \longrightarrow \mathrm{S}^{\star}$

Arquitectura



SOAP

Search::Glimpse

CGI: Receber pedidos HTTP

```
#!/usr/bin/perl -w

# indicamos qual o protocolo de transporte que pretendemos utilizar
use SOAP::Transport::HTTP;

# redirecciona qualquer pedido para o módulo "disttmx"
SOAP::Transport::HTTP::CGI
-> dispatch_to('disttmx')
-> handle;
```

PM: Processar os pedidos SOAP

```
package disttmx;

use Search::Glimpse;
use strict;

our %known_languages = (
        en => [qw/pt/]
       );
sub traduzes {
    my ($self, $l1, $l2) = @_;

    if (exists($known_languages{$l1})) {
      if (grep { $_ eq $l2 } @{$known_languages{$l1}}) {
return 1;
      }
    }
    return 0;
}
```

PM: Processar os pedidos SOAP

```
sub traduz {
  my ($self, $l1, $l2, $string) = @_;

  my $x = Search::Glimpse->new('server'  => 'holst.di.uminho.pt',
       'nr_hits' => 1);

  my @answers  = $x->search($string);

  my $i = 0;
  for (@answers) {
    $i++;
    s!^[^:]+:!!;
    s/<tu.*<tu id="[^"]+">//;
    s!</tu.*$!!;
  }

  if ($answers[0] eq "ERROR")
    { return undef }
  else
    { return $answers[0] }
}
```

PM: Processar os pedidos SOAP

```
sub traducoes {
  my ($self, $l1, $l2, $string) = @_;

  my $x = Search::Glimpse->new('server'  => 'holst.di.uminho.pt',
       'nr_hits' => 20);

  my @answers  = $x->search($string);
  return undef if $answers[0] eq "ERROR";

  @answers = map { s!^[^:]+:!!;
   s!^\s*<[^>]+>!!;
   s!</tu>$!!;
   s!</tu><tu[^>]+>!#XPTO#!;
   [ split /#XPTO#/ ]
 } @answers;
return [@answers];
}
```

Cliente: Enviar pedido SOAP

```
#!/usr/bin/perl

    use Data::Dumper;
use SOAP::Lite;

my $soapserver = SOAP::Lite
  -> uri('http://localhost:80/disttmx')
  -> proxy('http://localhost:80/cgi-bin/disttmx.cgi');

my $soapresult = $soapserver->traducoes("pt","en","european parliament");

unless($soapresult->fault) {
  my $result = $soapresult->result();

  if ($result) {
    print Dumper($result);
  } else {
    print STDERR "** server does not know how to translate it **\n";
  }
} else {
  print $soapresult->faultcode,": ",$soapresult->faultstring;
}
```

Resumindo

- comunicação é feita em XML, mas de forma transparente:
  - métodos invocados como métodos nativos;
  - tipos de dados nativos;
- para que duas aplicações possam comunicar é necessário definir uma API comum:
  - conjunto de métodos suportados;
  - tipo de dados recebido por cada método;
  - tipo de dados retornado por cada método;