Jax 2002

„Kommunikation zwischen Delphi und PHP, CBuilder und PHP mit XML−RPC und Performancevergleich zwischen einer PHP−SOAP−Implementation und der Implementation in Borland CBuilder 6 als CGI bzw. DSO“

Jochen Stärk, Borland
jstaerk@borland.com

Inhaltsverzeichnis

Sinn und Unsinn von Web Services

Da auch in diesem Vortrag auf Web Services eingegangen werden soll, muss der Nutzen von Web Services natürlich begründet werden.

„Ein Programmierer mit einem Web Service ist wie ein Esel mit einem Spinnrad – keiner weiß, woher er's hat und er weiß nicht, was er damit anfangen soll."

Indisches Sprichwort

Web Services sind ähnlich Corba Plattform− und Programmiersprachenunabhängig, können jedoch mit weniger Aufwand erstellt (anstelle des ORB tritt ein Web Server) und benutzt werden (durch das HTTP−Protokoll können SOAP−Aufrufe nach außen durch Firewalls hindurch erfolgen). Die Bibliotheken zum Benutzen von Web Services sind schlanker, weil Sie sich auf Standards wie HTTP und XML stützen und durch den Internet−Einsatz sind viele Web Services nicht nur für bestimmte Unternehmensgruppen einsetzbar sondern öffentlich.

Sinnvolle Web Services

z.B.: Weitergabe von Informationen gegen Entgelt
Web Service für Echtzeit−Börsendaten, Telefonnummern, „Content" also z.b. Tagesnachrichten o.ä. an angemeldete Benutzer sowie die Bereitstellung eines Produktkatalogs und eines Bestellsystems

z.B.: Dienstleistungen gegen Entgelt
−Web Service für Kreditkartenzahlungen an angemeldete Benutzer.

z.B.: Web Services zur Unterstützung eigener Entwicklungen
−Web Services zum Überprüfen auf neue Programmupdates, zum Anzeigen von Neuigkeiten über Produkte und zum kommunizieren (Bestellung, Katalog, Administration) mit eigenen Online−Angeboten

Ein proprietärer PHP Web Service mit Delphi Client

Eine Möglichkeit, mit PHP zu kommunizieren, ist ein selbstdefiniertes Datenformat über HTTP. Die PHP−Seite erhält beim Aufruf per Get Parameter, die sie verwendet um eine Ausgabe zu bilden.

Vorteil: Selbst wenn die Ausgabe als XML erfolgt, braucht man kein XML−Support in PHP zu kompilieren, weil XML−Erstellung i.d.R. auch ohne Parser geht

Nachteil:
−Schwierige Einbindung, da die Schnittstelle auf Konventionen basiert, nicht auf Interfaces die man maschinell importieren kann (Stichwort WSDL),
−Ändert man den Web Service, muss man das Programm dementsprechend komplizierter anpassen
−Es werden keine Variablen, sondern nur pure Daten übertragen, daraus folgt auch, dass Strukturen nur schwierig und Objekte gar nicht übertragen werden können
−Der Verwaltungsaufwand für das Eigenformat ist enorm – so muss man z.b. ein Erkennungsmerkmal schreiben, dass ein Array mit der Länge X folgt, ein Double von einem Integer unterscheiden u.s.w.
−Binärdaten können funktionieren, müssen aber nicht. Sie erfordern auf jeden Fall eine spezielle Behandlung, wenn sicht sogar eine Kodierung per Hand
−Eingabedaten müssen selbst z.b. auf Datentypen geprüft werden (falls aus versehen oder wegen Manipulation ein String anstelle eines Integers übermittelt wird)
−daraus folgt: Nicht zur Weitergabe geeignet

Vorteil:
−kleine Bandbreitennutzung
−Möglichkeit, zu komprimieren
−Je nach Aufwand teils schnelleres Parsen als ein SOAP−Request

Ein proprietärer Server

```
<?php
  mysql_connect("localhost");
  mysql_select_db("test");
  $sql=stripslashes($sql);
  $res=mysql_query("$sql") or die ("sqlerror in |$sql| is ".mysql_error());
  $numFields=mysql_num_fields($res);
  while ($row=mysql_fetch_row($res))
  {
    for ($i=0; $i<$numFields; $i++) echo $row[$i]."|";
    echo "\n";

  }

  mysql_close();
?>
```

Ein proprietärer Client

Benötigt die Komponenten Memo1, IdHTTP1 (Indy HTTP−Client) und ListView1.

```
procedure TForm1.Button1Click(Sender: TObject);
var
        statement, remainer : String;    // statement: sql−statement to mask,
              i : Integer;               // remainer: remaining string to parse
        newItem : TlistItem;             // i: index for rows
        newCol : TListColumn;
        countCols : Integer;
begin
```

```
  statement:=Edit1.Text; // prepare statement to be urlencoded (delphi command
slipped off my mind)
  statement:=StringReplace(statement,' ', '+', [rfReplaceAll]);
  statement:=StringReplace(statement,'=', '%3d', [rfReplaceAll]);
  statement:=StringReplace(statement,'''', '%27', [rfReplaceAll]);

Memo1.Text:=IdHTTP1.Get('http://127.0.0.1/jax/returnTable.php?sql='+statement);
  if pos('sqlerror',Memo1.Text)<>0 then ShowMessage(Memo1.Text);
  ListView1.Items.Clear;
  for i:=0 to Memo1.Lines.Count do
  begin
    newItem:=ListView1.Items.Add;
    newItem.Caption:=copy(Memo1.Lines[i],0,pos ('|', Memo1.Lines[i])-1);
    remainer:=copy(Memo1.Lines[i],pos ('|', Memo1.Lines[i])+1,9999);
    countCols:=1;

    while pos ('|', remainer)<>0 do
    begin
      inc(countCols);
      if countCols>ListView1.Columns.Count then
      newCol:=ListView1.Columns.Add;
      newItem.SubItems.Add(copy(remainer,0,pos ('|', remainer)-1));
      remainer:=copy(remainer,pos ('|', remainer)+1,9999);
    end;
  end;
end;
```

Um solch einen Web Service auf Performance zu testen, würde ich JMeter benutzen (jakarta.apache.org/jmeter). Jmeter ist ein Tool, um die Perfomance von Inhalten zu testen, die über eine HTTP−Verbindung ausgeliefert werden können (grob und ungenau gesagt ein Benchmark für Internetseiten).

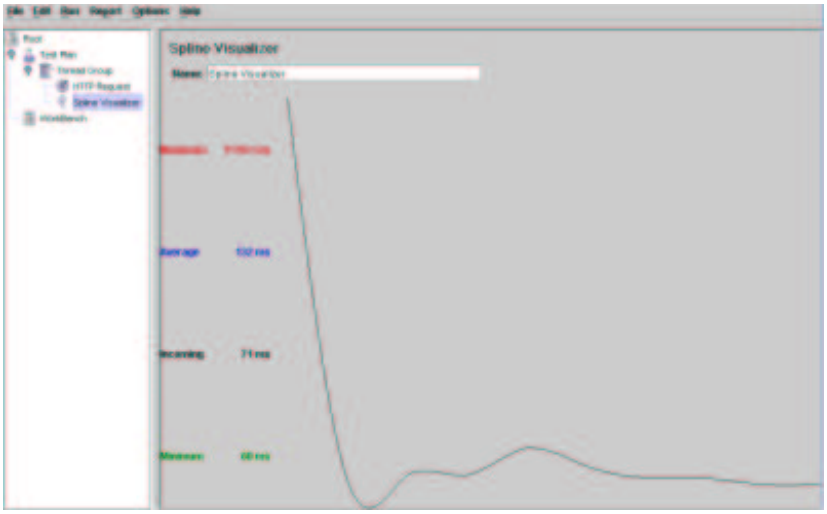


Abb. 1: JMeter bei einem 1−Thread−Test eines Services, der ein Double als Parameter bekommt und diesen wieder zurück gibt. Die Durchschnittsantwortzeit ist hier 130 ms.

−Klicken Sie dazu rechts auf Test Plan und fügen Sie eine Thread Group hinzu.
−Ändern Sie ggf. die Anzahl gewünschter Threads
−Dann klicken Sie rechts auf die Thread Group und fügen einen generativen HTTP−Request hinzu.
−Geben Sie Ihre lokale IP an (localhost oder 127.0.0.1) und als Path der URL−Pfad zu Ihrem PHP−Script, mit dem Namen des Scriptes und als Parameter z.B. `/jax/returnTable.php?sql=select+*+from+<tabellenname>.`
Ihr Aufruf muss leider URL−kodiert sein (Abstände als + u.s.w.).
−Klicken Sie nochmals rechts auf die Thread Group und fügen Sie einen Listener/Spline Visualizer hinzu
−Klicken Sie dann in Menü auf Run/Start und nach Ihrem Testzeitraum auf Run/Stop
−Klicken Sie auf den Spline Visualizer um Ihr Ergebnis zu sehen.

XML−RPC

Was ist XML−RPC

XML−RPC http://www.xmlrpc.org/ wie SOAP ein Standard zum Aufrufen von Funktionen über ein XML−Format und dem HTTP−Protokoll.
Wir benutzen die
Delphi−Implementation von Codepunk, http://www.codepunk.com/ und die
CBuilder−Implementation von mir, http://www.usegroup.de/software/xmlrpc/
PHP−Implementation von useful inc. http://xmlrpc.usefulinc.com/php.html#download

WSDLs sind für XML−RPC−Services technisch nicht möglich. In der Spezifikation findet sich eine Methode, die z.b. von einem Server alle zur Verfügung gestellten Services anzeigt, aber darauf kann man derzeit kein Interface generieren. Es gibt auch die Möglichkeit, Datenformate zu erzwingen, wie z.B. String.

Ein PHP−XML−RPC Server

Wir erstellen uns nun einen kleinen XML−RPC−Server in PHP

```
<?php
include("xmlrpc.inc");
include("xmlrpcs.inc");

function jax() {
      return new xmlrpcresp(new xmlrpcval("hallo, jax"));
}


$s=new xmlrpc_server( array("jax.test" =>array("function" => "jax")));
?>
```

Ein Delphi−XML−RPC Client

```
procedure TForm1.Button1Click(Sender: TObject);
var
  caller: TCaller;
  func : TFunction;
  rst  : TResult;

begin
  func := TFunction.Create;
  caller := TCaller.Create;
  caller.HostName := '192.168.0.103';
  caller.EndPoint := '/test/xmlrpc/simpleserver.php';

  func.ObjectMethod := 'jax.test';

  if rst.IsError then
    showmessage('Error:' + rst.GetErrorString())
  else
    ShowMessage(rst.GetString());

  func.Free;
  rst.Free;
  caller.Free;
end;
```

Ein CBuilder−XML−RPC Client

Installieren Sie den XML−RPC−Client von www.usegroup.de/software/xmlrpc/ und beginnen Sie ein neues VCL−Projekt, auf dem Sie einen Button und einen XML−RPC−Client (Im Tab „Web Services“) fallen lassen. Wählen Sie als Host http://localhost/,als MethodName in diesem Falle jax.test und in die „URL“ den Pfad und Dateinamen zu Ihrem PHP−Script ein. Fügen Sie nun als Onclick−Ereignis des Buttons folgenden Code ein:

```
 TResult *res=XMLRPCClient1->Execute(new TFunction());
if (res->IsError()) ShowMessage("Error:" + res->GetErrorString());
else ShowMessage(res->GetString());
```

Die Delphi−Implementation benötigt zwingend Indy 9, das Sie unter www.nevrona.com/indy/ erhalten. Bitte beachten Sie, dass ein installiertes Indy 9 selbst mit Delphi Service Pack 2 unter Umständen schwerwiegende Probleme bei den mitgelieferten Indy−Komponenten des CBuilder 6 auslösen kann.

Sie sehen, dass sich die Delphi− und CBuilder Implementation sehr ähneln – vor allem deshalb, weil 99% des Quelltextes der Delphi−Implementation 1 zu 1 in die CBuilder−Komponente übernommen wurden.

SOAP

Gegenüber den wenigen Nachteilen von SOAP (Simple Object Access Protocol), zum Beispiel dem, dass es zustandslos ist (bei jeder Anfrage muss neu authentifiziert werden. Siehe dazu: Managing Sessions with Delphi 6 Web Services (http://community.borland.com/article/0,1410,27575,00.html), treten viele Vorteile. Das Fehlen von Transactions, Security im Sinne von der sicheren Durchführung auch im Fehlerfall, Concurrency, Load−Balancing und Verfügbarkeit zähle ich nicht zu Nachteilen, da diesen Einschränkungen zum großen Teil auch Corba unterliegt.

Die Vorteile sind z.B.:
Durchlässigkeit durch Firewalls, vorhandene TCP/IP−Infrastruktur (bspw. Load−Balancer), kein Application Server nötig (übernimmt der Web Server), einheitliche Interface−Definition WSDL, ähnlich wie IDLs, zur maschinellen Interface−Generierung, Verfügbarkeit von XML−Parsern und HTTP−Clients für jede Plattform und viele Programmiersprachen. Somit sind SOAP−Server und Clients leicht zu Programmieren und Deployen.

Wir benutzen die
PHP−Implementation von Dietrich Ayala http://dietrich.ganx4.com/soapx4/ und die
Delphi/CBuilder Implementation von Borland

PHP’s Probleme mit SOAP

Zwei Dinge, die ich zur SOAP−PHP−Implementation von Ayala erwähnen möchte:

1) Sie hat ein Sicherheitsproblem; Sie nimmt ein Funktionsaufruf entgegennimmt und prüft dann mit der is_function bzw. eval−Funktion, ob diese Funktion vorhanden ist. Sie könnte deshalb anfällig sein gegenüber nicht vom Programmierer zur Verfügung gestelten Funktionen, z.b. der unlink−Funktion.

2) Sie hat keinen WSDL−Export, der das Benutzen aus Borland−Produkten heraus erst einfach macht. Die einzige Möglichkeit, überhaupt an Informationen zu kommen, welche Funktionen zur Verfügung stehen, besteht mangels Mißbrauchsmöglichkeiten des Interpreters in der PHP−Quelldatei, die man parsen muss.
   Ich habe einen solchen Parser geschrieben, der allerdings nur sehr simple PHP−Dateien akzeptiert. Es gibt ein Problem bei solchen Parsern, dass man aus PHP−Quelldateien nicht notwendigerweise erkennen kann, welcher Datentyp zurück geliefert wird. Erstens kann man den im Source nicht erkennen und zweitens kann der sich zur Laufzeit sogar ändern., z.B. in:

```
<?php
  function hallo($param)
  {
    if ($param==1) return 1;
    else return “string“;
  }
?>
```

Das muss umgangen werden, indem man explizit sagt, welcher Typ akzeptiert wird, in etwa

```
<?php
  function hallo(/* wsdlInt */ $param)
  {
    if ($param==1) return /* wsdlInt */1;
    else return /* wsdlInt */1;
  }
?>
```

Da es dafür keine Schlüsselwörter gibt, die man einsetzen kann, setze ich Kommentare ein (Casts könnte man für Rückgabewerte parsen, jedoch nicht für Parameter).

Ein CBuilder−SOAP−Server

Im C++Builder 6 wählen Sie
−Datei/Neu/Weitere/Web Services/SOAP Server Anwendung, dann CGI−Einzelanwendung, Interface erzeugen, vergeben Sie einen Service−Namen und wählen Sie Beispiel−Methoden generieren.
−Sie können dann aus der entsprechenden Datei echoEnum(SampleEnum eValue), echoDoubleArray(const TDoubleArray daValue), echoStruct(const TSampleStruct* pStruct) und die benötigten Strukturen und Arrays aus der Deklaration, Implementation und Header−Datei entfernen.
−Wählen Sie in Projekt/Optionen/Verzeichnisse das CGI−bin−Verzeichnis Ihres Apache als entgültige Ausgabe, und Speichen Sie das Projekt unter einem beliebigen Namen.
−Unter der Adresse http://localhost/cgi−bin/<derName>.exe können Sie dann eine Oberfläche ansprechen, die auch Links zur entstandenen WSDL enthält.

Performance

C++−CGI−SOAP, Windows	
1 Thread	0,18s (100%)
5 Anfrage−Threads	0,79s (100%)
bei 20 Anfrage−Threads	3,20s (100%)

Ein PHP−Soap−Server

Programmierung des Soap−Servers in PHP:

```
<?
require_once('class.soap_client.php');
require_once('class.soap_server.php');

      $server = new soap_server;
      $server->service($HTTP_RAW_POST_DATA);


function echoDouble(/*!wsdlDouble*/ $res)
  {

      return /*!wsdlDouble*/ $res;
  }
?>
```

Passen Sie dann den WSDL−Generator an,
Ändern Sie dazu die Zeilen

```
  $location="http://localhost/jax.php";
  // the location (port) of your web service
  $filename="jax.php";
  // the filename to parse
```

entsprechend Ihren Bedürfnissen.

Geben Sie für `$filename` kein Verzeichnis an, muss der WSDL−Generator im selben Verzeichnis liegen wie der SOAP−Server. Sprechen Sie die URL des Generators an, erhalten Sie die WSDL, die sie mit Datei/Speichern unter speichern können. Sprechen Sie die URL mit dem Parameter [IhreURL]?debug=true an, erhalten Sie ggf. Fehlermeldungen, warum die WSDL nicht erzeugt werden konnte.

Performance

PHP−SOAP, Windows
Dieser Soap−Server liefert auf meinem Athlon 1200 mit 256MB RAM, Windows Apache 1.3.26 und PHP 4.0.5 bei

1 Thread	0,14s (−18%)
5 Anfrage−Threads	0,71s (−11%)
bei 20 Anfrage−Threads	2,90s (−9,2%)

PHP unter Linux

Die bisherigen Geschwindigkeitsmessungen haben nur mit der Wirklichkeit so viel zu tun wie ein Schwein mit einer Mondrakete. Windows ist kein Serverbetriebssystem, alle mir bekannten Server, ausgenommen denen einer größeren amerikanischen Softwareschmiede, laufen auf Linux. Das allein wäre für die Statistik soweit noch nicht entscheidend, wenn nicht PHP unter Linux als Modul und unter Windows nur als CGI implementiert wäre. Da es das aber ist, ist die Linux−PHP−Performance eine wichtige Sache.

> Ich glaube nur an Statistiken, die ich selbst gefälscht habe.
>
> *Sir Winston Churchill (1874 - 1965), britischer Kriegsberichterstatter und Premierminister, 1953 Nobelpreis für Literatur*

Performance

PHP−SOAP, Linux bei 1 Anfrage−Thread Antwortzeiten von	0,059s (−66%)

Umstieg von CGI in DSO

Die Apache Shared Modules (DSOs) sind (besonders auf Plattformen, die nur langsam Prozesse erzeugen können wie Windows) teils erheblich schneller als CGIs. Sie werden nicht jedesmal neu erzeugt, sondern stehen zur gesamten Serverlaufzeit geladen zur Verfügung. Beachten Sie bitte, dass DSOs unter Linux statisch gegen den Apache gelinkt werden müssen. Ein Borland−spezifischer Patch muss dazu in Apache einkompiliert werden. (Siehe auch http://www.thedelphimagazine.com/samples/1222/1222.htm).

Um Ihr CGI in ein DSO umzuwandeln, gehen Sie wie folgt vor (da man den Server anhalten muss, um das Modul zu ersetzen, empfiehlt sich eine Entwicklung zunächst als CGI oder Web−App−Debugger−Anwendung).

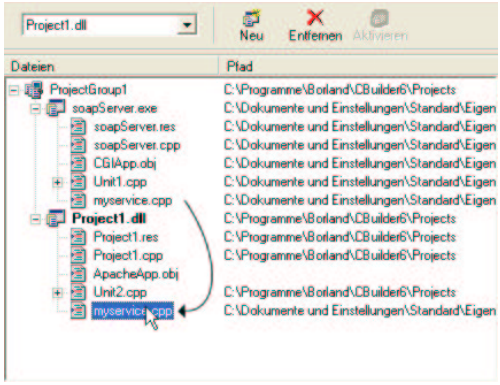


Abb. 2: Umwandeln von einem CGI in ein DSO

Öffnen Sie die Projektverwaltung bei geöffneter SOAP−CGI−Anwendung und klicken Sie auf „Neu“. Wählen Sie Webservices/Soap−Server−Anwendung/Apache Shared Module und erzeugen Sie kein Interface. Ziehen Sie die Quelldatei mit dem Namen Ihres Services aus dem alten CGI in das neue DSO und aktivieren Sie das neue DSO. Wählen Sie in Projekt/Optionen/Verzeichnisse das modules−Unterverzeichnis Ihres Apache−Servers für die endgültige Ausgabe. Stoppen Sie ggf. Ihren Server; drücken Sie STRG+F9 und ändern Sie Ihre httpd.conf.

Erstens müssen Sie bei den LoadModules das eigene Modul mit−laden und zweitens müssen Sie sagen, unter welcher URL es erreichbar sein soll. Im Bereich der Loadmosule müssen Sie also hinzufügen
```
LoadModule _Project1_module modules/Project1.dll
```

Bitte beachten Sie: Bei unter Delphi und Kylix erstellten Modulen entfällt der Unterstrich vor dem Modulnamen, die korrekte Zeile für diese Programmiersprachen lautet also:
```
LoadModule Project1_module modules/Project1.dll
```
Und bei ggf. schon vorhandenen Location−Einträgen

```
<Location /P0>
  SetHandler project1−handler
</Location>
```

Starten Sie dann Ihren Apache−Server und sprechen Sie die Location http://localhost/P0/ an.
Anmerkung: Sie können die Auslastung Ihres SOAP−Server zusätzlich überprüfen, indem sie eine Funktion den insgesamt noch zur Verfügung stehenden Speicher zurück liefern lassen. Eine Funktion wie
```
int TinterfaceImpl::echoLoad()
{
  _MEMORYSTATUS mem;
  GlobalMemoryStatus(&mem);
  return mem.dwAvailVirtual;
}
```
sollte genügen.

Performance

C++−DSO−SOAP, Windows	
1 Thread	0,054s (−69%)
5 Anfrage−Threads Antwortzeiten von	0,19s (−76%)
bei 20 Anfrage−Threads durchschnittliche Antwortzeiten von	0,74s (−77%)

In Kylix tritt dieses Verhalten nicht auf. Als betroffen ist derzeit nur Delphi 6 und Delphi 6 Update 2 bekannt.

Um genauer herauszufinden, ob WSDLs bei der Laufzeit des Programms benötigt werden, kann man in den PHP−WSDL−Generator folgende Zeilen einfügen:

```
$fi = fopen ("log.txt", "a+");
fwrite($fi, "\nCalled @".time());
fclose($fi);
```

Sie werden sehen, dass die WSDL dreimal beim ersten Aufruf einer Funktion über das RIO und zweimal bei jedem weiteren Aufruf gelesen wird. Der Grund dafür erschien mit fraglich, und entsprechendes habe ich auch in die Gruppe news://newsgroups.borland.com/borland.public.delphi.webservices.wsdl geäußert (why's the WSDL read at runtime von 1.5.2002). Eine Antwort des Entwicklers der WSDL−Komponenten:
„I took at quick peek at the code and [sadly :(] the culprit was the result of an accidental checkin I made."
Man könne in der Datei `WSDLNode.pas` in der `function ActiveWSDL` die auskommentierten Zeilen

```
{ if not WSDL.Active then begin }
WSDL.StreamLoader.UserName := Name;
WSDL.StreamLoader.Password := Password;
WSDL.StreamLoader.Proxy := Proxy;
WSDL.Load(WSDL.FileName);
{ end }
```

Wieder aktivieren. Löschen sie danach die `WSDLNode.dcu` im Delphi\Lib−Verzeichnis und nehmen das entsprechende Source−Verzeichnis (Delphi\Source\Soap) in den Suchpfad für Units auf (Projekt/Optionen/Verzeichnisse/Suchpfad). Wenn sie absichtlich einen Fehler in die Datei schreiben können Sie an der Fehlermeldung des Compilers erkennen, ob diese Datei tatsächlich benutzt wird.

Wenn Sie jetzt die fehlerfreie, geänderte Version verwenden, wird ein Abfragethread der die BCB6−DSO−Version 100x abfragt [auf einem Athlon 1200], anstelle 0,045s, wenn er die WSDL lokal hält, nur noch 0,032 (−28%) benötigen, und wenn er die WSDL über HTTP holt, wird er anstelle 0,105s ebenfalls nur noch 0,033s benötigen (−69%).

Abschließende Worte

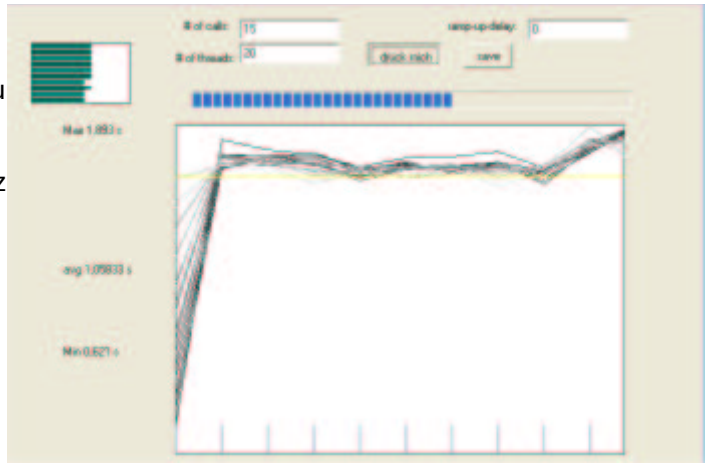
Die Perfomance von PHP ist also nicht notwendigerweise schlecht, genauso wenig wie die Performance von C++Builder notwendigerweise gut ist. Sollten Sie Ihre Performance jedoch im Web−Server und Web−Service−Bereich verdoppeln wollen, kommen Sie um Apache Shared Modules nicht herum.

Anhang A: serMeter

Unter Apache JMeter (jakarta.apache.org/jmeter) ist es zwar theoretisch möglich, Web Services auf Performance zu testen. Um das zu erreichen, müsste man eine Datei mit einem korrekten SOAP−Envelope anlegen und diese dann per HTTP Request an den entsprechenden Web Service übergeben. Das ist jedoch kompliziert und entbehrt Applikationslogik, so müsste man sich Fehlermeldungen im results view zusammenklamüsern und selbst Fehlermeldungen wie Server überlastet gingen per Default als korrektes Ergebnis in die Statistik.

Das sind genügend Gründe, sich einen eigenen Benchmakr für Web Services zu schreiben. Für ein JMeter−Modul habe ich nicht genügend Erfahrung, deshalb ist mein Benchmark, serMeter, in Kylix geschrieben.

SerMeter ist standardmäßig an Babelfish gebunden. Das können Sie im Quelltext leicht ändern (s.U.), leider ist es mir nicht gelungen, eine Schnittstelle zu beliebigen Web Services zu erzeugen.



Geben Sie in # of Calls ein, wie oft einen Instanz den Web Service aufrufen soll und in # of Threads wie viele Instanzen Sie möchten. Nach dem Starten jedes Threads wird Ramp−Up−Delay−Millisekunden gewartet, bevor der nächste Thread gestartet wird.

Sie erhalten den Fortschritt gemittelt in der Progressbar, und pro Instanz im Abschnitt links oben. Es wird maximal der Fortschritt von 60 Threads richtig anzeigt.

Die Antwortzeiten erhalten Sie grafisch im Hauptabschnitt, die gelbe Linie zeigt den Durchschnitt. Die einzelnen Threads nehmen zur besseren Unterscheidung Grauwerte an. Am unteren Rand werden zur besseren Übersicht die Positionen eingetragen, nach denen jeweils eine weitere Anfrage gestartet wurde (genauer: gestartet hätte werden sollen, wenn man von einem Thread ohne Ramp−Up ausgeht, der auch in der Liniendarstellung nicht berücksichtigt wird. Der Anzeigebereich wird einfach in # of calls Bereiche eingeteilt, insbesondere Ramp−Up müsste Auswirkungen auf die Einteilung haben und die Einteilung müsste eigentlich pro Thread erfolgen).

Der Linke Rand des Anzeigebereichs entspricht immer dem ersten Call, der Rechte dem letzten, der untere der schnellsten, der obere der langsamsten Antwort. Zoomen ist nicht möglich. Je mehr calls abgesetzt wurden, desto geringer ist der Abstand der einzelnen Calls im Fenster.

Save speichert die entstandene Grafik als BMP−Datei und übernimmt die Maximaldauer, den Durchschnitt und die Minimalantwortzeit in die Grafik.

Anpassung von serMeter auf andere Web Services
−Öffnen Sie das serMeter−Projekt in Delphi oder Kylix.
−Wählen Sie Datei/Neu/Andere/Web Services/WSDL Importer und erstellen Sie ein Interface zu Ihrer WSDL. Öffnen Sie die Datei smokeThreadUnit.pas und tragen Sie diese Unit im interface−Teil in Ihre uses−Klausel ein.
−Ändern Sie den Typ und, falls gewünscht, den Variablennamen von „babel“ in Ihren Porttypen.
−Ändern Sie in der Prozedur Execute `httprio.WSDLLocation, Service und Port` Ihrem Service entsprechend.
−Ändern Sie die Zeile `babel:=HTTPRIO as BabelFishPortType;` nach Ihrem Variablennamen und Port−Typen und die Zeile `babel.BabelFish('en_de','smoke test');` entsprechend der Funktion Ihres Services, die Sie benchmarken möchten.

Anhang B: Ungeziefer in der Seife (Bugs in SOAP)

WSDLs @ Runtime
Siehe Abschnitt Client−Side−Performance.

Inofficial bug fixes
Es gibt eine Liste von SOAP bug fixes, die zwar schon in die Borland−Entwicklerversionen eingecheckt sind, jedoch für die Öffentlichkeit noch nicht zur Verfügung stehen. Von diesen Bug−Fixes ist nicht einmal klar, ob sie in Form eines Update 3 oder Delphi 7 auf den Markt kommen werden, sie sind aber unter http://community.borland.com/article/0,1410,28514,00.html dokumentiert, sodass sie Kunden bei Bedarf an Ihrer Delphi−Version selbst patchen können.

WSDLs in Update 2
Seit dem Update 2 von Delphi wird die WSDL übrigens nicht mehr über MSXML geladen, sondern direkt über WinINet, was zu Problemen mit Windows 2000 führen kann.

Servererstellung
Für Kylix würde ich zur Erstellung eines Soap−Servers dringend das SOAP−Server Skeleton „soapex“ empfehlen (http://www.borland.com/kylix/webreg/k2/k2_registeredusers.html), für Delphi den Invokamatic Wizard „Invokable“ (http://www.borland.com/delphi/webreg/d6/d6_registeredusers.html) oder das Update 2. Das sind zwar im Grunde keine Bugs, aber bei der Servererstellung sehr wichtige Features. In CBuilder 6 ist der Invokamatic von Delphi schon eingebaut.

Kylix Multithreading
In Kylix verhinderte ein Bug (in meinem Programm oder in Kylix) das Multithreading mit SOAP−Komponenten. Ein Vorschlag, den ich nicht ausprobiert habe, lautet, die Datei `SOAPHTTPTrans` zu patchen, und in dieser anstelle Indy Synapse zu verwenden. (http://www.ararat.cz/synapse). Sehen Sie bitte auf die nächste Seite für den Code.

```
  //Hinzufügen
  procedure SetupSyna(SynaHttp: THTTPSend);
  var
    ContentHeader, ActionHeader: string;
    Protocol, Host, path, Document, Port, Bookmark: string;
  begin
    if (soNoValueForEmptySOAPAction in FInvokeOptions) and (SoapAction = '')
then
      ActionHeader := SHTTPSoapAction + ':'
    else if SoapAction = '""' then ActionHeader := SHTTPSoapAction + ': ""'
    else ActionHeader := SHTTPSoapAction + ': ' + '"' + FSoapAction + '"';
    SynaHttp.Headers.Add(ActionHeader);
    SynaHTTP.MimeType := 'text/xml';   { do not localize }
    SynaHttp.Protocol := '1.0' ;
  end;

{$IFDEF USE_SYNA}
  procedure PostData(Response: TStream);
  var
    SynaHTTP: THTTPSend;
    WireData: string;
  begin
    SynaHTTP := THTTPSend.Create;
    try
      SetupSyna(SynaHttp);
      WireData := UTF8Encode(DataMsg);
      SynaHTTP.Document.Write(Pointer(WireData)^, Length(WireData));
      try
      SynaHTTP.HTTPMethod('POST', FURL);
      Response.CopyFrom(SynaHTTP.Document, 0);
      finally
      end;
    finally
      FreeAndNil(SynaHTTP);
    end;
  end;
{$ENDIF}
//Und folgende Änderung
{$IFDEF USE_INDY}
    InitURL(FURL);
{$ENDIF}
```