A SOAP

A SOAP olyasmi, aminek akkor is nagy hasznát vehetjük, ha valójában nem is érdeklődünk a háromrétegű webes alkalmazások iránt.

A két előző Kovácsműhelyben egy egyszerű háromrétegű webalkalmazást mutattam be adatbázis, webkiszolgáló és a mod_perl Mason sablonrendszer használatával. Megvizsgáltuk a háromrétegű alkalmazások előnyeit és hátrányait, és összehasonlítottuk a kétrétegű alkalmazásokkal.
Azonban – ahogy arra a múlt hónapban is rámutattam – a mi kis háromrétegű alkalmazásunk még nincsen teljesen készen, ezért nem lehet megfelelően bemutatni a működését. Emiatt a Perl köztes objektumrétege ugyanazon a számítógépen található, mint a mod_perlre épülő Mason sablonrendszer segítségével készített HTML-elemek. Annak ellenére kétrétegű alkalmazásnak tekinthetjük, hogy a rétegeket objektumközpontú elvonatkoztatási réteg választja el egymástól.
Ahhoz, hogy külön számítógépre tegyük a Mason-elemeket és a Perl-objektumokat, meg kell oldanunk a hálózaton keresztüli objektumhívást. A következő sornak például attól függetlenül működnie kell Perlben, hogy a `$object` ugyanazon az Apache kiszolgálón van-e, vagy valahol máshol az Interneten:

```
$object->method($arg1, $arg2);
```

Az elosztott objektum módszer és a távoli eljáráshívás jó néhány éve elérhető a különböző felületeken. A legtöbb esetben ez a módszer egy független nyelvre vagy felületre korlátozódik. A DCOM modell (Distributed Component Object Model) lehetővé teszi az objektumok kapcsolattartását, de csak Windows alatt. A Java távoli eljáráshívása (Remote Method Invocation – RMI) csak Java-objektumokkal hajlandó együttműködni. A CORBA kivételesen lehetővé teszi a különböző nyelvek és felületek közötti kapcsolattartást, de nagyon összetett nyelv, és jelenleg a legtöbb programozó nem ismeri eléggé.
Ezekre az egyedi és összetett protokollokra válaszul az internetes közösség létrehozta saját egyszerű objektumelérési protokollját, a SOAP-ot (Simple Object Access Protocol). Segítségével pofonegyszerű elosztott alkalmazásokat létrehozni. A SOAP két legnagyobb támogatója a weblog hírlevélből ismert *Dave Winer* és a Microsoft. Bár e cég nem arról híres, hogy támogatja a nyílt szabványokat és a felületfüggetlen protokollokat. A linuxos közösség véleménye szerint a Microsoft azért támogatja nyilvánosan a SOAP-ot, mert a .Net programjának is ez az egyik alapköve.

A SOAP története és alapgondolata

A SOAP alapgondolata az, hogy az Interneten található bármilyen két számítógép a HTTP protokoll használatával képes kapcsolatba lépni egymással. Jelenleg a SOAP majdnem mindegyik magas szintű protokollon továbbítható, mint például az SMTP és a POP3, de a HTTP a leggyakoribb. Képes adatot továbbítani az XML használatával (ez a nyelv lehetővé teszi, hogy saját tagokat és dokumentumszabványokat hozzunk létre). A kiszolgáló a beérkező XML parancsokat lefordítja objektum eljáráshívásokra, majd az objektum válaszát befordítja egy XML dokumentumba, és HTTP válaszként továbbítja. Mivel a HTTP és az XML is olyan nyílt szabvány, amit a World Wide Web Consortium hozott létre, ezért minden gond nélkül megvalósítható és alkalmazható minden felületen.
A SOAP ősét XML-RPC-ként ismerjük. Ez egyszerűen lehetővé tette a távoli eljáráshívást (Remote Procedure Call) XML formátumú adatokkal HTTP protokollon keresztül. Az XML-RPC azonban nem kezeli a fejlett adatszerkezeteket, ezért a W3C létrehozta a SOAP-ot. Számos nyelv és felület továbbra is támogatja az XML-RPC-t, ennek következtében az néhány esetben némi fejfájást is okozhat. Röviden összefoglalva a SOAP annak köszönheti a kiemelt figyelmet, hogy a programkönyvtárak, a függvények használata és a hibakeresése fejlettebb, mint XML-RPC-é. A SOAP szélesebb körben alkalmazható, mint az XML-RPC, ez azt is jelenti, hogy szabadon választható a felület, a nyelv, vagy akár a protokoll.
A SOAP, ahogy a nevéből is sejthető, objektumokkal, és nem egyszerű eljáráshívásokkal dolgozik. A SOAP ügyfél a kiszolgálón található objektum bármelyik tagfüggvényét meg tudja hívni. A tagfüggvényt az XML dokumentum törzsében adhatjuk meg, az objektumot magát pedig a HTTP fejléc „SOAPAction” sorában. Természetesen meg kell határozni a számítógép nevét és a kaput is, amelyre a SOAP-kérést irányítjuk.
A kiszolgálót (a gép nevét és a kapu címét), ahová a kérést küldjük, SOAP proxynak hívjuk. Az elnevezés érthető, ha arra gondolunk, hogy a HTTP kiszolgáló egyszerűen csak továbbítja az elvégzendő parancsokat, ő maga nem végez feladatot. Ne tévesszük össze a SOAP proxyt és a HTTP proxyt. A HTTP proxy továbbítja a HTTP ügyfél kéréseit a HTTP kiszolgálónak, és gyakran biztonsági ellenőrzéseket hajt végre, ezenkívül gyorstárként is működik. Ezzel ellentétben a SOAP proxy az ügyfél és az objektum között zajló üzeneteket továbbítja.
A SOAP kiszolgálón található objektumra gyakran *végpont* névvel is hivatkozunk, ezt a fejléc „SOAPAction” sorában kell megadnunk. A végpont neve tulajdonképpen bármilyen szöveges változó lehet, beleértve a hierarchikus elválasztó jeleket is (mint például :: és /). A végpont írásmódja általában követi azt a programnyelvet, amiben a SOAP proxy is készült. Perlben a végpont lehet modjuk Foo/Bar, vagy hasonló, ami a Foo/Bar.pm-ben található Foo::Bar objektumra utal.

A SOAP belső felépítése

Most nézzünk egy egyszerű SOAP „beszélgetést”. Példánkban a SOAP a HTTP protokollon ül, mivel ez a leggyakoribb. Más protokollok használatánál némi eltérést tapasztalhatunk.
A HTTP nem állapotfüggő, ami azt jelenti, hogy két számítógép között létrejött kapcsolat egyetlen kérésből (az ügyféltől a kiszolgáló felé) és egyetlen válaszból (a kiszolgálótól vissza az ügyfélhez) áll. A kérés és a válasz is két részre bontható: a fejlécre és a törzsre. Természetesen az ügyfél és a kiszolgáló tetszőleges, más fejléceket is hozzátehet, ajtót nyitva ezzel számos különleges kapcsolattartó protokoll előtt.
A SOAP kérés és válasz törzse XML formátumú lesz. Ha sohasem dolgoztál még XML-lel, akkor sem kell megijedni. Bár ez egy nagy és furmányos témakör, ahhoz, hogy a SOAP-ot használd, nem kell túl

1. lista Caps.pm, a Perl modul

A Caps.pm szolgáltatja a SOAP végpontot.

```
package Text::Caps;

use strict;
use diagnostics;
#A végleges programban iktassuk ki.

# Egyetlen változót várunk. A kapott változót
# nagybetűssé alakítjuk
# a beépített uc függvénnyel, majd visszaadjuk.

sub capitalize
{
   my $self = shift;
   my $word = shift;

   return uc ($word);
}

# A capitalize_array egy listát vár.
# Az eljárás a kapott listához igazodó
# listát ad vissza, az elemeket nagybetűssé
# alakítva.

sub capitalize_array
{
   my $self = shift;
   my @words = @_;

   return [map {uc $_} @words];

}
```

sokat tudnod az XML-ről. Minden SOAP üzenet – akár kérés, akár válasz – tartalmaz egy kiegészítő SOAP fejlécet és egy kötelező SOAP törzset a SOAP borítékba csomagolva. A boríték azonosítja a SOAP csomag tartalmát, és meghatározza a névmezőt, amit a későbbiekben az üzenet további részéhez használhatunk fel. A fejlécek leírják a törzsben található adatokat, a törzs pedig tartalmazza a tagfüggvényhívást, vagy annak eredményét.

Hogy a SOAP segítségével egy távoli objektumot használjunk, először egy http kapcsolatot kell létrehoznunk a kívánt URL felé, és a „SOAPAction” sorban meg kell adjuk a használni kívánt objektum pontos nevét. Egy XML dokumentumot küldünk, benne a SOAP borítékkal, mely tartalmazza a SOAP fejlécet és a törzset. A törzsben megadjuk a meghívni kívánt tagfüggvényt és az összes további értéket. Az ügyfelet fel kell készíteni a SOAP kiszolgáló által visszaküldött adatszerkezetek kiértékelésére és további felhasználására.

A SOAP kiszolgáló hasonló műveleteket hajt végre, fogadja a SOAP kéréseket, elemzi azok tartalmát, és meghívja a megfelelő tagfüggvényt az átadott értékekkel együtt. Ezután visszaküldi a szükséges értékeket tartalmazó választ az ügyfélnek.

Most, hogy megismertük a SOAP szókincsét, a többségét el is felejthetjük. A SOAP elvonatkoztatási rétegekkel történő megvalósításánál lehetőségünk nyílik arra, hogy figyelmen kívül hagyjuk a HTTP kapcsolattartást és az XML kérés és válasz részleteit. Ha a programunk a SOAP segítségével kéri le a távoli objektumfüggvényt, a kérések és válaszok csomagolása a SOAP feladata.

2. lista Az egyszerű önálló SOAP kiszolgáló

Ez a program a 8080-as kapun várakozik a SOAP kérések fogadására, és átadja őket a megfelelő objektumnak.

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

use strict;
use diagnostics;
# A végleges programban iktassuk ki

use SOAP::Transport::HTTP;
# Az objektum fogadása a kiszolgáló számára

my $SERVER_PORT = 8080;
my $SERVER_NAME = 'localhost';

# A SOAP kiszolgáló objektum létrehozása
my $soap_server = SOAP::Transport::HTTP::Daemon
    -> new (LocalAddr => $SERVER_NAME,
           ➥LocalPort =>$SERVER_PORT)

# Mi az objektum alapkönyvtára?
# (Ne feledjük, az alapértelmezett Perl @INC
# útvonal le van tiltva.)

# Ne használjuk a /tmp útvonalat egy valódi
# kiszolgálónál!
    -> dispatch_to('/tmp/');

# Kiírjuk, melyik kapun várjuk a SOAP kéréseket.
print "SOAP server is waiting on port
$SERVER_PORT...\n";

# Most kezeljük a beérkező SOAP eljáráshívást,
# és visszaküldjük a megfelelő SOAP választ.
$soap_server->handle();
```

A kiszolgálóoldali objektum

Írtam néhány példaprogramot Perlben a *Paul Kulchenko* alkotta kitűnő SOAP::Lite modul segítségével. Ez adhat néhány ötletet a SOAP kiszolgálók és ügyfelek megírásához, valamint a webalkalmazásainkba építésük módozataihoz. Nevének dacára, a SOAP::Lite rengeteg szolgáltatást kínál számunkra, és remek eszköz lehet a Perl programok SOAP-pal történő bővítéséhez. Hasonló SOAP függvények és objektumok a jelentősebb programnyelveken is megtalálhatók, ne gondoljuk, hogy a SOAP csak a Perllel használható.

Mivel a SOAP az objektum számára proxyként működik, először létre kell hoznunk egy objektumot, mely elérhető a hálózaton. Az *1. lista* tartalmazza az egyszerű Text::Caps Perl modult, ez a modul két eléggé hasznavehetetlen eljárást tartalmaz:

- *capitalize* – a kapott szöveget nagybetűsre alakítva küldi vissza.
- *capitalize_array* – ez ugyanazt csinálja, mint a capitalize, de a kapott tömb összes elemével.

Jegyezzük meg, míg a SOAP minden kifejezést objektumalapú megnevezéssel ír le, ez a mintamodul inkább a hagyományos Perl írásmódot használja. Szóval, ha azt emlegetem, hogy a Text::Caps objektum capitalize eljárása, akkor valójában a Text::Caps::Capitalize eljárás meghívására gondolok.

3. lista SOAP-kérés

```
POST http://localhost:8080/
Content-Length: 509
Content-Type: text/xml
SOAPAction: "Text/Caps#capitalize"

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:xsi=
➥"http://www.w3.org/1999/XMLSchema-instance"
xmlns:SOAP-ENC=
➥"http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
SOAP-ENV:encodingStyle=
➥"http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV=
➥"http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<namesp3:capitalize xmlns:namesp3="Text/Caps">
<c-gensym19 xsi:type="xsd:string">abc</
➥c-gensym19>
</namesp3:capitalize>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Egy önálló SOAP kiszolgáló

A SOAP általában a HTTP-t használja, amely viszont a TCP/IP protokoll tetején „utazik”. Ez azt jelenti, hogy a Perl beépített, TCP/IP foglalatokat (socket) kezelő részének használatával akár egy egyszerű SOAP kiszolgálót is létrehozhatunk. Így a SOAP::Lite elvégzi helyettünk a piszkos munka nagy részét, nem nekünk kell létrehozni a foglalatot, vagy várakozni rá. Inkább, készítsünk egy SOAP::Transport::HTTP::Daemon-típusú objektumot, mely tudja, hogyan viselkedjen megfelelő típusú SOAP kiszolgálóként. Az egyszerű kiszolgáló forráskódja a *2. listában* látható.
A forráskód viszonylag egyszerű, ennek ellenére még a gyakorlott Perl programozók számára is furcsának tűnhet. Talán mert a SOAP::Lite-hoz kapcsolódó objektumok a sikeres működést többnyire úgy jelzik, hogy saját magukat adják meg visszatérési értékként. Ez teszi lehetővé számunkra, hogy egynél több eljárást is lekérhessünk egyetlen hívásban. Szóval, azt mondhatjuk, hogy

```
$object->method1()->method2();
```

az alábbi hagyományos írásmód helyett

```
$object->method1();
$object->method2();
```

A SOAP::Lite-ban mindkét formátumot használhatjuk, de az első változat gyakrabban fordul elő a leírásban. Amikor a SOAP::Transport::HTTP::Daemon „new” létrehozóját (constructor) hívjuk meg, két értéket adunk át neki: azt a gépnevet és kapucímet, ahol a démonnak figyelnie kell.
A kiszolgáló létrehozása után meg kell adnunk, hogy hol találja meg a keresett objektumokat. Erre biztonsági okokból van szükség, bár elsőre ez nem túl érthető. Általában a Perl a modulokat a @INC-ben keresi a könyvtárnevek sorrendjében. Amikor használni kívánunk egy modult, a Perl a @NIC minden elemét egymás után addig vizsgálja, amíg meg nem találja a keresett modult. Ha a keresés eredménytelen, hibaüzenetet kapunk.

4. lista SOAP-válasz XML-ben

```
HTTP/1.1 200 OK
Date: Sun, 07 Jan 2001 20:31:35 GMT
Server: libwww-perl-daemon/1.21
Content-Length: 523
Content-Type: text/xml
Client-Date: Sun, 07 Jan 2001 20:31:35 GMT
Client-Peer: 127.0.0.1:8080
SOAPServer: SOAP::Lite/Perl/0.44

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelopei xmlns:xsi=
➥"http://www.w3.org/1999/XMLSchema-instance"
xmlns:SOAP-ENC=
"http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
SOAP-ENV:encodingStyle=
➥"http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV=
➥"http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<namesp1:capitalizeResponse xmlns:namesp1=
➥"Text/Caps">
<s-gensym5 xsi:type="xsd:string">ABC</
➥s-gensym5>
</namesp1:capitalizeResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Bár a SOAP-modulok az egész világon megtalálhatók, jobb, ha óvatosak vagyunk, mielőtt elérhetővé teszünk egy modult. Megtörténhet, hogy egyes modulok bizalmas adatokat továbbítanak, vagy adatokat módosítanak egy relációs adatbázisban. Ezért győződjünk meg róla, hogy csak a működéshez szükséges modulok érhetők el a SOAP számára, így a SOAP::Lite teljesen figyelmen kívül hagyja az @INC-et a SOAP-kérések érkezésekor. Csak azok a modulok szerepeljenek a dispatch_to() hívásban, vagy a dispatch_to() nevű könyvtárban, melyeket elérhetővé kívánunk tenni a SOAP számára. Tulajdonképpen a dispatch_to() helyettesíti az @INC változót a SOAP számára. Ha egy modul egy olyan könyvtárban található, ami nem szerepel a dispatch_to()-ban, a SOAP számára láthatatlan lesz. Ez nem ugyanaz, mintha az @INC értékét módosítanánk. Megjegyzem, miközben én a példákban /tmp könyvtárat használom, rossz ötlet ezt a könyvtárat használni egy éles fejlesztés során, vagy működő rendszernél. Ha a /usr/lib/perl könyvtárból külön könyvtárba szeretnénk elhelyezni a SOAP-hoz tartozó Perl-modulokat, határozottan javaslom, hogy a központi fájlrendszeren tartsuk azokat, például a /usr/lib/soaplite könyvtárban.

Kiszolgálónk kipróbálása

Most, hogy az önálló SOAP kiszolgálónk üzemel, ki kell próbálnunk hogy valóban működik-e. Ehhez először létre kell hoznunk egy SOAP-kérést, el kell küldeni a kiszolgálónak, és meg kell vizsgálni a visszaérkező XML kódolású üzenetet. Szerencsére a SOAP::Lite tartalmazza a SOAPsh.pl nevű segédprogramot. Ennek segítségével tudunk készíteni és küldeni SOAP-kéréseket, és interaktív módon az eredményeket azonnal meg is jeleníti a képernyőn. Már csak a SOAPsh.pl kedvéért is megéri letölteni a SOAP::Lite-ot, még akkor is, ha más SOAP függvényekkel fogunk majd dolgozni.
Ha a SOAP kiszolgálónkat ugyanazon a számítógépen futtatjuk, mint

5. lista A hibakeresés eredménye

```
POST http://localhost:8080/
Content-Length: 628
Content-Type: text/xml
SOAPAction: "Text/Caps#capitalize_array"

<?xml version="1.0"encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:xsi=
➥"http://www.w3.org/1999/XMLSchema-instance"
xmlns:SOAP-ENC=
➥"http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
SOAP-ENV:encodingStyle=
➥"http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV=
➥"http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<namesp4:capitalize_array
➥xmlns:namesp4="Text/Caps">
<c-gensym24 xsi:type="xsd:string">
reuven
</c-gensym24>
<c-gensym26 xsi:type="xsd:string">
shira
</c-gensym26>
<c-gensym28 xsi:type="xsd:string">
atara
</c-gensym28>
</namesp4:capitalize_array>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

HTTP/1.1 200 OK
Date: Sun, 07 Jan 2001 20:41:03 GMT
Server: libwww-perl-daemon/1.21
Content-Length: 738
Content-Type: text/xml
Client-Date: Sun, 07 Jan 2001 20:41:03 GMT
Client-Peer: 127.0.0.1:8080
SOAPServer: SOAP::Lite/Perl/0.44

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:xsi="http://www.w3.org/1999/
XMLSchema-➥instance"
xmlns:SOAP-ENC=
➥"http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
SOAP-ENV:encodingStyle=
➥"http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV=
➥"http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<namesp2:capitalize_arrayResponse
➥xmlns:namesp2="Text/Caps">
<SOAP-ENV:Array xsi:type="SOAP-ENV:Array"
SOAP-ENC:arrayType="xsd:string[3]">
<s-gensym10 xsi:type="xsd:string">
REUVEN
</s-gensym10>
<s-gensym10 xsi:type="xsd:string">
SHIRA
</s-gensym10>
<s-gensym10 xsi:type="xsd:string">
ATARA
</s-gensym10>
</SOAP-ENV:Array>
</namesp2:capitalize_arrayResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

a SOAPsh.pl-t, és a 8080-as kaput használjuk, a következő kérést alkalmazzuk:

```
perl SOAPsh.pl http://localhost:8080/ Text/Caps
```

Megjegyzem, az SOAPsh.pl első paramétere a SOAP kiszolgáló címe, a második pedig a használni kívánt objektum. Sok nehezen lenyomozható hibától mentjük meg magunkat, ha a második paramétert az URL-ekben megszokott módon, tehát perjellel tagoljuk. A Text::Caps helyett inkább a Text/Caps formát használjuk. Ha a SOAPsh.pl sikeres, a következőt láthatjuk:

```
Usage: method[(parameters)]
>
```

A > jel azt jelenti, hogy kapcsolatba léptünk az objektummal, és bármelyik tagfüggvényét meghívhatjuk. Egy szó nagybetűssé alakításához egyszerűen csak be kell gépelni:

```
> capitalize(´abc´)
```

6. lista soap-client.pl

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

use strict;
use diagnostics;  # A végleges programban
                  iktassuk ki
use SOAP::Lite;


# Adjuk meg a kapott változót az átalakításhoz
my $result = SOAP::Lite -> uri('Text/Caps') ->
    proxy('http://localhost:8080') ->
    capitalize($ARGV[0]) -> result();

# Írassuk ki az eredményt
print "Result = '$result'\n";
```

Mivel a SOAP kiszolgáló és az ügyfél azonos számítógépen van, ezért a válasz szinte azonnal megérkezik. A SOAPsh.pl ezt írja ki:

```
--- SOAP RESULT ---
$VAR1 = ´ABC´;
```

Hú, ez nem semmi! Hálózaton keresztül kértem objektumeljárást. Nem is volt olyan vészes, igaz?
A SOAP könnyűnek látszik, ha csak egyszerű változókat küldözgetünk oda-vissza. De küldhetünk különböző típusú adatokat is. Például próbáljuk ki a `capitalize_array` parancsot egy tömb küldésével:

```
> capitalize_array(´abc´, ´def´, ´GHi´)
```

A visszaérkező értékek az alábbiak lesznek:

```
--- SOAP RESULT ---
$VAR1 = bless( [
         ´ABC´,
         ´DEF´,
         ´GHI´
               ], ´Array´ );
```

Az érkezett adatok egy kicsit viccesen néznek ki, mivel abban a formátumban látjuk, ahogy azt a SOAP::Lite küldi és fogadja. Hamarosan megláthatjuk, hogy a programunk hogyan kerüli meg ezt a „szakszerűtlen” formátumot, és hogyan cserélgeti észrevétlenül az összetett adatszerkezeteket az Interneten.

A SOAP vizsgálata

Ahogy azt az imént is láthattuk, a SOAP-pal lehet anélkül is dolgozni, hogy mélységeiben ismernénk az XML kódolás rejtett titkait. Mégis, a hibakeresés közben gyakran alakul úgy, hogy ismernünk kell az XML részleteit, legalább annyira, amennyire a HTTP fejléceket.
A SOAP::Lite objektumok támogatják az on_debug( ) eljárást, mely egy eljárás címét várja. Ezt az eljárást azután minden tranzakció után meghívja, lehetőséget adva ezzel számunkra, hogy az eseményekről naplót készítsünk, vagy folyamatosan megjelenítsük azokat a képernyőn. Egy egyszerű példa erre a következő:

```
on_debug(sub{print STDERR @_})
```

Megkérjük a SOAP::Lite-ot, hogy küldjön mindenről másolatot az STDERR-re. Így megtudhatjuk, hogy tulajdonképpen mi is történik a színfalak mögött. Az eljárás futtatása után a SOAPsh.pl emlékeztet, hogy egy helyi eljárást hívtunk meg, és nem a SOAP-on keresztül hívtuk azt meg:

```
--- METHOD RESULT ---
SOAP::Lite=HASH(0x82e1174)
```

Most láthatjuk, ahogy a capitalize(abc) kérés SOAP-kéréssé alakul át (lásd a *3. listát*).
Amint azt láthatjuk, a kérés két részből (a fejlécből és a törzsből) áll, mint minden HTTP-kérés. És mint minden átlagos HTTP-kérésnél, meg kell jelölnünk a tevékenységet („POST”), a teljes címet („URL”), a tartalom hosszát (Content-Length), és a tartalomtípust (Content-Type, ez mindig text/xml).
Most jön a lényeg: az utolsó sor a fejlécben a SOAPAction, amely megnevezi az objektumot és a hívás módját. A SOAPAction működését úgy alakították ki, hogy a vállalati tűzfalak kiszűrhessék a veszélyes objektumokat és tagfüggvényeket. Ennek ellenére jelenleg

7. lista cgi-soap.pl

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

use strict;
use diagnostics;     # A végleges programban
                       iktassuk ki

use SOAP::Transport::HTTP;

SOAP::Transport::HTTP::CGI
     -> dispatch_to('/tmp')
     -> handle ;
```

viszonylag nehéz SOAPAction támogatást találni. Akárhogy is, az objektum és a tagfüggvények adatai is be vannak ágyazva az XML törzsbe, ezért a fejlécet gyakran felesleges kielemezni.
Maga az XML egy XML bevezetővel kezdődik, majd egy SOAP borítékkal. A borítékon belül egy fejléc (nem kötelező rész) és egy törzs (kötelező rész) szerepel. A törzs nevezi meg az objektumot, valamint a használandó tagfüggvényt, az összes átadandó értékkel. Ezután a kapott anyagot a kiszolgáló átalakítja az operációs rendszer által értelmezhető formába, majd továbbítja azt a célobjektum felé. Az objektum által visszaadott értéket a kiszolgáló XML formátumra alakítja (*4. lista*).
A válasz, akárcsak a kérés, a HTTP-t használja, és a HTTP fejléc tartalmaz olyan metaadatokat is, mint a kiszolgáló típusa, a dátum, a tartalom hossza, a típus (text/xml), és végül a futtató SOAP kiszolgáló típusa. A boríték ebben az esetben (akárcsak a kérésben) nem tartalmaz fejlécet. Így a törzs tartalmazza a visszatérő értéket (xsd:string típusú). Míg a kérés a namesp3:capitalize névteret használja, a válasz a namesp1:capitalizeResponse-t. Ez egy szabvány a SOAP-ban; az XML névtereket használja, hogy azonosítsa, hogy kérdésről, vagy válaszról van szó, illetve, hogy milyen kéréshez tartozik a válasz.
Az *5. listán* látható (magyarázat nélkül) a capitalize_array(reuven, shira, atara) hívás hibakeresési kimenete.

Egy SOAP ügyfél

SOAPsh.pl jól példázza az interaktív kérések működését, de a SOAP sokkal hasznosabb, ha a saját programunk készíti el és dolgozza fel a kéréseket. A *6. listán* látható, ahogy a SOAP ügyfél a 8080-as kapun csatlakozik a kiszolgálóhoz. Látható, hogy az URI-ban még egyszer az objektum neve, a proxyban pedig a SOAP kiszolgáló neve szerepel.
Éppen ez a megoldás oly különleges a SOAP::Lite-ban, ahogy lehetővé teszi a hálózaton keresztül az objektumok tagfüggvényeinek lekérdezését. Az uri, a proxy és a result nyilvánvalóan léteznek a SOAP::Lite objektumban. De a capitalize eljárás csak a távoli Text/Caps objektumban létezik. A SOAP::Lite elég okos ahhoz, hogy észrevegye a különbséget, és képes kiválasztani azokat az eljárásokat is, amelyek helyben nem oldhatók meg.

A CGI-alapú SOAP kiszolgáló

A mi kis önálló kiszolgálónkat egyszerűként emlegettük, mert az is. Vajon mi történik akkor, ha naponta több millió kérés fut be? Ezt a mi egyszerű kiszolgálónk nem bírja, és a felhasználók kérelmeit nem teljesíti.
Célszerű megoldás a HTTP-forgalom kezelésére készített program, név szerint az Apache használata. Az Apache segítségével könnyedén kezelni tudjuk a kisebb és nagyobb forgalmat is. Az Apache haszná-

lata esetén a programunk nem kell foglalkozzon ezekkel a kérdésekkel, összpontosíthatunk a SOAP csomagok fogadásának részleteire és a Perl-modulok közötti kapcsolatokra.
A CGI-alapú proxy készítése nem sokban különbözik az önállóan futtatható program készítésétől. A legfontosabb dolog, amit fejben kell tartanunk az, hogy ne használjuk a CGI.pm-et, vagy egyéb ehhez hasonló CGI-s modult! A mi esetünkben a CGI-feladatokat a SOAP proxy valósítja meg, és a CGI protokoll használatával tudjuk az XML-kódolású üzeneteket oda-vissza küldözgetni.

Egyéb nyalánkságok

A SOAP::Lite rengeteg hasznos segédprogramot tartalmaz, ezért elég nehéz megállni, hogy az ember fel ne sorolja mindet. Például azoknak, akik a mod_perlt használják a CGI helyett, jó hír, hogy a SOAP::Lite beépített mod_perl-, valamint CGI-támogatással is bír. Saját programunkban kihasználhatjuk az „autodispatch” szolgáltatást, melyet az előbbiekben is láthattunk működés közben, ha egy helyileg ismeretlen tagfüggvényt hívunk meg, azt a rendszer továbbítja a távoli objektumnak.
A SOAP::Lite képes kezelni a legtöbb, SOAP által is támogatott adatszerkezetet, beleértve az objektumokat is. Például meghívhatjuk egy távoli objektum new( ) függvényét, majd a visszakapott objektummal további feladatokat hajthatunk végre. Ez a szolgáltatás már évek óta létezik számos különleges felületen, de az a tény, hogy a SOAP felületfüggetlen, valóban bámulatra méltó.
Végül, a SOAP növekedésére jellemző, hogy nem is olyan régen még csak a HTTP protokollt támogatta, napjainkban már olyan gyakori protokollal is együttműködik, mint például a POP3 és az SMTP, és a lehetőségek tárháza folyamatosan bővül.

A SOAP és a háromrétegű alkalmazások

Most, hogy láttuk, hogyan működik a SOAP egyszerű körülmények között, megfigyeljük, hogyan használható összetettebb környezetben. Tegyük fel, hogy szeretnék építeni egy hatalmas weboldalt egy relációs adatbázissal. A legtöbb esetben – mint ahogy a Kovácsműhely rendszeres olvasói is tudják – a feladatot a mod_perl és a HTML::Mason párossal szoktam megvalósítani.
Mivel a kiszolgálóoldali Java-piac is fellendült, néhány vagy az összes háttérművelet megoldható JavaBeanekkel is. Továbbá az Enterprise JavaBeans (EJB) egyre inkább fejlődő módszer a tranzakciókat használó elosztott alkalmazások számára, éppen ezért mindig is előnyben részesítem a javás megoldásokat.
A SOAP-pal, most kedvem szerint, szabadon keverhetem a nyelveket és a felületeket. Ha létrehozok egy SOAP kiszolgálót, amely hozzáfér a megfelelő Java-objektumokhoz, nincs semmi oka, hogy a Mason-elemek ne tartsanak kapcsolatot a középső Java-réteggel. Néhány esetben ez még jobb eredményekkel is járhat, mintha csak egyetlen nyelvet használnánk. Mivel a Perl nem támogatja a hálózati tranzakciókat, ezért a kezdetekben Perlben készített programot később majd az EJB segítségével szeretnénk továbbfejleszteni.
A SOAP használatával mindez lehetséges, sőt, egyenesen kívánatos.

Forrásanyagok

Számos weboldal és írás foglalkozik a SOAP-pal, mégsem láttam túl sok példaprogramot a használatára. Remek kiindulási pont *Dave Winer* oldala
➲ http://soap.weblogs.com/
Itt találhatunk magyarázatokat SOAP-megvalósításokra, és a SOAP hibák felderítésére a weblog segítségével.

A World Wide Web Consortium által közzétett SOAP szerkezet ➲ http://www.w3.org/TR/SOAP/
A SOAP::Lite modul letölthető az alábbi címről

➲ http://www.soaplite.com/

Paul Kulchenko, a SOAP::Lite szerzője, keményen dolgozik a modul fejlesztésén, és megfizethetetlen segítséget nyújtott a hibakeresésben e cikk megszületése során.

Végezetül, a webes alkalmazáskiszolgálók már elkezdték támogatni a SOAP-ot. Nem csupán azt teszik lehetővé, hogy más gépeken lévő, különböző nyelveken írt objektumok kapcsolatot tartsanak egy kiszolgálóval, de megnyitják a teret olyan internetalapú szolgáltatások felé, melyek nincsenek szükségszerűen hozzákötve a Webhez. Lehet, hogy hamarosan némelyik webes újság SOAP-alapú hírkezelő rendszert biztosít, amin keresztül egyetlen kéréssel lehívhatjuk az össze bennünket érdeklő hírt. Egy ilyen szolgáltatásra alapozva a felhasználó kirakhat az asztalára egy (nem webes) programot, mely folyamatosan a legizgalmasabb híreket mutatja.

Zárszó

A SOAP az új típusú elosztott internetes alkalmazások előfutára, jelenleg az egyetlen, amely képes távoli eljáráshívásra operációs rendszerek és programnyelvek között. Nem kell tovább vesződni a „nehéz megérteni”, vagy a „nehéz meghívni” típusú eljárásokkal, és akár egy délutáni oktatáson el lehet készíteni egy egyszerű elosztott alkalmazást. Hogy mindez mit jelent, az Internet és a Web jövőjére nézve? Jó kérdés, de már készülnek azok az asztali alkalmazások, amelyek grafikus felülete SOAP-kéréseket továbbít a központi kiszolgálók felé. Attól függetlenül, hogy mit hozhat a jövő, az a tény, hogy a Perl és a többi szabad nyelv használja a SOAP-ot, azt jelenti, hogy hamarosan könnyebb lesz a kapcsolattartás, mint valaha. Vajon ki ne akarná az egész Internetet? (Ide nekem az oroszlánt…)

Reuven M. Lerner
(reuven@lerner.co.il) egy izraeli web- és internet-tanácsadó cég tulajdonosa és vezetője. A Kovácsműhely rovat honlapja a ➲ http://www.lerner.co.il/atf/ címen érhető el.