Session:F07

How to cope with soap? Web Services on DB2 for z/OS

Frank Petersen
Bankdata, Denmark

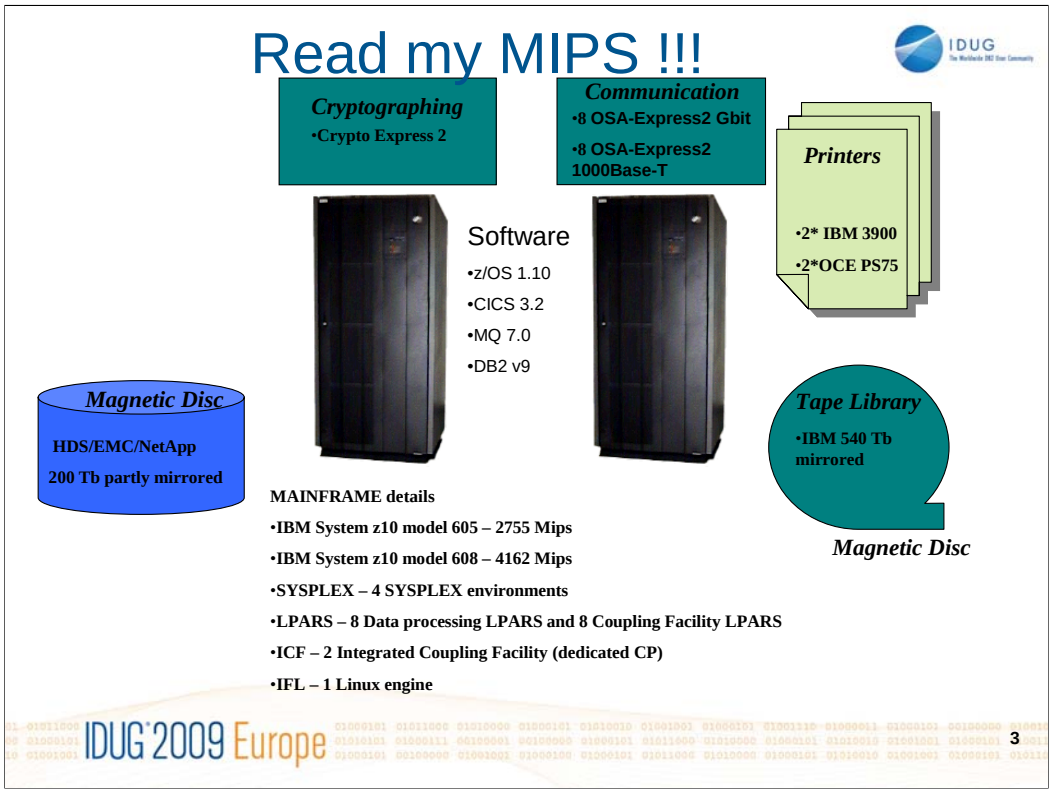
IDUG
The Worldwide DB2 User Community

Tuesday 6th October 2009 •14.15-15.15
Platform:z/OS

How to cope with soap? WebServices on DB2 for z/OS

- Objectives :
  - Make you see the possibilities in using Web Services running DB2 for z/OS
  - Make you understand the ***needed*** terminology
  - Make you able to code a DB2 call to a Web Service
  - Make you able to understand how DB2 works as a service provider
  - Useful ways to exploit this as a DBA or a SysProg.

This presentation will give a walk through of the possibilities of using Web Services when running DB2 for z/OS. It will give a brief explanation of the terminology, but will try to look at web services from a practical angle instead from a theoretical angel. There will be coding examples of how to call a Web Service running at DB2 for z/OS from Visual Basic (.NET) using a browser and there will be examples of how to call Web Services running 'out in the world' or at a in-house Tomcat server from a standard z/OS COBOL/DB2 program. It will be shown how to make a z/OS REXX into a callable Web Service, thus making these callable from a piece of .NET code from a browser. And do you think you can display a z/OS dataset onto a browser with 2 minutes of code? And what about displaying SDSF-information on a browser ? And getting MSSQL data directly into a z/OS COBOL program ? This session will enable you to go home and be able to do this sort of things !



Some boxes with different sizes, but always toooo small !!!!

How to cope with soap? WebServices on DB2 for z/OS

- Agenda :
  - The development of Web Services (brief ☺)
  - The current “status” of the software.
  - Let’s see which words you need to understand.
  - How to code a Web Service running under DB2 for z/OS (DB2 as a provider).
  - How to code a call from DB2 for z/OS to a Web Service running at a ‘random site’ (DB2 as a consumer).
  - Examples including “how this can change your life” as a DBA or technician !!

The agenda for the presentation.

We need to understand a few words, but I promise the description will be brief. In this presentation we will focus more on the usage and the practical issues in using Web Services.

The history of WebServices

- Arrived at the scene around year 2000.
- A romantic idea of public web services everywhere.
- Reality soon took its toll.
- Finding useful, free, public web services is next to impossible.
- Is Web Services yet another of these “fast in – fast out” things where the time it takes to learn them is approx the same as their live time ? (CORBA ?, Ruby ?)

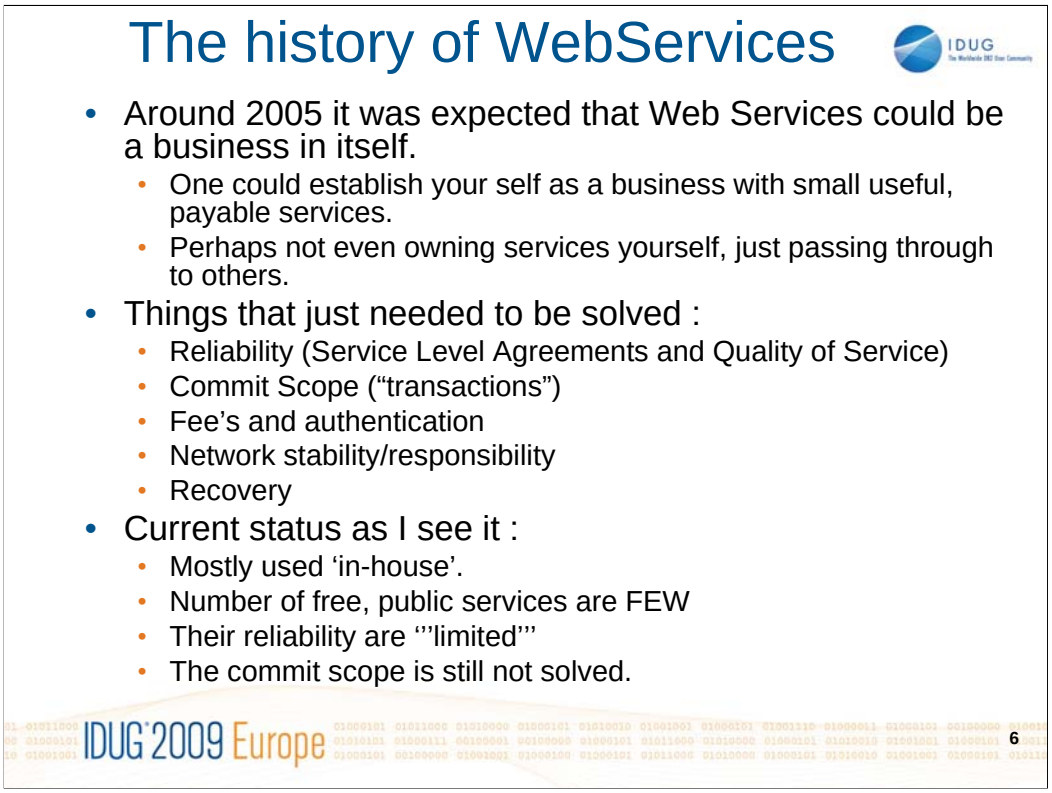
The history of Web Services.

When I saw the first presentations of Web Services there was this imagination of a cloud of free services that would exist around the globe that we all could call. For instance if we needed information of a ISBN-number we could just call a service. Or an exchange rate.

-

When I wanted to build this presentation I was chocked over the few number of good, reliable and free services I could find.

When one has been around this industry for ‘a few years’, one is always skeptical whether this is yet another of these things that are alive for just a few years, soon to be replaced by something else, more fancy.



The problems that seem to have slowed the progress of Web services in the direction it was intended was a strange thing called reality !!

It was expected that one could be in business simply by hosting Web Services or just by relaying to other sites. However before someone will use a service on the network for business purposes they want guaranty for QoS and response times. And this costs money - so most services end up being payable.

The "UOW" thing (as we database people know it), is not present so no one can control a "ROLLBACK", should an error occur later in the transaction after a updatable Web Service has been executed.

However the usage of Web services locally (inside an installation) can be extremely beneficially. I hope that it will be clear why after this hour !

Words don't come easy…

- To be able to use Web Services and/or to take part in the discussions a few words are enough:
  - UDDI
    - Universal Description, Discovery and Integration
  - SOAP
    - Simple Object Access Protocol
  - XML
    - Extensible Markup Language
  - WSDL
    - Web Services Description Language
- Good news : you just need very basic knowledge.

As I promised we will only look very briefly at the theory and the words.

We will however have to understand what UDDI is and why we really don't have to care. And SOAP as well – if not for anything else then to be able to understand the title of this presentation.

XML, however, has to be understood as resonable level. This is needed anyway as I assume no installation will be running without XML usage inside their DB2 system in a few years from now.

And when we have talked about XML, we can take the last step into WSDL.

Words don't come easy…

- UDDI intentions:
  - Supposed to be the link between service providers and consumers
  - Just like "Yellow Pages"
  - Should have been the core anchor point of Web Services
- UDDI problems:
  - Multiple UDDI services – which one to use
  - Who decides which service is returned first by the UDDI provider if multiple services exists ?
  - Who pays ?
  - Who earns money ?
- UDDI today:
  - Several UDDI have been closed.
  - If Web Services are an 'in-house thing', UDDI has no importance !
  - UDDI could be a dead-end ?

UDDI was supposed to be a number of dictionaries where one could look up when a Web Service is needed. Here we could find the Web Services that could help us and find where they reside and how to call them. Either as a human intervention or programmatically. And if a service moves to another physically URL, it would be reflected in the UDDI thus picked up automatically.

However, as Web Services have ended up to be mostly a 'in-house' thing UDDI's are not used that much. Several of the big actors have closed down their UDDI's as running a UDDI in itself is an expense.

In my magnifying glass I think that UDDI's will be a thing that never will play a really important role.

Words don't come easy…

- SOAP – The protocol to wrap a Web Service call
  - An XML wrapping of keywords to enable the understanding of the request by the partners.
  - The sending mechanism is normally HTTP (or HTTPS)
  - The root element is the SOAPENV :

```
<soapenv:Envelope
xmlns:q0="urn:example
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
<soapenv:Body>
<q0:Statement4><VAR01>A00</VAR01></q0:Statement4>
</soapenv:Body>
</soapenv:Envelope>
```

  - Body contains the application-only parameters.
  - xmlns is definitions of namespaces and in my opinion you don't have to care about these in this context.

Now let's address SOAP. This is a wrapping of information that can be transported in different ways. In most cases we will see it being embedded over the HTTP protocol. That the last P in SOAP stand for protocol is (in my opinion) an overkill. SOAP is really not a protocol but a wrapping so it should perhaps have been SOAW.

There are some complicated parameters in the start of the SOAP-data, xmlns, which is so-called namespaces.

If you are new to SOAP, forget them. If anyone asks you at a pub, simply say "This is namespace settings".

What we need to look at is the SOAP envelope, which contains all the data our Web Service needs and returns. All the SOAP-data are naturally XML strutures.

Words don't come easy…

- XML – Is this just a rebirth ?
  - XML is indeed simple.
  - Start and stop tags have been around for decades.
  - Attributes and parent/child philosophy are well know in the industry.
- No – XML is genius !
  - One of the strong things are the enforcing of "well-formed" documents.
  - The implementation of parsers in all languages and environments can not be overrated.
  - Can be used for everything :
    - Parameter blocks between programs.
    - Protocol definitions (like in SOAP)
    - Parameter decks for definitions (why not ZPARM ?)
    - Storing of data (but do take care)
    - Exchange of data between programs and environments.
- Could XML be a 'clone' in the database evolution?
  - Storing data without 'fixed' columns and data types.
  - Reading of data without knowing its context ?

XML has been debated a lot amongst older mainframe people. We have seen it all before, haven't we ??

We know that html-codes are just GML/DCF-tags in a newer version, don't we ?

And XML is just a smart way of wrapping parameters and we have done similar things in most installations 30 years ago !!

NO, NO, NO – XML is more that this. XML ensures a thing called "well formed structures". This guaranties that all parameters are set correctly and that someone else, somewhere else can read the document. Style sheets , schemas and parsers ensures that this facility is available every where !! And modern languages like Visual Studio even allow XML navigations to be entered at same level as SQL on objects !

This leads me to the next phrase; that XML to some degree will be/or is a mutation of a database. Relational DBA's etc have meet the first developers that just want a table with a few huge columns, where the programmers just want XML inside to be independent of number and format of columns. And with the Pure XML and the integration in modern languages we can almost say that we are there !

Words don't come easy…

- WSDL - the strongest feature of Web Services ?
  - Documents data types, how to call and where to call
  - Naturally in XML…..
  - When you look at a WSDL in your browser don't panic :
    - most of it is irrelevant (to you).
  - A few fields is of interest in this context – see later
  - The WSDL is normally built automatically.
  - The integration with development tools is simply elegant !!
  - When you want to exploit a Service : point to the WSDL:
    - Will help build most of the code
    - Some tools even make all this transparent to the developer.

This is perhaps the cornerstone of Web Services. The WSDL ensures that a Web Service will describe itself in every needed aspect !! No matter platform or web server software.

-

The integration is both at the development side and at the execution side and will allow an easy usage of a Web Service.

We will see, however, that when using DB2 as a Web Service consumer, DB2 t not obey the rules and use the WSDL.

The DB2 implementation.

- DB2 as provider (hosting a service).
  - The fact is that DB2 does not know that it owns a Web Service.
  - Using a tool like IBM Data Studio (Optim Development Studio) :
    - Import existing Stored Procedures or SQL pieces.
    - Build new ones and create these at the DB2 in question.
    - Then build and deploy to a Web Server (WAS, Tomcat or..)
    - At the Web Server all wrapping and connection code is taking place.
    - Really 'just' an automated 'EXEC SQL CALL MYPROC USING :A,:B'
  - At the end of the day, it is very important to understand that DB2 has NO build-in Web Server. So an external one is needed !!!!!!!!
  - Therefore NO installation, customization nor maintenance needed at DB2.
  - But, naturally, you need infrastructure to handle Stored Procedures.

When we are looking at DB2 as a provider of Web Services, we have seem many presentations where a stored procedure is dragged to a newly defined Web Service. Or a specific piece of SQL.

It is clear that stored procedures and Web Services are much alike. Both have defined the parameters in and out. And the execution environment as well. Stored Procedures, however, have no WSDL or anything that will do the connection to the database.

When we are talking about DB2 for z/OS I will focus towards using Stored Procedures as these will have a package on DB2. If using SQL this will not be known to DB2 before it arrives to be executed.

What many of these presentations fail to notice is the fact that DB2 does NOT KNOW that one of its stored procedures have been promoted to a Web Service. In Data Studio you deploy the Web Service to a Web Server and this means that all the wrapping is shipped to that server together with the WSDL and XML-files describing the connection etc. In this way these two tings compliment each other !

The DB2 implementation.

- DB2 as Consumer (calling a service).
  - The implementation is done by using IBM delivered UDF's.
  - These have to be installed (=created) and maintained.
  - The first delivery used UDF's SOAPHTTPV and SOAPHTTPNC.
  - From V8 : use the UDF's SOAPHTTPNV and SOAPHTTPNC.
  - No Bind required !! Just create them using SDSNSAMP job DSNTIJSG (V9) or DSNTIJSP (V8).
  - Change to use a new WLMENV to be able to isolate from other.
  - Be aware that rerunning DSNTIJSG/DSNTIJSP will destroy your changes. So document these.
  - In special cases it may use some of the system stored procedures, for instance SYSIBM. SQLFUNCTIONCOLS.
  - The WLM environment has to be defined and a set of JCL be created.
    - Total time required to setup ? 1 hour !!

When we are looking at DB2 as a consumer of Web Services, this support was delivered in V7 and revised in V8. If you want to look at the delivery in detail a good PTF is UK31856.

The implementation is done by UDF's and the installation is extremely easy. So easy that one thinks one has forgotten something. In a V9 installation the UDF's are assumingly already installed and no Bind is needed. The UDF's have direct socket programming and the modules are in SDSNLOD2.

I advise you to make a new WLM-environment and a new set of STC JCL for that. Expect to be finished within one hour!!!!

The DB2 implementation.

- WLM considerations
  - Running in its own environment can be beneficially
    - More controlled settings of environment variables
    - Need an OMVS output file :
      //WSERROR DD PATH='/tmp/wsc.db2vwlmu.err',
      // PATHOPTS=(ORDWR,OCREAT,OAPPEND),
      // PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP,SIROTH,SIWOTH)
    - This file can not be read at a 'normal' ispf screen as it will be in unicode :
      Go to an OMVS screen and type :
      iconv -f UTF-8 -t IBM-1047 < /tmp/wsc.db2vwlmu.err
  - SDSNLOD2 has to be at STEPLIB.
  - Default timeout value of 2 minutes if the endpoint is unavailable.
  - Can be overruled with a environment variable at the UDF level :
    RUN OPTIONS 'POSIX(ON),XPLINK(ON),
    ENVAR("_CEE_ENVFILE=/u/ubifap/soap.envvars")'
    /u/ubifap/soap.envvars contains :
    DB2SOAP_TIMEOUT=300

The WLM environment can use many TCB's (use 10), but need a WSERROR USS file. When having problems look in this file but you will not be able to see anything if you look from a USS shell.

-

The file is in unicode so you have to FTP it to a PC or use the command mentioned on the slide to view it.

Remember to put SDSNLOD2 onto STEPLIB and if you follow my idea on the last few slides about REXX as Web Services you need a SYSEXEC as well to point to a dataset with the REXX-code.

If the endpoint of the Web Service is not available, there is a timeout value of 2 minutes. This can be overruled at the UDF-level OR at the WLM environment level as indicate don the slide.

The DB2 implementation.

- Now we have the environment up and running.
- We can use Web Services both as consumer and provider.
- To verify you can call your own stored procedure wrapped as a Web Service (provider).
- Or you can call a simple Web Service you build yourself in for instance Visual Studio (consumer).
- It is so essential that you try this yourself from scratch.
- Let's see……

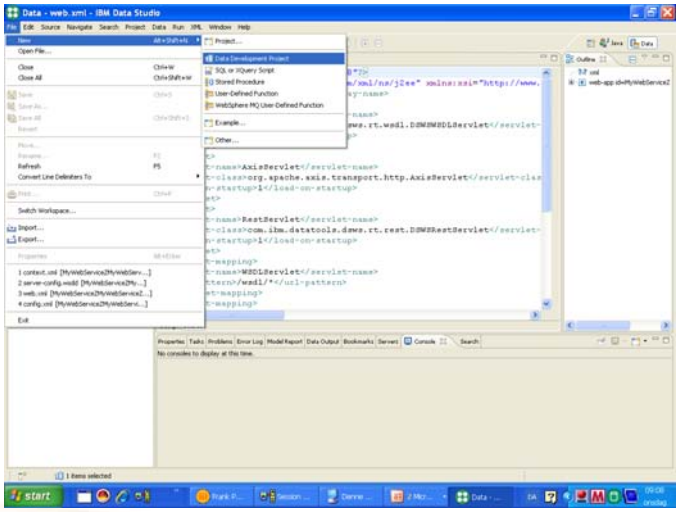
All is now installed and you have opened DB2 up for Web Services in both directions !

But surely we must validate the installation and as the aim of this presentation is to encourage you to use Web Services in you daily work as a DBA or Systems Programmer, we need to be sure you understand how to code programs to use the facility.

So let's try this :

Building a DB2 Web Service.

- Use Data Studio(Optim)
- Define new ”Data development project”



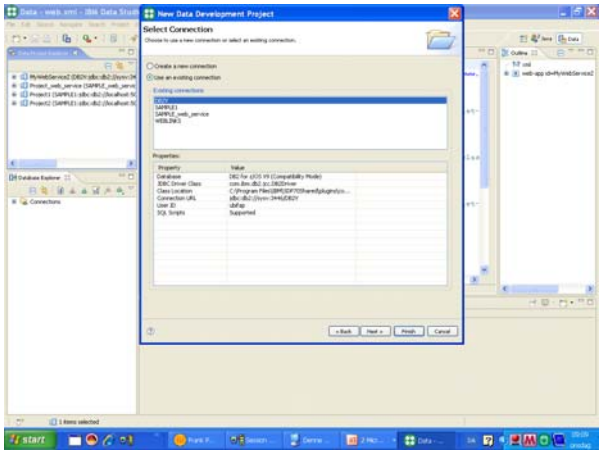
At the time of writing I intent to do it live at my laptop, screenshots just provided as ‘backup’ or to document what is done for readers of the foils “outside class”.

Download and install Visual Studio (now part of Optim) on you workstation.

Make a new project as a “Data development project”.

The DB2 implementation.

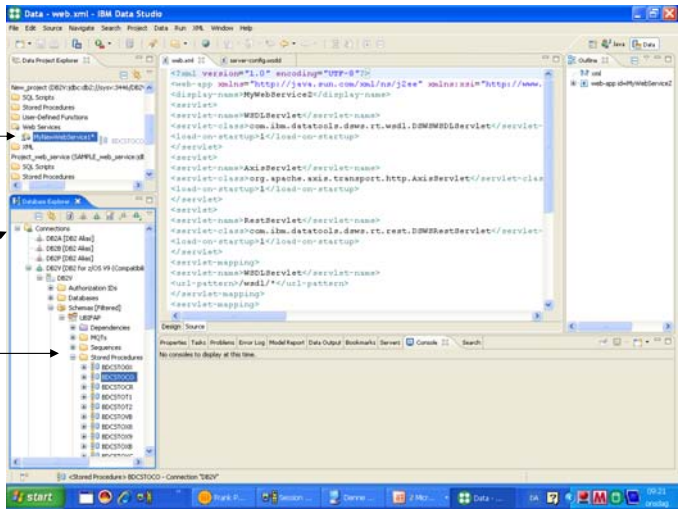
- Build a connection to DB2 on z/OS
- Please note that you will be required to type in the password when a connection is defined.
- How will we change this password later ? We will see !!



Open up the DB2 connection pane and define a new connection towards the requested DB2 subsystem or reuse an existing connection if you have one from earlier.

The DB2 implementation.

- Define a new Web Service under the Web Service bullet in upper left panel.
- Explode the connection in the bottom left panel.
- Explode the ”Stored Procedure”
- You can filter on Schema !
- Drag the SP up to the new Web Service



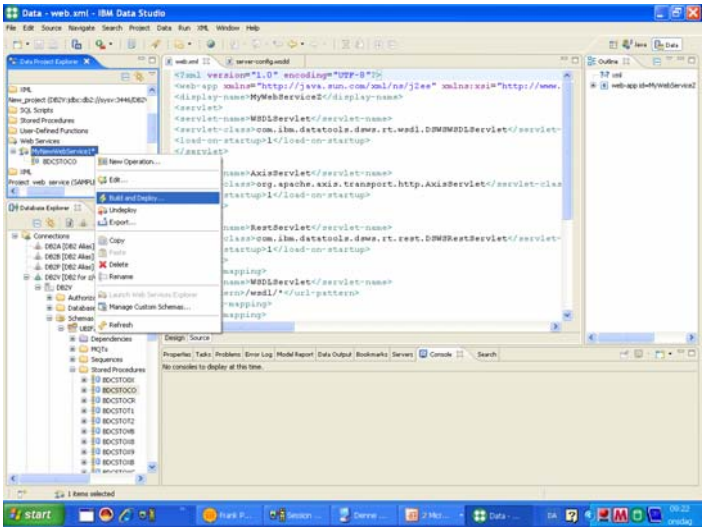
Define a new Web Service.

Explode the DB2 until you find Stored Procedures. You can use a filter to limit the amount of information returned.

When you have found the desired Stored Procedure simply drag it up on the new Web Service you made.

The DB2 implementation.

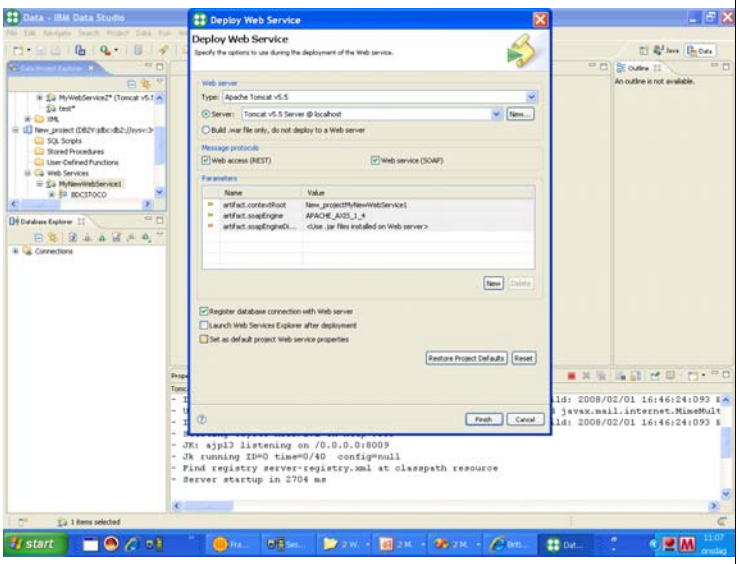
- Right click on the new Web Service
- Pick ”build and deploy”
- We are now ready to build all the wrapping to a Web Server of your own choice



By right clicking on the new Web Service you can choose “build and deploy” and all the wrapping and connection stuff is send to the Web server you choose.

The DB2 implementation.

- On this panel all the available Tomcat's and IBM WAS are present.
- Pick the one you want. Here I use a Open Source Tomcat installed
- No server available ? You can download a WAS CE for free or a Tomcat.



The Web Server can be a Tomcat or WAS.

Now we can use the Web Service !

The DB2 implementation.

- After deployment it is always nice to see something familiar :
  - Behold – a WSDL file !!!
  - Deep down there is a config.xml file containing :

```
<dsc:call operationName="BDCSTOCO">
<![CDATA[ CALL "UBIFAP"."BDCSTOCO"(:p1, :p2)]]>
<dsc:property name="procedureName" value="BDCSTOCO" />
  <dsc:property name="procedureSchema" value="UBIFAP" />
<dsc:parameter isNullable="false" isSigned="false" jdbcType="12" mode="in/out"
precision="254" scale="0" typeName="VARCHAR" />
  <dsc:parameter isNullable="false" isSigned="false" jdbcType="12" mode="in/out"
precision="2000" scale="0" typeName="VARCHAR" />
```

  - Somewhere you will also find a context.xml with :

```
maxWait="5000" maxIdle=2 maxActive=4 DriverClassName="com.ibm.db2.jcc.DB2Driver"
url="jdbc:db2://sysv:3446/DB2V:maxStatements=3;" username="ubifap" password="xx"
name="jdbc/MyWebServiceZ"
```


There is nothing as the joy of recognition !! So after you have deployed try to dig into the files produced.

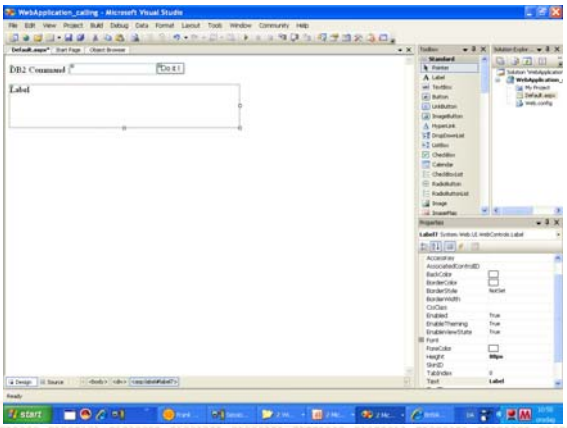
You will find a WSDL file. Looking into it you will find the XML-stuff that describes your stored procedure.

You will also find a config.xml file that will contain the actual CALL statement. If we had not picked a Stored Procedure but instead build a SQL script that SQL would have been here !!!

Finally you will find a context.xml file that will contain all the connection related stuff. This is important as you might need to change userid and password at some point in time and perhaps the DB2 subsystem id and port number as well. The “name” in the last line in the JNDI-name as it will have to be defined as Data Source in WebSphere

The DB2 implementation.

- Now let us call this new Web Service :
- For instance using Visual Studio 2005.
- Define a new WebProject and drag some controls to the canvas.
- The Stored Procedure used here will execute a DB2 Command :



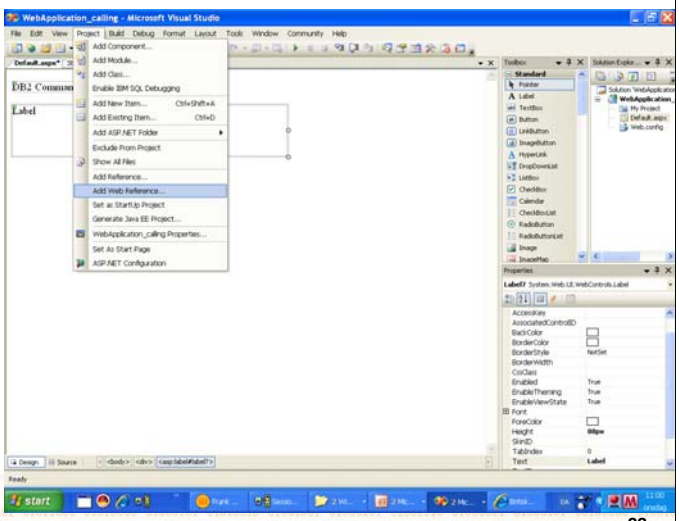
So now we will try to call this new Web Service. As I love Visual Basic I will prefer to try it from here.

I have a z/OS stored procedure that executes DB2 commands and at IDUG in Warsaw 2008 I showed how to call this directly from VB. Now we will take it one step further and try it as a Web Service.

I define a few fields on the panel for the DB2 command and the answer returned.

The DB2 implementation.

- We know that we need to find the WSDL
- ”Add Web Reference” will do this!

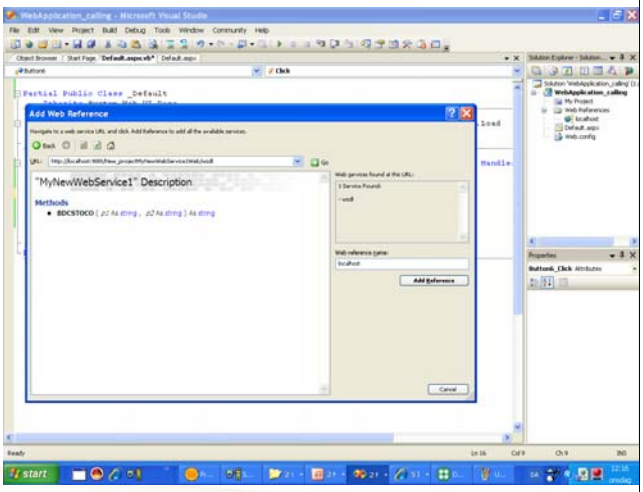


As you should have understood by now, the development tool (here Visual Studio 2005) will need the WSDL to help you build code.

Pick “Project – Add Web Reference” to feed VS with a WSDL URL

The DB2 implementation.

- Here you will need the URL for the WSDL
- When you deployed earlier, you could click on ”Lauch Web Services explorer” to find it !
- Click ”Add Reference”



The big question is “how do I know the URL for that WSDL”. Easy - go back to Data Studio and instead of “build and deploy” select “Lauch Web Services Explorer”.

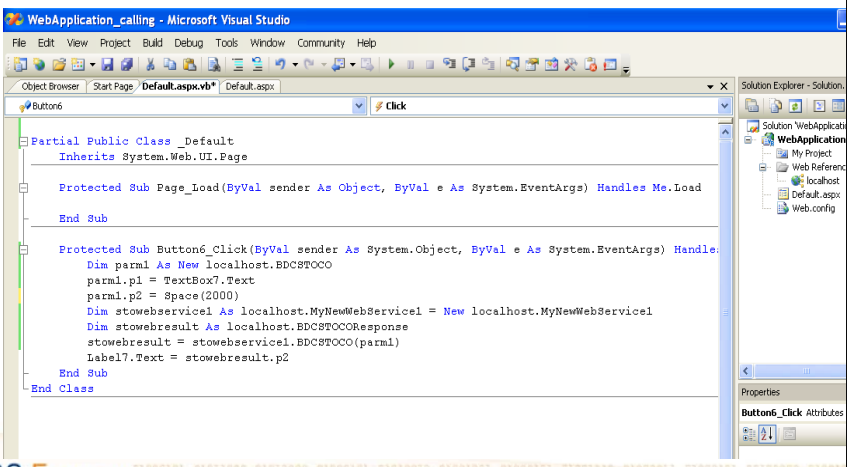
This will fire up a browser with a screen showing the picture taken from the WSDL. On the first page you will even find the URL for that WSDL to copy into the field on Visual Studio.

Visual Studio now reports back with a screen where you can recognize things from you stored procedure. It can be a good idea to give it a understandable name in “Web Reference Name” before you click on “Add Reference”.

After this Visual Studio has been through the entire WSDL and now it knows everything about you Stored Procedure or your Web Service to be more correct ! There will now be objects to relate to in the coding window, where we will progress to on next slide :

The DB2 implementation.

- Now we build the needed code driven when clicked at the button. 7 (seven) lines !!!
- Observe no connection, just simple code.
- The layers at the Web Server is dealing with all the boring stuff !

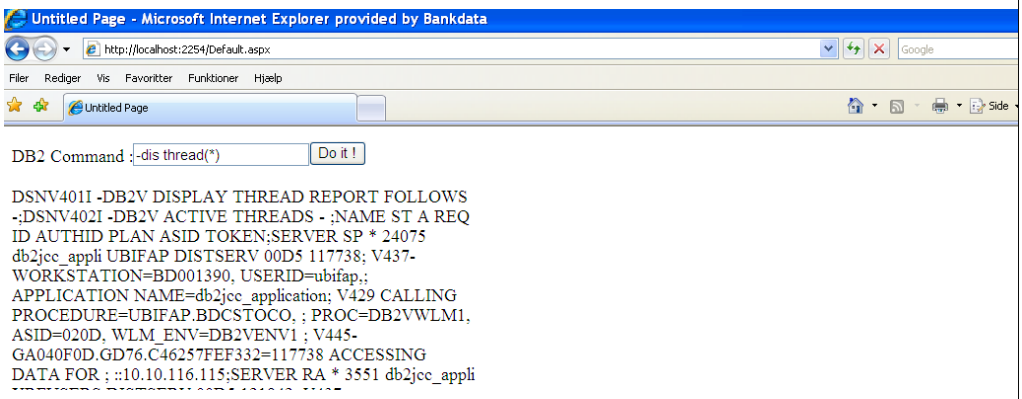


Let's build the code driven when the button is clicked. Double-clicking the button will open up the coding panel and we will build the needed 7 lines of code to be able to show DB2 commands on a browser.

All the wrapping etc is taken care by of the Web Server.

The DB2 implementation.

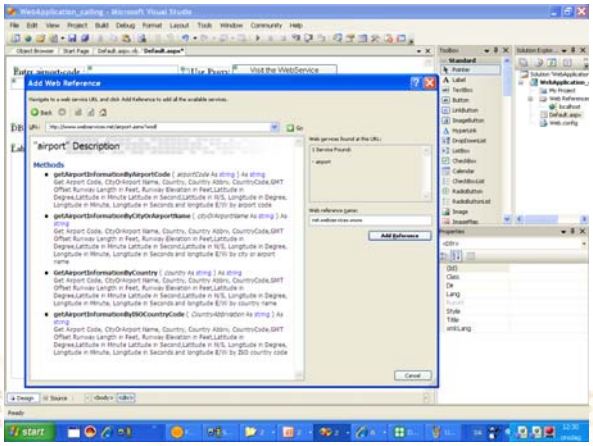
- Now let's try my stored procedure that is doing DB2 commands :



As nobody will believe it works – here is the proof. The result from a DIS THREAD back on a browser with max 7 lines of code !!

The DB2 implementation.

- Before we try to do a call to a Web Service from SQL, let's build a call to an external service.
- There are public services at a site www.webservicex.net
- You can translate an airport code to description data.
- On their site you can find the URL to the WSDL etc.
- So we add a reference :



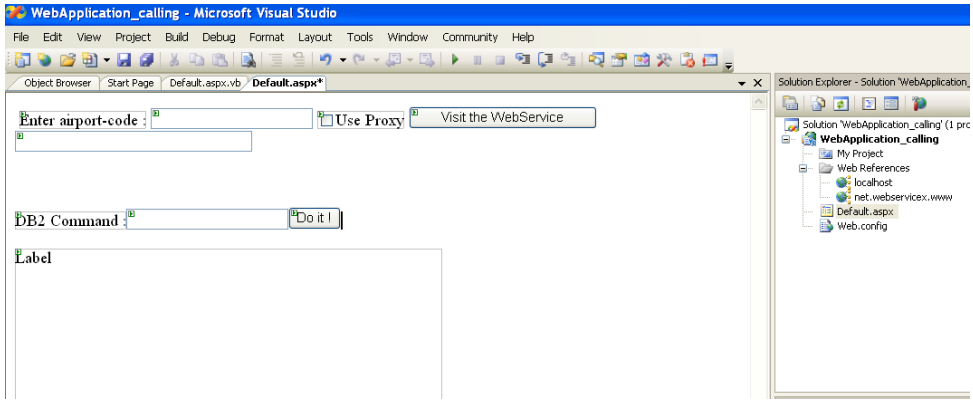
In the previous slide we called a Web Service with DB2 as Web Service provider. When I try DB2 as Web Service consumer I will need to call an external Web Service to make the picture more simply. Of course I could call a Web Service from DB2 hosted under DB2 but it would mess the presentation up.

To get ready for calling an external Web Service from DB2 I will try to call it from Visual Basic. So an exercise much like the previous one, but I will use a public and free Web Service that can translate a 3 character airport code to some physical data about the airport.

I go to their site and find the WSDL to be ready to build my VB-program.

The DB2 implementation.

- Let's define a few fields to the form.
- As we now leave our network we have to do some proxy stuff !



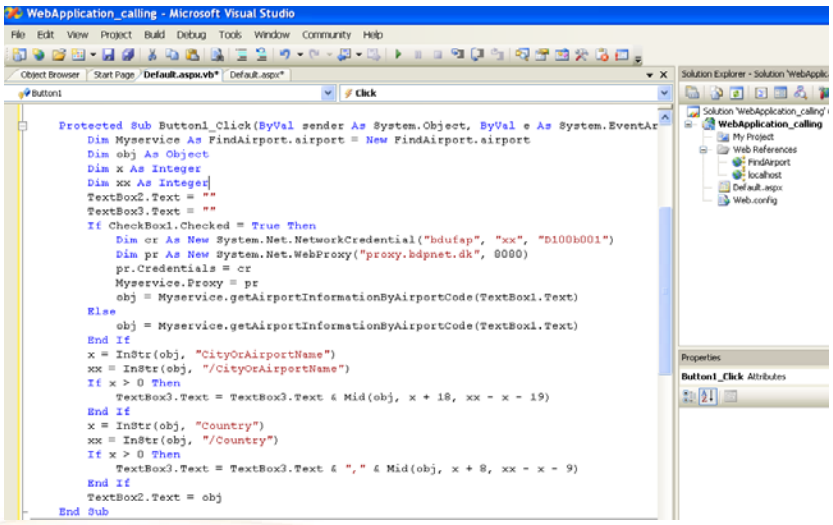
To verify that we can call an external Web Service, we will do a call to that service from a VB program to be sure that everything works as we expect.

So we define a few fields on the canvas for typing in the airport code and for displaying the data that is returned. Normally you will need a proxy definition to be allowed to exit your network and 'going public'. So we add an indicator to set if we want to use a proxy.

In a few slides we want to call this external Web Service from COBOL. This means we will run a batch COBOL program, that will translate an airport code using this service. When I did this I realized that is was difficult as I could not call directly from COBOL because of security settings in the installation that will not allow processes on z/OS to exit the secure network. I assume this will also be that case in most other installations. What worked for me was to try on a windows server and I realized that I could call the Web Service from here. And I was also allowed to call that server from my COBOL program. So I just made a Web Service that would pass the call on to the external service. So now I could call that service from my COBOL program and in this way get out on the server and here that service would just pass the call on to the external service. So a call to a Web Service that is just relaying the call to another Web Service !!!

The DB2 implementation.

- Here we have the code.
- Notice that the result comes in XML, but I don't strip it down !

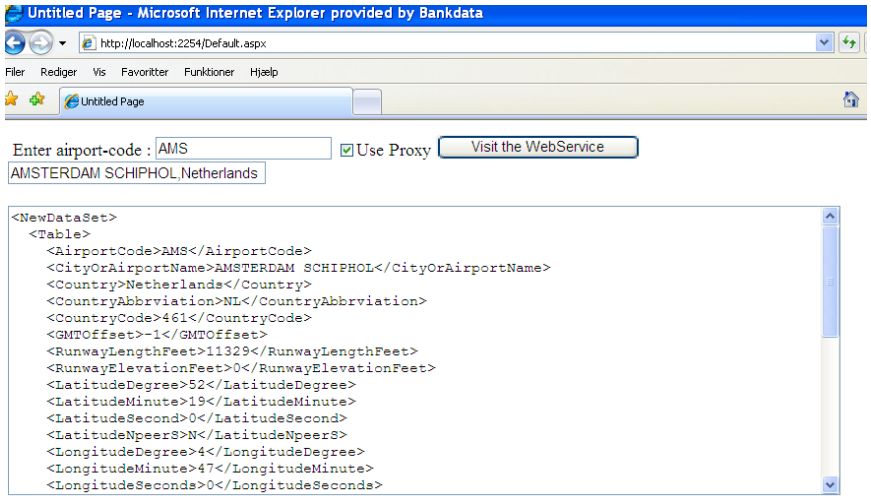


Here you can see the code that will call the external Web Service handling an airport code. Notice the check on the proxy checkbox that will set all the proxy properties.

The return data from that Web Service is naturally XML, but to keep it simple for this demonstration, I do a simple search for the needed tag instead of using DOM or XPATH/XQUERY on the XML.

The DB2 implementation.

- Let's try it using AMS (Amsterdam Airport)



Just a shot to prove that the thing indeed works !!!

The DB2 implementation.

- What have we learned so far ?
  - We can promote a SP to become a Web Service
  - We can call this Web Service from a Browser function
  - We have verified that we can call an external Web Service.
  - We can install the UDF's needed to call Web Services from SQL.

- What to do now ?
  - Let's try to call the 2 Web Services above from SQL
  - From SPUFI
  - And from a COBOL program (batch and CICS)

Now we have an idea of how we let a Stored Procedure become a Web Service and how we can build a VB program to execute under a browser and in this way get the result from these Web Services back on the browser.

We know we can call a DB2 hosted Web Service and we are also able to call an external Web Service from a VB program.

We have also installed the DB2 UDF's needed to let DB2 on z/OS become a Web Service consumer. The only thing we need is to try to call the 2 Web Services we have just seen directly from DB2. In the next slides we will see how we can do it from SPUFI and/or COBOL.

The DB2 implementation.

- All the UDF's require 3 input parameters:
  - ENDPOINT_URL VARCHAR(256)
  - SOAP_ACTION VARCHAR(256)
  - SOAP_INPUT

- They all return one parameter.
- The format of 3. input parameter (SOAP_INPUT) and the return parameter will vary between the 4 UDF's.
- You can set the DB2SOAP_TIMEOUT variable at the "RUN OPTIONS" instead of at the WLM JCL level.
- All 3 parameters will be set according to the WSDL of the service.
- No help at the moment in DB2 for this.
- Someone could easy build a shareware tool and become famous ☺

The 4 UDF's in the DB2 implementation all has 3 parameters : endpoint, action and envelope. The envelope will vary in format and size depending on which of the 4 UDF's you use.

Remember that there is a default timeout value of 2 minutes in the socket programming in the UDF's. There is an environment variable DB2SOAP_TIMEOUT that we can set at the JCL level for the WLM environment or directly on the UDF.

The challenge here is that there is no development tool involved here that can take the WSDL and help build the 3 parameters. So we must walk through the WSDL manually and build the parameters. If anyone need a challenge there could be basics for a little tool here !!!

The DB2 implementation.

- The UDF's :

Name	Soap-parameter	SDSNSLOD2
SOAPHTTPNVICO	IN:VARCHAR(32672) Out:CLOB(1 M)	DSNWSCVC
SOAPHTTPNCICO	In:CLOB(1 M) Out:CLOB(1 M)	DSNWSCCC
SOAPHTTPNVIVO	In:VARCHAR(32672) Out:VARCHAR(32672)	DSNWSCVV
SOAPHTTPNCIVO	In:CLOB(1 M) Out:VARCHAR(32672)	DSNWSCCV

Here we have the 4 UDF's with their names, parameter structure and the name of the module in the SDSNLOD2-dataset.

The DB2 implementation.

- Doing the call :

```
SELECT DB2XML.SOAPHTTPNV (ENDPOINT_URL,SOAP_ACTION,SOAP_INPUT) FROM
SYSIBM.SYSDUMMY1
```

ENDPOINT_URL : Open the WSDL in a browser and search for ”soap:address location”. The value here is the endpoint :
<soap:address location=
"http://localhost:9080/MyWebServiceZMyWebServiceZWeb/services/MyWebServiceZ" />

SOAP_ACTION : Open the WSDL in a browser and search for ”operation soapAction”. The value here is the action:
”soap:operation soapAction="http://tempuri.org/HelloWorld" style="document" />

SOAP_INPUT : Here you need the envelope. The site driving the service will use the WSDL to return calling information and test-facilities.
For the earlier mentioned Airport code translator look at :
http://www.webservicex.net/airport.asmx?op=getAirportInformationByAirportCode.
Here you can see the envelope needed.

Here we have an example of a call to a Web Service. Here we select from SYSIBM.SYSDUMMY1 to get one call, but naturally we could have taken part of the parameters from another table and in this way have called a Web Service many times in the same SQL. It would also be possible instead of building the 3 parameters directly in the code, to insert these in a DB2 table to isolate changes in the parameters away from the program.

On the next foil I have described how to find the things to use as the 3 parameters by looking in the WSDL.

The DB2 implementation.

- From SPUFI :

Endpoint for the earlier used Web Service

Action for the Web Service

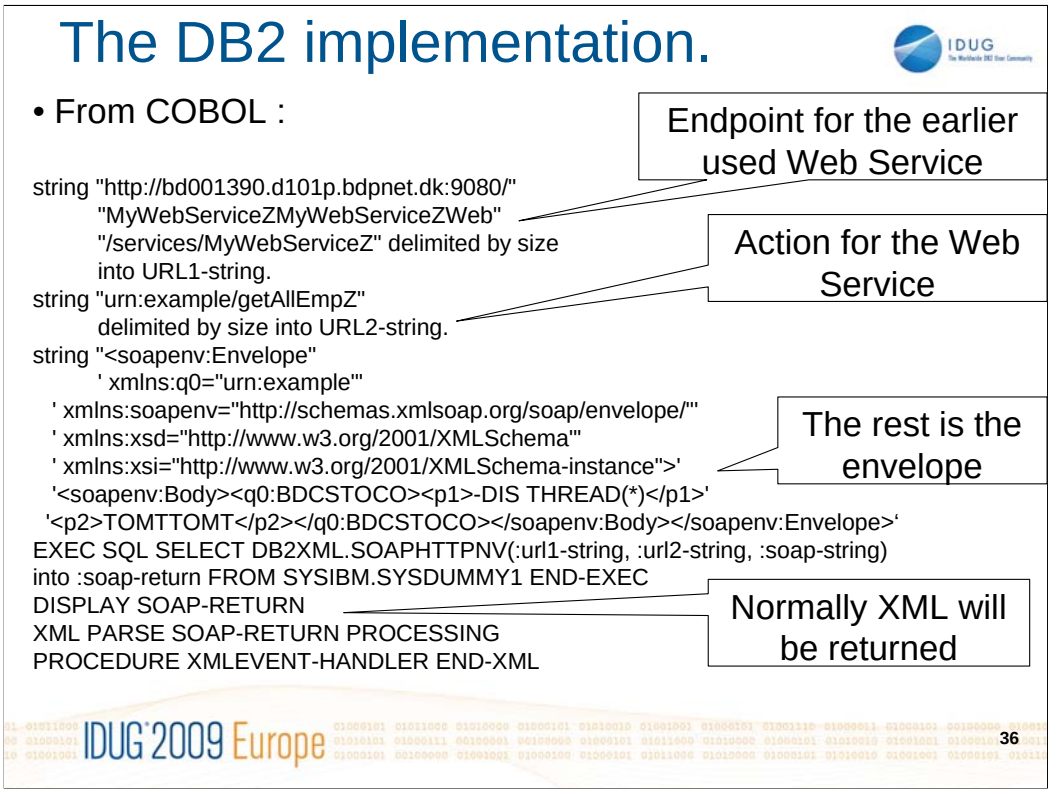
The rest is the envelope

SPUFI only returns part of the answer

```
SELECT substr(DB2XML.SOAPHTTPNC(
'http://bd001390.d101p.bdpnet.dk:9080/' concat
'MyWebServiceZMyWebServiceZWeb' concat
'/services/MyWebServiceZ',  'urn:example/getAllEmpZ ',
'<soapenv:Envelope' concat' xmlns:q0="urn:example"' concat
' xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"' concat
' xmlns:xsd="http://www.w3.org/2001/XMLSchema"' concat
' xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">' concat
      '<soapenv:Body>' concat   '<q0:BDCSTOCO>'       concat
       '<p1>-DIS THREAD(*)</p1>'   concat    '<p2></p2>'     concat
       '</q0:BDCSTOCO>'       concat    '</soapenv:Body>'      concat
       '</soapenv:Envelope>' ),320,160) FROM SYSIBM.SYSDUMMY1;
---------+---------+---------+---------+---------+---------+---------+---------+
Schema-instance"><p1>-DIS THREAD(*)</p1><p2>DSNV401I  -DB2V DISPLAY THREAD
REPOR
DSNT400I SQLCODE = 000,  SUCCESSFUL EXECUTION
```


Here we call the Web Service deployed earlier that can do DB2 commands directly from SPUFI. It looks complicated but as mentioned earlier the parameters are taken from the WSDL and are more easy to find than it looks. SPUFI will only return some part of the long answer and therefore I have done a SUBSTR to show you the DIS THREAD and the start of the answer.

The DB2 implementation.



Let's move to COBOL. It's naturally more or less the same as in SPUFI except that we get the complete answer back. When the answer comes back I will use COBOL's XML PARSER (which works extremely nice) to find the desired tags

The DB2 implementation.

- Calling external Web Service from COBOL :

```
string "http://bd001390.d101p.bdpnet.dk/frank_web_service"
       "/Service1.asmx" delimited by size into URL1-string.
string “http://tempuri.org/FindAirport”delimited by size into URL2-string.
string "<soapenv:Envelope " 'xmlns:q0="http://tempuri.org/"'
 ' xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"'
 ' xmlns:xsd="http://www.w3.org/2001/XMLSchema"'
 ' xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">'
  '<soapenv:Body>'
  '<FindAirport xmlns="http://tempuri.org/">'
  '<airportcode>‘ AIRPORT-CODE '</airportcode>' '</FindAirport>' '</soapenv:Body>'
  '</soapenv:Envelope>'
    delimited by size
    into soap-string.
EXEC SQL SELECT DB2XML.SOAPHTTPNV(:url1-string, :url2-string, :soap-string)
into :soap-return FROM SYSIBM.SYSDUMMY1 END-EXEC
DISPLAY SOAP-RETURN
XML PARSE SOAP-RETURN PROCESSING
PROCEDURE XMLEVENT-HANDLER END-XML
```

The earlier used Web Service with proxy

Action for the Web Service

The rest is the envelope

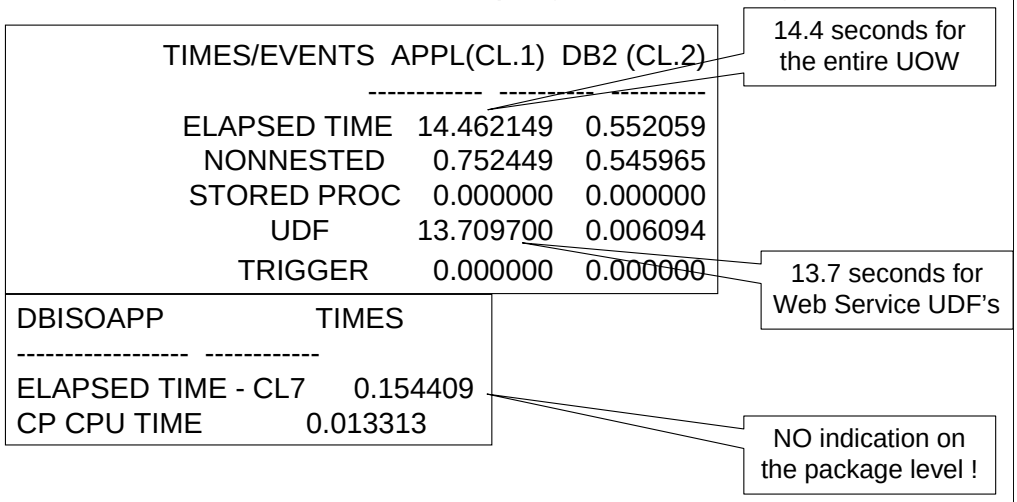
Normally we get XML back

I will now try to call an external Web Service. Because of security policies I can not exit over the network from z/OS. Therefore I use the earlier shown FindAirPort service with the proxy definitions placed on a server. In this way I can call an external Web Service from a COBOL batch program.

Pretty neat, isn’t it ??

The DB2 implementation.

- What about accounting ?
  - Can we see which COBOL programs are using the UDF's ?



When we use COBOL in CICS or batch to call Web Services we have static SQL. Naturally we are interested in how performance are and to verify the possibilities we have for seeing which package is consuming the time we will take a little look at it.

Here we have a surprise. We see above that the compete batch job tool 14.46 seconds and that the UDF's consumed 13.71 seconds. This information is on the UOW level and not at the package level. If we go down to the information about the program that is actually doing the UDF's it is indicated that this only takes 0.15 seconds in DB2. Well, technically this is correct but as a developer it would have been nice if the UDF time was accounted on the package level as well. As it is now we could have many packages doing many Web Service calls and we could not have seen which program called the most expensive Web Services. This could present a problem !!

The DB2 implementation.

- I have tried the following calls to Web Services from COBOL :
  - Web Service running on Tomcat on windows
  - Web Service running on IIS (windows)
  - Web Service running on CICS
  - Web Service running on Tomcat on AIX
  - Web Service relaying to external Web Service (proxy)
  - Web Service reading MSSQL
- Observe the following problem areas :
  - Accounting
  - Commit scope (“transaction”)
  - Invalid data (x’00’) in data stream gives non-understandable errors.

The above scenarios was verified from a COBOL program executed from IMS batch !!!!

Note that it can be difficult to understand error messages if the datastream is containing invalid characters. At some point in time, I implemented a translate of all x’00’ to a space to eliminate a part of this ‘white space’ problem.

The DB2 implementation.

- What we have learned by now :
  - We can call a Web service hosted by DB2 for z/OS from anywhere !
  - We can call any Web Service from DB2 on z/OS (read : COBOL)
  - This last bullet gives endless possibilities :
    - Including data from MSSQL directly into COBOL
    - Batch or CICS
    - Calling in-house Services written in other languages (VB)
    - Calling external Web Services (using PROXY)
    - Only imagination is the limiting factor !!
  - Now we just need to be able to call a piece of z/OS REXX
    - From any language (VB)
    - This will drive the flexibilities to new dimensions.

So now we have seen we can use DB2 as a Web Service consumer as well and call every possible Web Service, internal or external.

This means we can do anything !!! Really anything !! I have made a Web Service reading MSSQL data on a server and calling this from IMS Batch COBOL enables us to get MQSQL data synchronously into a batch program on z/OS !!!!

And even calling external services running outside in the world somewhere.

But if I want to revolutionize the tools a DBA uses I have to realize that many of these are coded in REXX. So let's finish this talk with a look at REXX as a Web Service. In this way we could build a presentation layer in VB and call the existing pieces of REXX code and use these as business logic or data access modules.

REXX on z/OS as WebServices

- Letting a piece of REXX-code turn into a Web Service.
  - Define it as a REXX Stored Procedure.
  - Promote this to a a Web Service
  - One problem is the lack of a TSO-environment, so no “ADDRESS TSO”
  - Most can however be done in other ways (like ALLOC)
  - The REXX really should return data as XML.
- But, but, but :
  - Then you will have to do the promoting on each and every piece of REXX
  - You will have to recode the existing REXX.
  - You will add development time to promote and recode every REXX.
- My invention : The generic REXX Web Service
  - Define an “Entrance REXX” as a Web Service
  - Let this REXX take a parameter like “CALL ‘SOME_REXX’ using parm”
  - When the REXX returns to the generic REXX it will simply return its return data back to the caller (the “Visual Basic-program”)
  - Lets see…..

So we want one piece of REXX code to become one Web Service. Naturally we can just define it as a REXX stored procedure and promote this as a Web Service as seen earlier.

There would however be some problems we would have to face. Firstly we will have to register the input and output parameters. We will also end up with a lot of REXX stored procedures. And the definitions, the recode and the deploy etc would take time !!!

So I thought about making a “stub” or “driver” REXX that will just pass on to every possible REXX. Then we will avoid some of these things.

There will however still be a little recode work, as WLM has no TSO environment running so all ADDRESS TSO has to be recoded.

REXX on z/OS as WebServices

- Calling a REXX on z/OS from VB :
  - We create a REXX stored procedure like this :

```
CREATE PROCEDURE UBIFAP.BDCSTOCR
(INOUT P1 VARCHAR(254) FOR SBCS DATA CCSID EBCDIC ,
INOUT P2 VARCHAR(12000) FOR SBCS DATA CCSID EBCDIC )
EXTERNAL NAME 'BDCSTOCR' LANGUAGE REXX  etc……..;
```

  - The REXX itself :

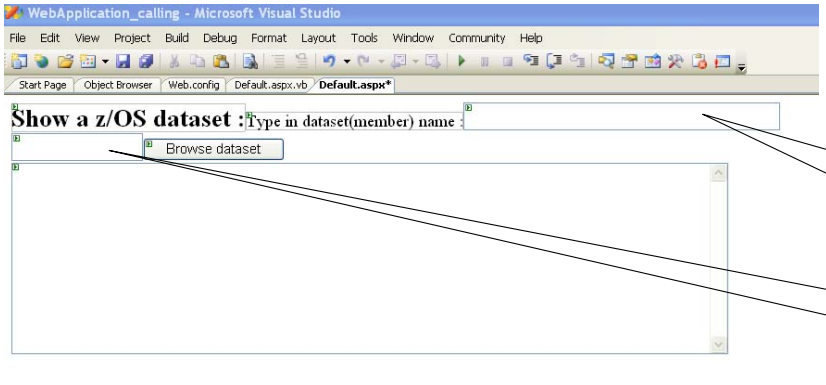
```
parse upper arg p1 p2
  say 'Received parm p1:-' !!p1!! '-' 'p2:-' !!p2!! '-'
  interpret_field = P1
  interpret "retfelt = "interpret_field !! "(" !! P2 !! ")"
  p1 = '.'
  P2 = 'retur p2'
  if retfelt = "0" then do
   do queued()
    pull retfelt
   end
  end
  p2 = retfelt
  p2.= retfelt
RETURN p1 p2.
```


I create the generic REXX that will become a Web Service.

Then I code that little REXX. Just some 12 lines of code. We will se later how it works but the idea is that the caller (the one who is calling the Web Service) will call with a parameter which is the 'real' REXX to call, together with its parameter. This is done in the INTERPRET statements in the start of the "generic" REXX above!

REXX on z/OS as WebServices

- Let's make a Web Service that will display a z/OS dataset.
- Let's show this on a browser using Visual Basic
- Let's implement this 'generic Web Service'.
- The VB-form :



Dataset name

Member name

To verify the idea I will use a small REXX that will open a z/OS dataset and return the lines in the dataset.

So I build a VB form where I can type in dataset name and member.

REXX on z/OS as WebServices

•The VB-code :

Parameter 1 is the name and parameter of the 'real' REXX to execute

```
TextBox6.Text = ""
Dim parm1 As New localhost.BDCSTOCR
Dim wfelt1 As String
TextBox6.Text = ""
parm1.P1 = "DSNBROWS " & TextBox4.Text & "(" & LTrim(RTrim(TextBox5.Text)) & ")"
parm1.P2 = Space(2000)
Dim stowebservice1 As localhost.MyNewWebService1 = New localhost.MyNewWebService1
Dim stowebresult As localhost.BDCSTOCRResponse
stowebresult = stowebservice1.BDCSTOCR(parm1)
wfelt1 = Mid(LTrim(RTrim(stowebresult.P2)), 3)
Dim scanpos As Integer = 1
While scanpos <= wfelt1.Length
   If Mid(wfelt1, scanpos, 1) = ";" Then
      wfelt1 = Mid(wfelt1, 1, scanpos - 1) & vbNewLine & Mid(wfelt1, scanpos + 1)
      scanpos = 1
   Else
      scanpos = scanpos + 1
   End If
End While
TextBox6.Text = TextBox6.Text & wfelt1
```


Here I have the code driven when the button is clicked. It will call the 'generic' REXX (BDCSTOCR) with a parameter which is the name of the REXX that will look in the dataset (DSNBROWS) and with the dataset and member as parameter. Then it will call the Web Service, which is merely the 'generic' REXX. By the previous mentioned INTERPRET commands it will call on to DSNBROWS. This will read the dataset and return the lines to the 'generic' REXX that will just return these back to the VB program.

The VB code above will take line by line and insert these to the form.

REXX on z/OS as WebServices

•The DSNBROWS-code :

```
ARG dsname
ret_parm = ""
 dsname = """ !! strip(dsname,'B') !! """
a_string = "alloc dd(FILEDD) da("dsname") shr reuse"
rx_rc = bpxwdyn(a_string)
do i =1 to s99msg.0
    say s99msg.i
end
drop lines
"execio * diskr filedd (STEM lines."
imax = 135
if lines.0 < imax then do
      imax = lines.0
end
do i =1 to imax
   ret_parm = ret_parm!!lines.i !! ";"
end
"execio 0 diskr filedd (FINIS"
rx_rc = bpxwdyn("FREE FI(FILEDD)")
return ret_parm
```

Parameter 1 is the dataset and member name.

As we can not use TSO, I used REXX services for allocation

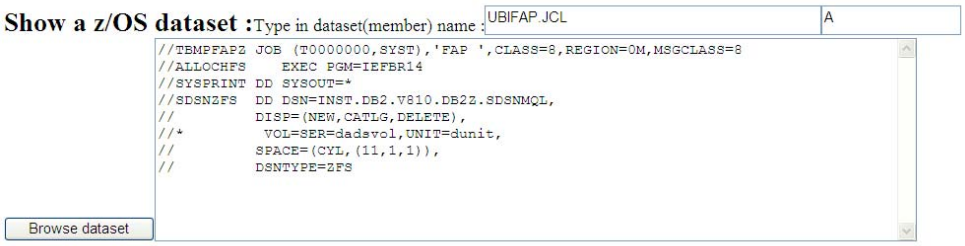
All lines from the dataset is returned to the 'generic Web Service' with a ; as separation character.

Here I have the code of DSNBROWS. See the bpxwdyn function which is a way to alloc datasets without using TSO ALLOC commands.

The last thing done, is returning the lines in the return area separated by ";"

REXX on z/OS as WebServices

- The DSNBROWS-code : The result
  - Just an example – the same can be achieved with the new DB2 V9 administration package.



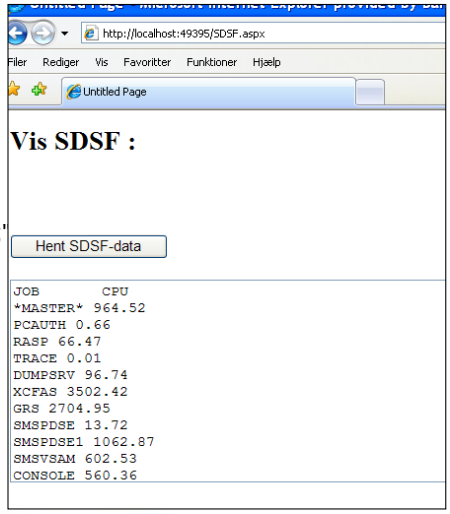
Here we have the result from the DSNBROWS Rexx. We have a z/OS dataset (member) displayed on a browser.

REXX on z/OS as WebServices

•Last example – show SDSF information on a browser using VB:

- New REXX interface for SDSF in z/OS 1.10 is fantastic !!
- See this REXX :

```
sdsfdata = ""
rc = isfcalls('ON')
isfprefix = "*"
isfowner = "*"
 address sdsf 'ISFEXEC DA'
 do ix = 1 to jname.0
    sdsfdata = sdsfdata !! jname.ix !! " " !! CPU.IX";"
 end
rc = isfcalls('OFF')
return sdsfdata
```



Here I have used the ‘generic’ REXX Web Service to make a simple WEB-driven SDSF-panel.

In z/OS 1.10 IBM has build SDSF functionality that suits the purpose perfectly !!! The code above will extract names of active tasks with the CPU time consumed and return these separated by ‘;’.

The screenshot from the browser shows the output (or the user experience).

Get the idea ??

Conclusions

- The provider implementation of Web Services is very elegant
- Does what it is supposed to – nothing more, nothing less
- The consumer implementation with UDF's is surprisingly well working.
- Accounting and statistics can be difficult to understand.
- There is a lack "2 phase commit" support in Web services.
- The idea with 'generic' REXX web services can be very useful.
- Web Services can remove the last excuse for building presentation layer in ISPF.
- As a DBA or Sys.prog it can indeed revolutionize your world.

The conclusions.

DB2 as provider is doing it's job as expected. When I first looked at the way the consumer interface is build by IBM with the UDF's, I was very skeptical but I must admit that it in fact works extremely well.

I could have wished better accounting possibilities.

The 'generic' REXX idea might change the way you develop small helping tools etc as there is NO excuse for using SPF-panels anymore !!!!

Session 17760

How to cope with soap? WebServices on DB2 for z/OS

Frank Petersen

Bankdata, Denmark

fap@bankdata.dk