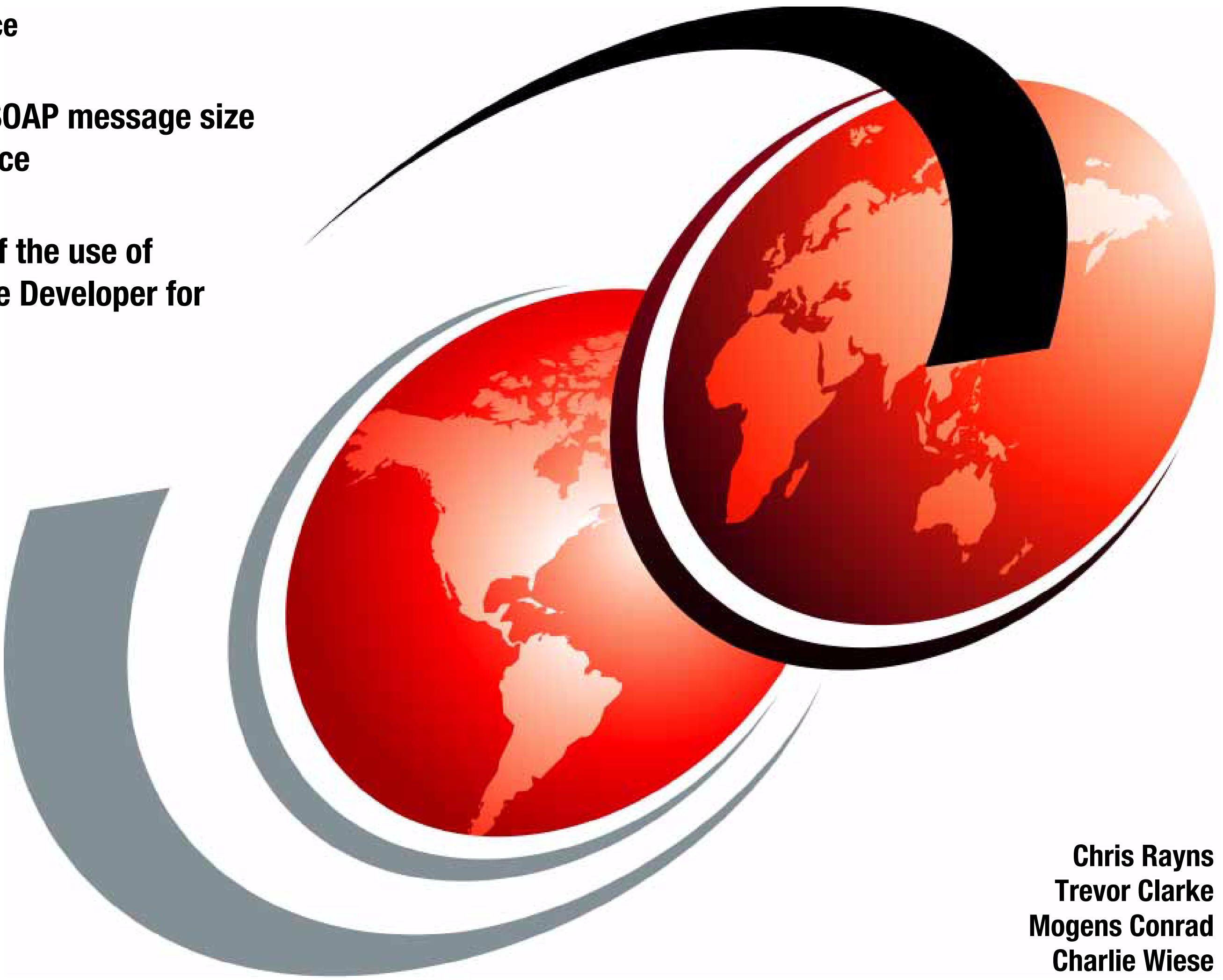
IBM

SOAP Message Size Performance Considerations

DanskeBank ATM application performance

A look at SOAP message size performance

Example of the use of WebSphere Developer for zSeries



Chris Rayns
Trevor Clarke
Mogens Conrad
Charlie Wiese

Redpaper

ibm.com/redbooks

International Technical Support Organization

SOAP Message Size Performance Considerations

August 2007

REDP-4344-00

Note: Before using this information and the product it supports, read the information in “Notices” on page v.

First Edition (August 2007)

This edition applies to Version 3, Release 1, CICS Transaction Server.

Contents

Notices

Trademarks

Preface

This IBM® Redpaper publication examines performance when using different SOAP message sizes in CICS® Web Services.

We used a preexisting basic Web Services environment with no message handlers or header handlers. Using WebSphere® Studio Workload Simulator (WSWS), we ran a number of simulations using predefined scripts. For more information about WSWS, refer to *WebSphere Studio Workload Simulator User's Guide*, SC31-6307. We then processed the CICS SMF data using CICS Performance Analyzer (CICS PA) and collated the statistics. For more information about CICSPA, see *CICS Performance Analyzer User's Guide,* SC34-6307.

This paper also reviews a customer scenario, DanskeBank. We provide an overview of the bank's ATM system, and briefly discuss the IFX framework. We detail our use of WebSphere Developer for zSeries® (WD/z) to simulate request messages from an ATM, present a performance comparison between optimized and non-optimized schema versions, and cover schema versus container definitions, large fields, and schema complexity. Finally, we provide an example of the impact of mapping levels.

The team that wrote this paper

This paper was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Chris Rayns is an IT Specialist and the CICS project leader at the International Technical Support Organization, Poughkeepsie Center. He writes extensively on all areas of CICS. Before joining the ITSO, Chris worked in IBM Global Services in the United Kingdom as a CICS IT Specialist.

Trevor Clarke is a CICS Performance Specialist in Hursley, England. He has 30 years of experience in the IT industry, working both for IBM and as a customer. For most of this period he has worked with CICS in technical support, application development, and performance roles, and also as an educator. His areas of expertise also include DB2 and networking. He holds a BSC Mathematics degree from Southampton University.

Mogens Conrad is an IT Architect working in Strategy and Design Authority at IBM Service Delivery Denmark. He has 30 years of experience in the computer industry, working with DB2®, CICS, Application design, Business Continuity and Intrastructure. Mogens is a member of CICS Architectural Forum.

Charlie Wiese is a CICS Level 2 Software Engineer in the United States. He has 23 years of experience with CICS working for customers and IBM, and has been with Level 2 Support for the last three years. His primary focus areas include CICS Web Support, Web Services, and Storage Management. He holds a Bachelor of Science degree in Business Administration from Southern Oregon University.

Acknowledgments

Thanks to the following people for their contributions to this project:

Richard M Conway
Mike Connolly
International Technical Support Organization, Poughkeepsie Center

John Burgess, Performance Software Development Engineer
Phil Hanson, CICS Product Manager
IBM Hursley

Steven Webb, CICS Level 2 Support
IBM Raleigh

The team that wrote the IBM Redbooks® publication *Implementing CICS Web Services,* SG24-7206: Nigel Williams, Grant Ward Able, Paolo Chieregatti, Robert Herman, Tommy Joergensen, Luis Aused Lopez, and Steve Wall.

1

SOAP message size performance considerations

This chapter discusses how SOAP message size and complexity affect performance.

1.1 Environment

For these performance tests we used a pre-existing basic Web Services environment with no message handlers or header handlers.

Using WebSphere Studio Workload Simulator (WSWS), we ran a number of simulations, using predefined scripts (see “CICS Web Services assistant” on page 6). For more information about WSWS, refer to *WebSphere Studio Workload Simulator User’s Guide*, SC31-6307.

We then processed the CICS SMF data using CICS Performance Analyzer (CICS PA) and collated the statistics (see “Processing the results” on page 10). For more information about CICSPA, see *CICS Performance Analyzer User’s Guide,* SC34-6307.

Finally, we presented the statistics in report and graphical format. We discuss the findings in “The results” on page 19.

1.2 Preparing our tests

We ran 13 tests, each with a different SOAP message size and complexity. We followed these steps to set up our test runs:

1. Generate the WSBIND and WSDL files.
    a. Create HFS directories in which to store the generated files. We used /ciws/sc32/wsbind and /ciws/sc32/shelf.

       Figure 1-1 shows a listing of our HFS directory.

```
Type  Filename
_ Dir   .
_ Dir   ..
_ Dir   config
_ File  run01
_ File  run03
_ File  run04
_ File  run06
_ File  run08
_ File  run09
_ File  run11
_ File  run30
_ File  run31
_ File  run32
_ File  run33
_ File  run34
_ File  run35
_ Dir   shelf
_ Dir   wsbind
```

Figure 1-1 HFS directory list

    b. Create a TCPIPSERVICE resource definition.

       Figure 1-2 shows our RDO definition parameters.

```
CEDA  View TCpipservice( S3C2     )
 TCpipservice   : S3C2
 GROup          : S3C2EXWS
 DEscription    : CICS Example Application - TCPIPSERVICE
 Urm            : DFHWBAAX
 POrtnumber     : 14302             1-65535
 STatus         : Open              Open | Closed
 PROtocol       : Http              Iiop | Http | Eci | User
 TRansaction    : CWXN
 Backlog        : 00005             0-32767
 TSqprefix      :
 Ipaddress      : 10.1.6.17
 SOcketclose    : No                No | 0-240000 (HHMMSS)
 Maxdatalen     : 000040            3-524288
SECURITY
 SSl            : No                Yes | No | Clientauth
 CErtificate    :
 (Mixed Case)
PRIvacy        :                  Notsupported | Required |
Supported
 CIphers       :
 AUthenticate   : No                No | Basic | Certificate |
AUTORegister
                                | AUTOMatic | ASserted
 ATtachsec     :                 Local | Verify
DNS CONNECTION BALANCING
 DNsgroup      :
 GRPcritical    : No                No | Yes
```

Figure 1-2 TCPIPSERVICE RDO definition

Figure 1-3 shows the result when displayed using CEMT.

```
I TCPIPSERVICE(S*)
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Tcpips(S3C2    ) Ope Por(14302) Http Nos Tra(CWXN)
    Con(00001) Bac( 00005 ) Max( 000040 ) Urm( DFHWBAAX )
```

Figure 1-3 CEMT IT CPIPSERVICE

2. Create a PIPELINE resource definition.

   a. Create a service provider pipeline configuration file.

      A *pipeline configuration* file is an XML file that describes, among other things, the message handler programs and the SOAP header processing programs that CICS will invoke when it processes the pipeline.

      Figure 1-4 shows our RDO definition parameters.

```
OBJECT CHARACTERISTICS                                          CICS
RELEASE = 0640
 CEDA  View PIpeline( PIPE3    )
  PIpeline      : PIPE3
  Group         : S3C2EXWS
  Description    :
  STatus        : Enabled          Enabled | Disabled
   Configfile     :
/CIWS/S3C2/config/ITSO_7206_basicsoap11provider.xml
  (Mixed Case)   :
                :
                :
                :
  SHelf         : /CIWS/S3C2/shelf
  (Mixed Case)   :
                :
                :
                :
  Wsdir         : /CIWS/S3C2/wsbind
   (Mixed Case)   :
```

Figure 1-4 PIPELINE RDO definition

Figure 1-5 shows the result when displayed using CEMT.

```
I PIPE(PIPE3)
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Pip(PIPE3   ) Ena Con(/CIWS/S3C2/config/ITSO_720)
    She(/CIWS/S3C2/shelf/         ) Wsd(/CIWS/S3C2/wsbind/        )
```

Figure 1-5 CEMT I PIPE

b. Create an HFS directory in which to store installable WSBIND and WSDL files.

   This is the *pickup* directory, because CICS will pick up the WSBIND and WSDL files from this directory and store them on a *shelf* directory.

c. Create an HFS directory for CICS to store installed WSBIND files.

   This is the shelf directory.

d. Create a PIPELINE resource definition to handle the Web Service request.

   - Specify the CONFIGFILE attribute to point to the file created in step 2a.
   - Specify the WSDIR attribute to point to the directory created in step 2b.
   - Specify the SHELF attribute to point to the directory created in step 2c.

3. Install the TCPIPSERVICE and PIPELINE resource definitions.

   After the PIPELINE definition is installed, CICS scans the pickup directory for WSBIND files. When CICS finds the WSBIND file created in step 1, CICS dynamically creates and installs a WEBSERVICE resource definition for it.

   CICS derives the name of the WEBSERVICE definition from the name of the WSBIND file. The WEBSERVICE definition identifies the name of the associated PIPELINE definition and points to the location of the WSBIND file in the HFS.

During the installation of the WEBSERVICE resource, the following occurs:

- CICS dynamically creates and installs a URIMAP resource definition. CICS bases the definition on the URI specified in the input to DFHLS2WS in step 1, and stored by DFHLS2WS in the WSBIND file.
- CICS uses the WSBIND file to create main storage control blocks to map the inbound service request (XML) to a COMMAREA or a container, and to map to XML the outbound COMMAREA or container that contains the response data.

In our test scenarios CICS is a service provider, and the following processing flow occurs:

1. The CICS-supplied sockets listener transaction (CSOL) monitors the port specified in the TCPIPSERVICE resource definition for incoming HTTP requests. When the SOAP message arrives, CSOL attaches the CICS-supplied Web attach transaction CWXN.
2. CWXN finds the URI in the HTTP request and then scans the URIMAP resource definitions for a URIMAP that has its USAGE attribute set to PIPELINE and its PATH attribute set to the URI found in the HTTP request. If CWXN finds such a URIMAP, its uses the PIPELINE and WEBSERVICE attributes of the URIMAP definition to get the name of the PIPELINE and WEBSERVICE definitions that it will use to process the incoming request.

   CWXN also uses the TRANSACTION attribute of the URIMAP definition to determine the name of the transaction that it should attach to process the pipeline; normally, this will be the CPIH transaction.
3. CPIH starts the pipeline processing. It uses the PIPELINE definition to find the name of the pipeline configuration file. CPIH uses the pipeline configuration file to determine which message handler programs and SOAP header processing programs to invoke.
4. A message handler in the pipeline (typically, a CICS-supplied SOAP message handler) removes the SOAP envelope from the inbound request and passes the SOAP body to the data mapper function.
5. CICS uses the DFHWS-WEBSERVICE container to pass the name of the required WEBSERVICE definition to the data mapper. The data mapper uses the WEBSERVICE definition to locate the main storage control blocks that it needs to map the inbound service request (XML) to a COMMAREA or a container.
6. The data mapper links to the target service provider application program, providing input in the format that it expects. The application program is not aware that it is being executed as a Web Service. The program performs its normal processing, then returns an output COMMAREA or container to the data mapper.

   Note that the output data from the CICS application program cannot just be sent back to the pipeline code. The data mapper must first convert the output from the COMMAREA or container format into a SOAP body.

1.2.1 Back-end program

A back-end program was used as an application provider. It simply placed a dummy value (all 9s) in the COMMAREA and returned control to CICS.

Example 1-1 shows the source for BUSLOG02.

Example 1-1 The BUSLOG02 program

```
ID DIVISION.
       PROGRAM-ID.  BUSLOG02.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
```

```
LINKAGE SECTION.
01 DFHCOMMAREA.
   02 FILLER     PIC X OCCURS 32500 DEPENDING ON EIBCALEN.
PROCEDURE DIVISION.
MAIN-PROCESSING SECTION.
    MOVE ALL '9' TO DFHCOMMAREA(1:EIBCALEN)
    EXEC CICS RETURN END-EXEC.
```

1.2.2 CICS Web Services assistant

Prior to each test, we ran the CICS Web Services assistant utility program DFHLS2WS to create the wsbind file.

DFHLS2WS

The CICS Web Services assistant is a set of batch utilities that can help you to transform existing CICS applications into Web Services, and enable CICS applications to use Web Services provided by external providers.

The Web Services assistant helps you to deploy an application with the least amount of programming effort. For example, if you want to expose an existing application as a Web Service, you can start with a high level language data structure and use DFHLS2WS to generate the Web Services description.

Alternatively, if you want to communicate with an existing Web Service, you can start with its Web Service description as defined in its WSDL file and use DFHWS2LS to generate a high level language structure that you can use in your program.

Both DFHLS2WS and DFHWS2LS generate a file called wsbind. When your application runs, CICS uses the wsbind file to transform your application data into a SOAP message on output, and to transform the SOAP message to application data on input.

In this chapter we use DFHLS2WS to generate our wsbind file for each test run.

Figure 1-6 shows an example of the DFHLS2WS job we used.

```
//DFHLS2WS JOB 1013DBDC,'CICS LS 2 WS',MSGLEVEL=(1,1),
//            CLASS=A,NOTIFY=&SYSUID,MSGCLASS=H,REGION=0M
//*****************************************************
//* LANGUAGE STRUCTURE TO WSDL CONVERSION ROUTINE
//*****************************************************
//  SET QT=''''
//LS2WS  EXEC DFHLS2WS,PATHPREF='',
//
TMPDIR='/u/cicsrs8',USSDIR='cicsts31',TMPFILE=&QT.&SYSUID.&QT
//INPUT.SYSUT1 DD *
 PDSLIB=//CIWS.CICS.SOURCE
 LANG=COBOL
 PGMNAME=BUSLOG02
 REQMEM=DATA01
 RESPMEM=DATA01
 URI=cicsws/soap11/run01
 PGMINT=COMMAREA
 MAPPING-LEVEL=1.2
 WSBIND=/CIWS/S3C2/wsbind/run01.wsbind
 WSDL=/CIWS/S3C2/wsdl/run01.wsdl
//
```

Figure 1-6 DFHLS2WS job

The main input parameters are as follows:

- PDSLIB

  This specifies the name of the partitioned data set that contains the high level language data structures to be processed.

- LANG

  This specifies the language of the language structure to be created.

- PGMNAME

  This specifies the name of the target CICS application program that is being exposed as a Web Service. In our tests this is the BUSLOG02 dummy back-end program; refer to "Back-end program" on page 5 for more information about this topic.

- REQMEM

  This specifies the name of the partitioned data set member that contains the high level language structure for the Web Service request.

  – For a service provider, the Web Service request is the input to the application program.
  – For a service requester, the Web Service request is the output from the application program.

  Refer to Table 1-2 on page 9 for a list of the input data and output data for each test run.

- RESPMEM

  This specifies the name of the partitioned data set member that contains the high level language structure for the Web Service response:

  – For a service provider, the Web Service response is the output from the application program.
  – For a service requester, the Web Service response is the input to the application program.

Refer to Table 1-2 on page 9 for a list of the input data and output data for each test run.

- URI

  In a service provider, this parameter specifies the relative URI that a client will use to access the Web Service. CICS uses the value specified when it generates a URIMAP resource from the Web Service binding file created by DFHLS2WS. The parameter specifies the path component of the URI to which the URIMAP definition applies.

- PGMINT

  For a service provider, this parameter specifies how CICS passes data to the target application program (using a COMMAREA or a channel).

- WSBIND

  This specifies the HFS name of the Web Service binding file.

- WSDL

  This specifies the HFS name of the Web Service description file.

For more information about DFHLS2WS, see *CICS Transaction Server for z/OS Web Services Guide*, SC34-6458, and the IBM Redbooks publication *Implementing CICS Web Services,* SG24-7206.

1.2.3 The test scripts

As mentioned, Table 1-1 on page 8 shows the input and output data for each test run.

Table 1-1 Inbound and outbound data size for test runs

	Inbound data	**Outbound data**
Run01	1 byte field name with 1 byte data element	1 byte field name with 1 byte data element
Run02	1 byte field name with 4 K bytes data element	1 byte field name with 1 byte data element
Run03	1 byte field name with 8 K bytes data element	1 byte field name with 1 byte data element
Run04	1 byte field name with 32 K bytes data element	1 byte field name with 1 byte data element
Run05	1 byte field name with 1 byte data element	1 byte field name with 4 K byte data element
Run06	1 byte field name with 1 byte data element	1 byte field name with 8 K byte data element
Run07	1 byte field name with 1 byte data element	1 byte field name with 32 K bytes data element
Run08	1000 x 3 byte field names each with 1 byte data element	1 byte field name with 1 byte data element
Run9	2000 x 3 byte field names each with 1 byte data element	1 byte field name with 1 byte data element

	Inbound data	Outbound data
Run10	3000 x 3 byte field names each with 1 byte data element	1 byte field name with 1 byte data element
Run11	1 byte field name with 1 byte data element	1000 x 3 byte field names each with 1 byte data element
Run12	1 byte field name with 1 byte data element	2000 x 3 byte field names each with 1 byte data element
Run13	1 byte field name with 1 byte data element	3000 x 3 byte field names each with 1 byte data element

Table 1-2 shows the partitioned data set member that contains the high level language structure for the Web Service request, specified on the RESPMEM and REQMEM parameters of the DFHLS2WS job.

Table 1-2 Input to DFHLS2WS

REQMEM/RESPMEM member	Data contents
DATA01	02 X PIC X
DATA4K	02 X PIC X(4096)
DATA8K	02 X PIC X(8192)
DATA325	02 X PIC X(32500)
DATA1K	1000 iterations of 02 *nnn* PIC X(1)
DATA2K	2000 iterations of 02 *nnn* PIC X(1)
DATA3K	3000 iterations of 02 *nnn* PIC X(1)

Figure 1-7 shows an example of the SOAP message data we pass in test run 02.

```
"<?xml version='1.0' encoding='UTF-8'?>" + CRLF + "<SOAP-ENV:Envelope
xmlns:SOAP-ENV=\"http://schemas.xmlsoap.org/soap/envelope/\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\">" + CRLF + "<SOAP-ENV:Body>" + CRLF +
"<BUSLOGO2Operation><x>AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJ
JJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBB
BBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCD
DDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEE
EFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGG
GGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIII
IIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAA
AAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBC
CCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCAAAAAAAAAABBBBB
BBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDD
DDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEF
FFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGG
GHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIII
IIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAA
AAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCC
CCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDE
EEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFF
FGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFF
FFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGH
HHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIII
IJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAA
AAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCC
CCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEE
EEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFG
GGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHH
HIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJ
JJJAAAAAAAAAABBBBBBBBBBCCCAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJ
JJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAA
ABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCC
CCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEE
EEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGG
GGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHI
IIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJ
JAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBB
BBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJAAAAAAAAAABBBBBBBBBBCCCC</x></BUSLO
GO2Operation>" + CRLF + "</SOAP-ENV:Body>" + CRLF + "</SOAP-ENV:Envelope>" + CRLF,
    "Host: 10.1.6.17",
    "Content-Type: text/xml; charset=utf-8",
    "Content-Length: 4420",
    "Accept: application/soap+xml, application/dime, multipart/related, text/*",
    "User-Agent: IBM Web Services Explorer",
    "Cache-Control: no-cache",
    "Pragma: no-cache",
    "SOAPAction: \"\""
```

Figure 1-7 SOAP data passed in Run02

1.2.4 Processing the results

To process the SMF data, we used CICS PA. The transactions we are interested in are CPIH and CWXN. Example 1-2 shows the JCL.

Example 1-2 CICS PA JCL

```
//CICSRS4A JOB MSGCLASS=H,NOTIFY=CICSRS4
//RUN001   EXEC PGM=CPAMAIN,REGION=4M,PARM=NOSTAE
//STEPLIB  DD  DSN=CPA.V1R4.SCPALINK,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SMFIN001 DD  DISP=SHR,DSN=CICSRS4.RUNXXX.SMF281
//CPAXW001 DD DISP=(NEW,DELETE),UNIT=SYSDA,SPACE=(CYL,(200,50)),
//            VOL=(,,,30)
//CPASWK01 DD DISP=(NEW,DELETE),UNIT=VIO,SPACE=(CYL,(90,50))
//CPASWK02 DD DISP=(NEW,DELETE),UNIT=VIO,SPACE=(CYL,(90,50))
//CPASWK03 DD DISP=(NEW,DELETE),UNIT=VIO,SPACE=(CYL,(90,50))
//CPASWK04 DD DISP=(NEW,DELETE),UNIT=VIO,SPACE=(CYL,(90,50))
//SYSOUT   DD SYSOUT=*
```

```
//SYSIN    DD  *
         CICSPA IN(SMFIN001),
                SMFSTART(08:00:00),
                SMFSTOP(23:00:00),
            SELECT(PERFORMANCE(INCLUDE(APPLID(A6POS3C2),
                   TRAN(CPIH,CWXN,)))),
            SUMMARY(OUTPUT(PSUM0001),
            INTERVAL(1:00),
                BY(STOP,TRAN),
                FIELDS(STOP,TRAN,
                      TASKCNT,
                      RESPONSE,
                      CPU(TIME(AVE))),
                EXTERNAL(CPAXW001))
```

1.3 Running the tests

We ran 13 separate tests, running 100 tasks through WSWS with zero think time.

Run01

Figure 1-8 shows our DFHLS2WS input parameters for Run01.

```
PDSLIB=//CIWS.CICS.SOURCE
LANG=COBOL
PGMNAME=BUSLOG02
REQMEM=DATA01
RESPMEM=DATA01
URI=cicsws/soap11/run01
PGMINT=COMMAREA
MAPPING-LEVEL=1.2
WSBIND=/CIWS/S3C2/wsbind/run01.wsbind
WSDL=/CIWS/S3C2/wsdl/run01.wsdl
```

Figure 1-8 DFHLS2WS input parameters - Run01

Figure 1-9 shows the CICS PA results for Run01.

```
 Avg      Avg
Stop     Tran       #Tasks User CPU Response
Interval                      Time     Time
11:25:00 CPIH          100    .0009    .0013
11:25:00 CWXN          100    .0003    .0039
11:25:00               200    .0006    .0026
```

Figure 1-9 CICS PA results for Run01

Run02

Figure 1-10 shows our DFHLS2WS input parameters for Run02.

```
PDSLIB=//CIWS.CICS.SOURCE
LANG=COBOL
PGMNAME=BUSLOG02
REQMEM=DATA4K
RESPMEM=DATA01
URI=cicsws/soap11/run02
PGMINT=COMMAREA
MAPPING-LEVEL=1.2
WSBIND=/CIWS/S3C2/wsbind/run02.wsbind
WSDL=/CIWS/S3C2/wsdl/run02.wsdl
```

Figure 1-10 DFHLS2WS input parameters - Run02

Figure 1-11 shows the CICS PA results for Run02.

```
 Avg      Avg
Stop     Tran       #Tasks User CPU Response
Interval                       Time     Time
11:28:00 CPIH          100    .0010    .0014
11:28:00 CWXN          100    .0004    .0007
11:28:00               200    .0007    .0010
```

Figure 1-11 CICS PA results for Run02

Run03

Figure 1-12 shows our DFHLS2WS input parameters for Run03.

```
PDSLIB=//CIWS.CICS.SOURCE
LANG=COBOL
PGMNAME=BUSLOG02
REQMEM=DATA8K
RESPMEM=DATA01
URI=cicsws/soap11/run03
PGMINT=COMMAREA
MAPPING-LEVEL=1.2
WSBIND=/CIWS/S3C2/wsbind/run03.wsbind
WSDL=/CIWS/S3C2/wsdl/run03fieldfield.wsdl
```

Figure 1-12 DFHLS2WS input parameters - Run03

Figure 1-13 shows the CICS PA results for Run03.

```
 Avg      Avg
Stop     Tran       #Tasks User CPU Response
Interval                       Time     Time
11:31:00 CPIH          100    .0011    .0016
11:31:00 CWXN          100    .0004    .0016
11:31:00               200    .0008    .0016
```

Figure 1-13 CICS PA results for Run03

Run04

Figure 1-14 shows our DFHLS2WS input parameters for Run04.

```
PDSLIB=//CIWS.CICS.SOURCE
LANG=COBOL
PGMNAME=BUSLOG02
REQMEM=DATA325
RESPMEM=DATA01
URI=cicsws/soap11/run04
PGMINT=COMMAREA
MAPPING-LEVEL=1.2
WSBIND=/CIWS/S3C2/wsbind/run04.wsbind
WSDL=/CIWS/S3C2/wsdl/run04.wsdl
```

Figure 1-14 DFHLS2WS input parameters - Run04

Figure 1-15 shows the CICS PA results for Run04.

```
 Avg      Avg
Stop     Tran       #Tasks User CPU Response
Interval                      Time     Time
11:33:00 CPIH          100    .0019    .0024
11:33:00 CWXN          100    .0007    .0075
11:33:00               200    .0013    .0049
```

Figure 1-15 CICS PA results for Run04

Run05

Figure 1-16 shows our DFHLS2WS input parameters for Run05.

```
PDSLIB=//CIWS.CICS.SOURCE
LANG=COBOL
PGMNAME=BUSLOG02
REQMEM=DATA01
RESPMEM=DATA4K
URI=cicsws/soap11/run05
PGMINT=COMMAREA
MAPPING-LEVEL=1.2
WSBIND=/CIWS/S3C2/wsbind/run05.wsbind
WSDL=/CIWS/S3C2/wsdl/run05.wsdl
```

Figure 1-16 DFHLS2WS input parameters - Run05

Figure 1-17 shows the CICS PA results for Run05.

```
 Avg      Avg
Stop     Tran       #Tasks User CPU Response
Interval                      Time     Time
11:35:00 CPIH          100    .0007    .0011
11:35:00 CWXN          100    .0002    .0040
11:35:00               200    .0004    .0025
```

Figure 1-17 CICS PA results for Run05

Run06

Figure 1-18 shows our DFHLS2WS input parameters for Run06.

```
PDSLIB=//CIWS.CICS.SOURCE
LANG=COBOL
PGMNAME=BUSLOG02
REQMEM=DATA01
RESPMEM=DATA8K
URI=cicsws/soap11/run06
PGMINT=COMMAREA
MAPPING-LEVEL=1.2
WSBIND=/CIWS/S3C2/wsbind/run06.wsbind
WSDL=/CIWS/S3C2/wsdl/run06.wsdl
```

Figure 1-18 DFHLS2WS input parameters - Run06

Figure 1-19 shows the CICS PA results for Runo6.

```
 Avg      Avg
Stop     Tran       #Tasks User CPU Response
Interval                       Time     Time
11:37:00 CPIH          100    .0009    .0012
11:37:00 CWXN          100    .0001    .0040
11:37:00               200    .0005    .0026
```

Figure 1-19 CICS PA results for Run06

Run07

Figure 1-20 shows our DFHLS2WS input parameters for Run07.

```
PDSLIB=//CIWS.CICS.SOURCE
LANG=COBOL
PGMNAME=BUSLOG02
REQMEM=DATA01
RESPMEM=DATA325
URI=cicsws/soap11/run07
PGMINT=COMMAREA
MAPPING-LEVEL=1.2
WSBIND=/CIWS/S3C2/wsbind/run07.wsbind
WSDL=/CIWS/S3C2/wsdl/run07.wsdl
```

Figure 1-20 DFHLS2WS input parameters - Run07

Figure 1-21 shows the CICS PA results for Run07.

```
 Avg      Avg
Stop     Tran       #Tasks User CPU Response
Interval                       Time     Time
11:39:00 CPIH          100    .0016    .0064
11:39:00 CWXN          100    .0002    .0040
11:39:00               200    .0009    .0052
```

Figure 1-21 CICS PA results for Run07

Run08

Figure 1-22 shows our DFHLS2WS input parameters for Run08.

```
PDSLIB=//CIWS.CICS.SOURCE
LANG=COBOL
PGMNAME=BUSLOG02
REQMEM=DATA1K
RESPMEM=DATA01
URI=cicsws/soap11/run08
PGMINT=COMMAREA
MAPPING-LEVEL=1.2
WSBIND=/CIWS/S3C2/wsbind/run08.wsbind
WSDL=/CIWS/S3C2/wsdl/run08.wsdl
```

Figure 1-22 DFHLS2WS input parameters - Run08

Figure 1-23 shows the CICS PA results for Run08.

```
 Avg      Avg
Stop     Tran       #Tasks User CPU Response
Interval                      Time     Time
11:41:00 CPIH          100    .0031    .0037
11:41:00 CWXN          100    .0005    .0021
11:41:00               200    .0018    .0029
```

Figure 1-23 CICS PA results for Run08

Run09

Figure 1-24 shows our DFHLS2WS input parameters for Run09.

```
PDSLIB=//CIWS.CICS.SOURCE
LANG=COBOL
PGMNAME=BUSLOG02
REQMEM=DATA2K
RESPMEM=DATA01
URI=cicsws/soap11/run09
PGMINT=COMMAREA
MAPPING-LEVEL=1.2
WSBIND=/CIWS/S3C2/wsbind/run09.wsbind
WSDL=/CIWS/S3C2/wsdl/run09.wsdl
```

Figure 1-24 DFHLS2WS input parameters - Run09

Figure 1-25 shows the CICS PA results for Run09.

```
 Avg      Avg
Stop     Tran       #Tasks User CPU Response
Interval                      Time     Time
11:44:00 CPIH          100    .0054    .0062
11:44:00 CWXN          100    .0006    .0053
11:44:00               200    .0030    .0058
```

Figure 1-25 CICS PA results for Run09

Run10

Figure 1-26 shows our DFHLS2WS input parameters for Run10.

```
PDSLIB=//CIWS.CICS.SOURCE
LANG=COBOL
PGMNAME=BUSLOG02
REQMEM=DATA3K
RESPMEM=DATA01
URI=cicsws/soap11/run10
PGMINT=COMMAREA
MAPPING-LEVEL=1.2
WSBIND=/CIWS/S3C2/wsbind/run10.wsbind
WSDL=/CIWS/S3C2/wsdl/run10.wsdl
```

Figure 1-26 DFHLS2WS input parameters - Run10

Figure 1-27 shows the CICS PA results for Run10.

```
 Avg      Avg
Stop     Tran       #Tasks User CPU Response
Interval                      Time     Time
11:46:00 CPIH          100    .0078    .0090
11:46:00 CWXN          100    .0007    .0093
11:46:00               200    .0043    .0092
```

Figure 1-27 CICS PA results for Run10

Run11

Figure 1-28 shows our DFHLS2WS input parameters for Run11.

```
PDSLIB=//CIWS.CICS.SOURCE
LANG=COBOL
PGMNAME=BUSLOG02
REQMEM=DATA01
RESPMEM=DATA1K
LOGFILE=/CIWS/S3C2/run11
URI=cicsws/soap11/run11
PGMINT=COMMAREA
MAPPING-LEVEL=1.2
WSBIND=/CIWS/S3C2/wsbind/run11.wsbind
WSDL=/CIWS/S3C2/wsdl/run33.wsdl
```

Figure 1-28 DFHLS2WS input parameters - Run11

Figure 1-29 shows the CICS PA results for Run11.

```
 Avg      Avg
Stop     Tran       #Tasks User CPU Response
Interval                      Time     Time
11:49:00 CPIH          100    .0018 .0016
11:49:00 CWXN          100    .0002    .0042
11:49:00               200    .0010 .0029
```

Figure 1-29   CICS PA results for Run11

Run12

Figure 1-30 shows our DFHLS2WS input parameters for Run12.

```
PDSLIB=//CIWS.CICS.SOURCE
LANG=COBOL
PGMNAME=BUSLOGO2
REQMEM=DATA01
RESPMEM=DATA2K
URI=cicsws/soap11/run12
PGMINT=COMMAREA
MAPPING-LEVEL=1.2
WSBIND=/CIWS/S3C2/wsbind/run12.wsbind
WSDL=/CIWS/S3C2/wsdl/run12.wsdl
```

Figure 1-30   DFHLS2WS input parameters - Run12

Figure 1-31 shows the CICS PA results for Run12.

```
 Avg      Avg
Stop     Tran       #Tasks User CPU Response
Interval                      Time     Time
11:52:00 CPIH          100    .0027 .0049
11:52:00 CWXN          100    .0002    .0040
11:52:00               200    .0014 .0045
```

Figure 1-31   CICS PA results for Run12

Run13

Figure 1-32 shows our DFHLS2WS input parameters for Run13.

```
PDSLIB=//CIWS.CICS.SOURCE
LANG=COBOL
PGMNAME=BUSLOGO2
REQMEM=DATA01
RESPMEM=DATA3K
URI=cicsws/soap11/run13
PGMINT=COMMAREA
MAPPING-LEVEL=1.2
WSBIND=/CIWS/S3C2/wsbind/run13.wsbind
WSDL=/CIWS/S3C2/wsdl/run13.wsdl
```

Figure 1-32   DFHLS2WS input parameters - Run13

Figure 1-33 shows the CICS PA results for Run13.

```
 Avg      Avg
Stop     Tran       #Tasks User CPU Response
Interval                      Time     Time
11:54:00 CPIH          100    .0036    .0085
11:54:00 CWXN          100    .0002    .0042
11:54:00               199    .0019    .0064
```

Figure 1-33 CICS PA results for Run13

1.4 The results

The CPU usage per Web Service (CWXN+CPIH) can be plotted against inbound and outbound message sizes, as shown in Figure 1-34.

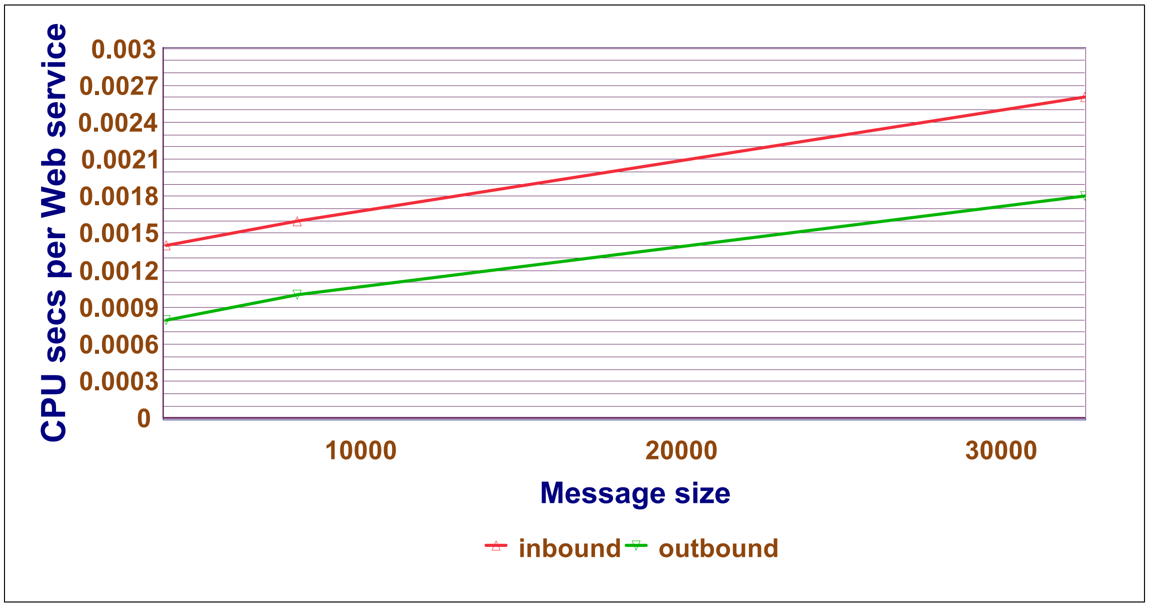


Figure 1-34 Results for message size

The CPU usage per Web Service (CWXN+CPIH) can also be plotted against the number of inbound and outbound XML elements, as shown in Figure 1-35.

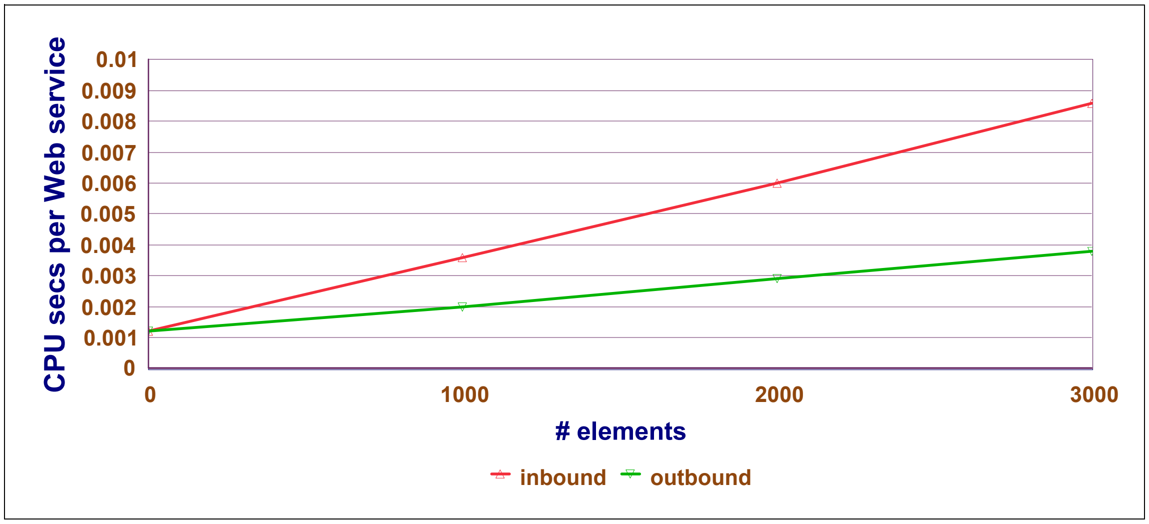


Figure 1-35 Results for number of elements

1.5 Summary

The CPU consumed by the parsing process is affected by both the overall size of the message, and also by the number of XML elements within the message.

Although only a few measurements were taken, it is possible from the graphs above to:

1. Estimate the CPU overhead for every 1K of inbound message data as 0.05 milliseconds
2. Estimate the CPU overhead for every 1K of outbound message data as 0.04 milliseconds
3. Estimate the CPU overhead for every 100 inbound XML elements as 0.24 milliseconds
4. Estimate the CPU overhead for every 100 outbound XML elements as 0.08 milliseconds

2

DanskeBank ATM application

DanskeBank is upgrading and modernizing its current system for automated teller machines (ATMs). The ATM machines is being upgraded from IBM OS/2® to Windows® XP. The communication between the ATMs and the host is changed from SNA to IP, and the communication protocol is changed to be based on the Interactive Financial eXchange™ specification (IFX) and Web Services/SOAP.

We use the new system as an example of Web Services from the “real” world. The system was not yet in production at the time of writing, but the recommendations made here are planned to be implemented before large scale production.

For the purpose of this book, the DanskeBank application is simplified, as described in this chapter, to highlight the use of Web Services. We also provide here an overview of the bank’s ATM system, and briefly discuss the IFX framework. We detail our use of WebSphere Developer for zSeries (WD/z) to simulate request messages from an ATM. Next, we present a performance analysis between optimized and non-optimized schema versions, and cover schema versus container definitions, large fields, and schema complexity. Finally, we provide an example of the impact of mapping levels.

2.1 Overview of the DanskeBank ATM system

The DanskeBank ATM application runs under Windows and is to be implemented on a large number of teller machines throughout the countries where DanskeBank is located. The Windows application controls the different drivers in the teller machine, printer, dispenser, and card reader. The application also communicates with the host system when authenticating card and pin codes, and when running requests such as withdrawals or account listings.

The ATM application in its first version consisted of 25 request types, some of which were customer-related and others that were technical. Figure 2-1 provides an overview of the ATM system.

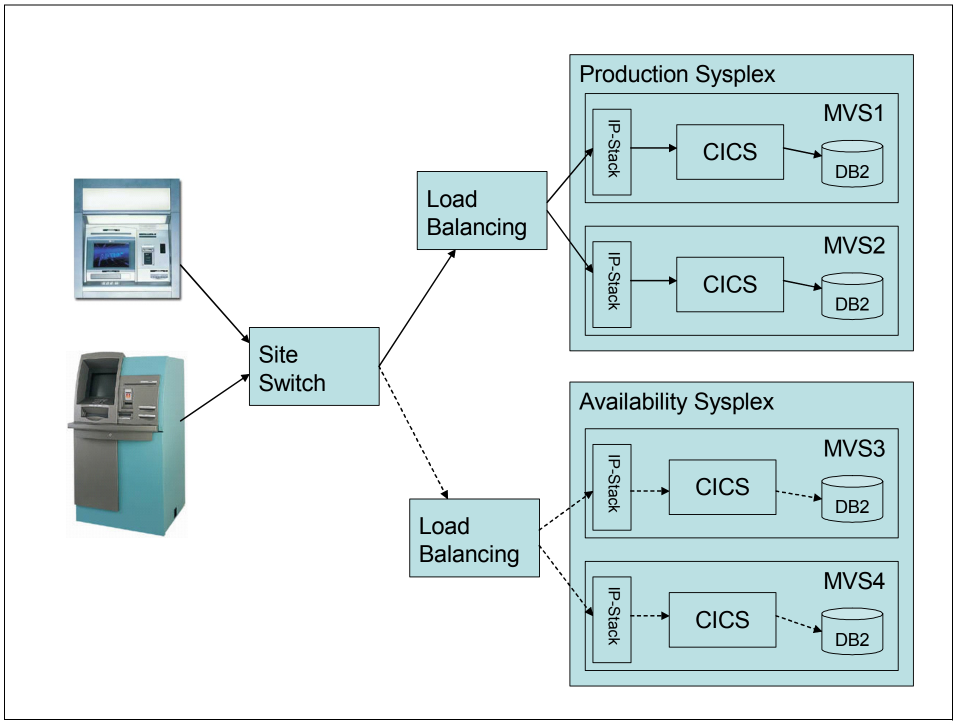


Figure 2-1 ATM overview

To make the application as flexible as possible, the communication between the ATM and host was based on a customized version of IFX, the transport layer was HTTP/SOAP, and the application was based on CICS Web Services on the host side.

To provide for availability 24 hours a day, 7 days a week, the host application was cloned on two CICS regions running on two MVS™ systems in the production sysplex. Furthermore, the system was designed to switch to a similar availability sysplex under planned or unplanned system stops in the production sysplex.

Load balancing and switching between the production sysplex and the availability sysplex was controlled by a Cisco Content Switch. Thus, extra CICS regions could be added without any change in the application, if extra capacity is needed.

Data is replicated between the two sysplexes. Each request type is implemented as a Web Service and runs under its own transaction code. Data is delivered from the Web Services interface to the application programs using channels and containers. Figure 2-2 shows a more detailed view of the ATM system.

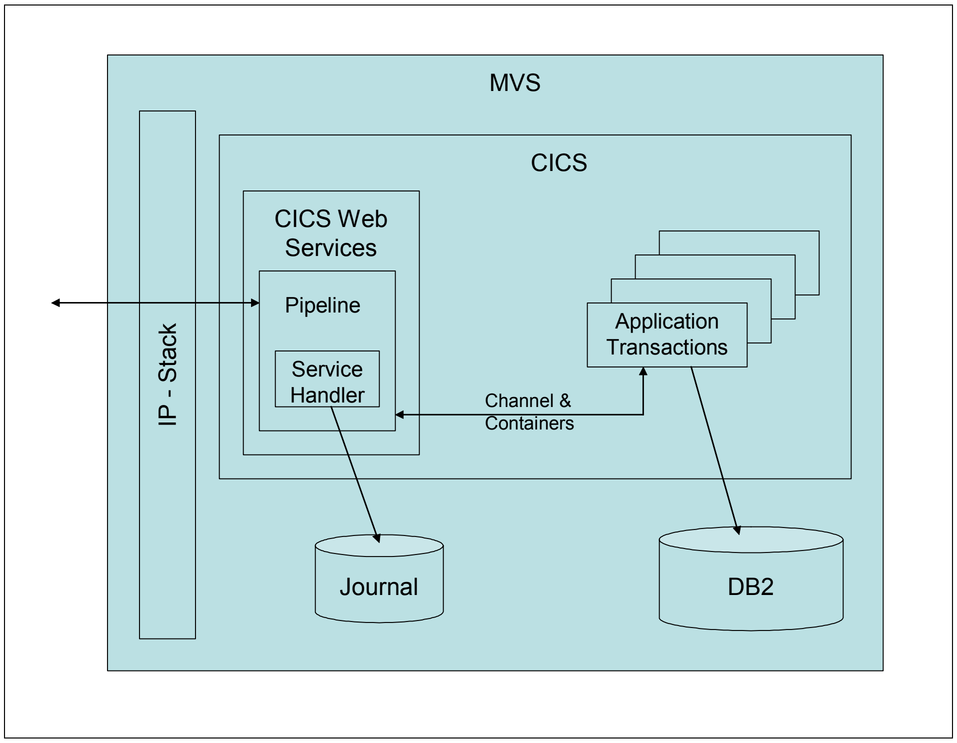


Figure 2-2   ATM details

A Service Handler program is added to ensure that all request input and output is logged in an audit journal. DanskeBank chose to use a Service Handler for the journal activity for two reasons:

- To be able to log the request in XML form; that is, to log data as it is received from or send to the ATM
- To remove logging from the application program (and thereby from the application programmers)

The application handled customer requests and technical requests of the following types:

Customer request

- Card authentication
- Withdrawal
- Account listing
- International Card authentication

Technical request

- Open ATM
- Close ATM
- Print Status

For this investigation we used the Card authentication request and the Withdrawal request.

Example 2-1 illustrates the control structure in the application program for handling the Web Services containers.

Example 2-1   Code for reading request containers

```
** Check in mother container if child is present
*
* if child-name-num > 0 then
*    child-name-num children is present
* else
*    child container not available
*
** get size of input container
   exec cics
       get container (name-from-mother-container)
       flength       (size)
       nodata
       nohandle
   end-exec
*
** code for control of container size
*
** if size OK, get container
   exec cics
       get container (name-from-mother-container)
       set           (pointer)
       flength       (length)
       nohandle
   end-exec
```

Example 2-2 shows the code for the writing response container.

Example 2-2 Code for writing response container

```
* put child container
   exec cics
       put container (child-name)
       flength       (size)
       from          (pointer)
       nohandle
   end-exec
*
* move child-name to mother container
* add 1 to child-name-num in mother container
```

Example 2-3 on page 24 shows the JCL to generate a Web Service. We generate one Web Service for each request type in the application. In this way we are able to allocate separate transaction codes for each request type without coding a wrapper module or a handler.

We started by using mapping level 1.0 when generating the Web Services for this chapter, but later changed to mapping level 1.2. because the performance was considerably better and mapping level 1.2 offers better functionality.

Example 2-3 DFHWS2LS

```
//WS2LSR01 JOB (999,POK),'CICS ws2ls TOOL',MSGCLASS=T,
//             CLASS=A,NOTIFY=&SYSUID,TIME=1440,REGION=0M
/*JOBPARM  S=SC66
//*
//JOBPROC JCLLIB ORDER=CICSTS31.CICS.SDFHINST
//*
//WS2LS     EXEC DFHWS2LS,
//    JAVADIR='java/J1.4',
//    USSDIR='cicsts31',
//    PATHPREF=''
//INPUT.SYSUT1 DD *
  PDSLIB=//CICSRS3.ATM.COBOL
  LANG=COBOL
  PGMINT=CHANNEL
  CONTID=DFHWS-DATA
  REQMEM=ATRQ01
  RESPMEM=ATRS01
  LOGFILE=/u/cicsrs3/webservices/atm/log/atmhost01.log
  WSBIND=/u/cicsrs3/webservices/atm/wsbind/atmhost01.wsbind
  WSDL=/u/cicsrs3/webservices/atm/atmhost01.wsdl
  URI=/PA/PA031100
  BINDING=ATMHostHTTPSoapBinding
  PGMNAME=PA031100
  TRANSACTION=PA11
  MAPPING-LEVEL=1.2
/*
```

The change from mapping level 1.0 to mapping level 1.2 is not just to regenerate the Web Services and recompile the application programs, so choose the correct mapping level from the beginning.

The main changes for the application programs are:

- Every text field is associated with a length variable that must be updated with the actual number of characters in the text string.
- The naming conventions are slightly different. Simple data types referenced in more than one complex type are no longer suffixed for uniqueness.

2.2 Interactive Financial eXchange

The Interactive Financial eXchange (IFX) framework is an open standard for financial transactions in a multiplatform environment. IFX Forum, a non-profit organization, governs the standard. IFX Forum is supported by a large number of leading financial organizations.

IFX is based on XML, and provides a solid base for developing systems that handle financial transactions in applications like branch systems, ATM systems, and business banking. The framework is designed to support multiple currencies and it is easy to customize for the individual organization.

IFX was accepted in 2006 by the ISO organization as a Liaison Member.

For more information about IFX, visit the following Web site:

http://www.ifxforum.org/home

2.3 Changes for this paper

As previously mentioned, for the purpose of this redbook the ATM application was simplified in the following ways:

- All DB2 requests were removed.
- All module calls or links to DanskeBank service components were removed.
- Windows applications were simulated by the use of WebSphere Developer for z/Series; see 2.3.1, “Using WebSphere Developer for zSeries” on page 26 for more information about this topic.

As a result, we have programs that reflect the Web Services handling in CICS and the original control structure of the application program. The setup is run without the Service Handler program.

Figure 2-3 reflects the modified ATM setup that was used.

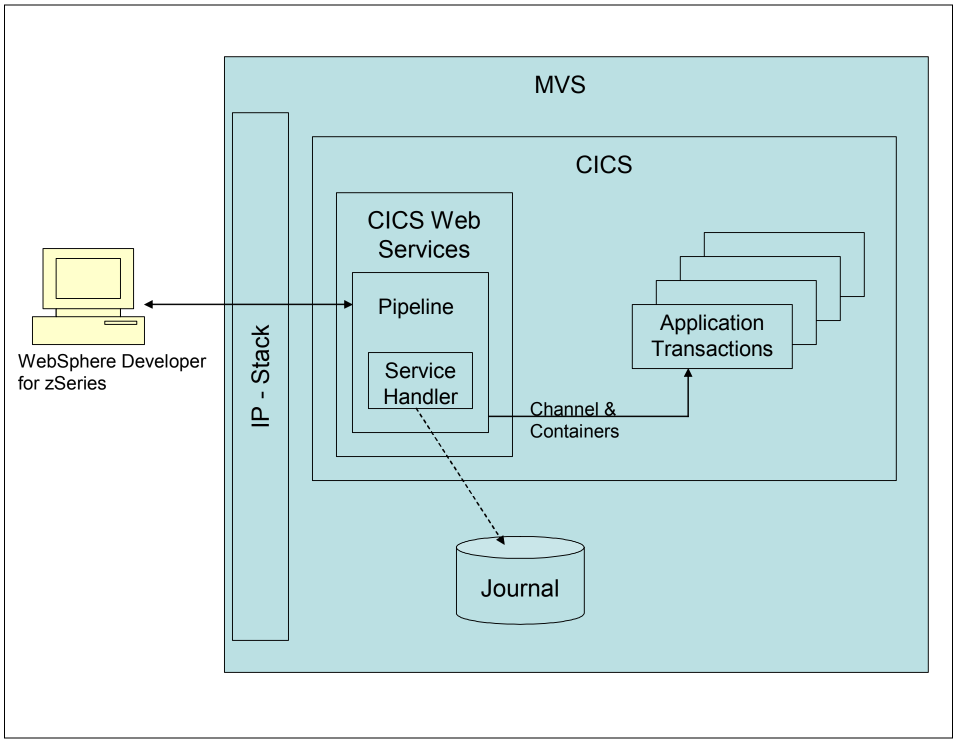


Figure 2-3 Modified ATM setup

2.3.1 Using WebSphere Developer for zSeries

To simulate request messages from an ATM machine, we used WebSphere Developer for zSeries (WD/z). Note that Rational® Application Developer or Rational Software Architect is needed in order to use WD/z.

The simulation starts with the creation of a project in WD/z, as illustrated in Figure 2-4 through Figure 2-8. Figure 2-4 shows the WD/z Start page.

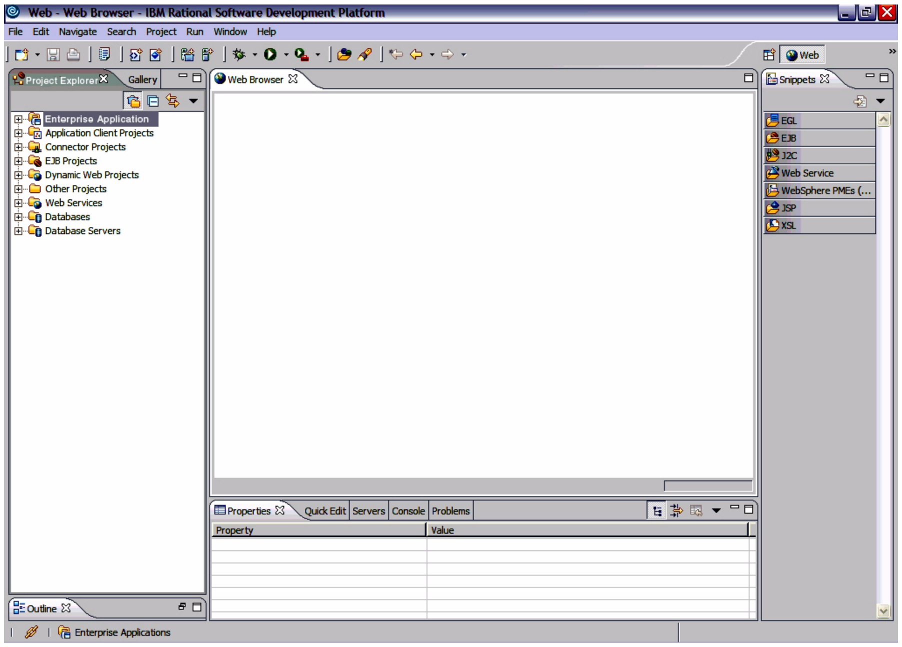


Figure 2-4 WD/z Start page

As illustrated in Figure 2-5, we chose the Simple project, and then clicked Next.

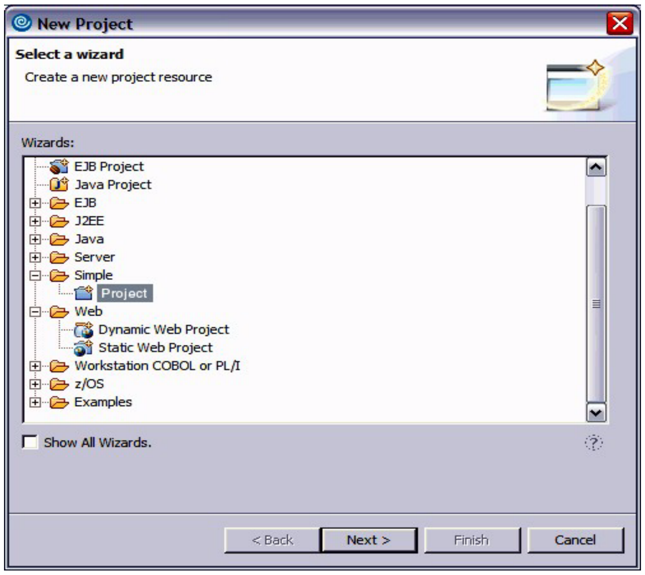


Figure 2-5 Choose Simple project

We named the project and clicked Finish; see Figure 2-6.

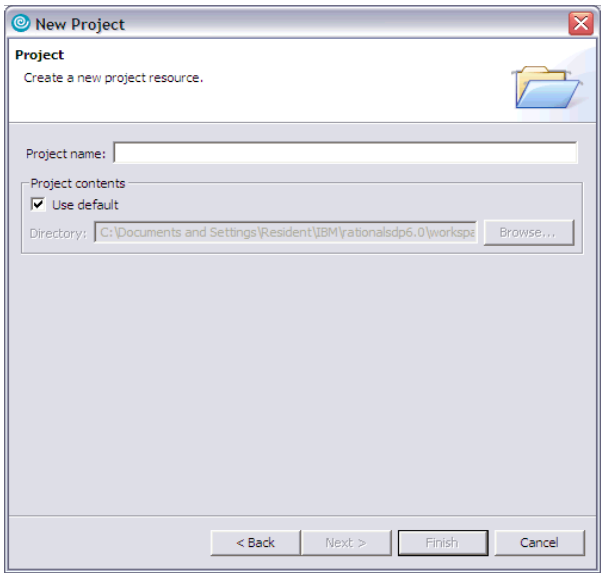

New Project
Project
Create a new project resource.
Project name:
Project contents
Use default
Directory: C:\Documents and Settings\Resident\IBM\rationalsdp6.0\workspa
Browse...
< Back
Next >
Finish
Cancel

Figure 2-6 Name the project and click Finish

We chose the project in the left menu, then right-clicked the project name and clicked Import; see Figure 2-7.

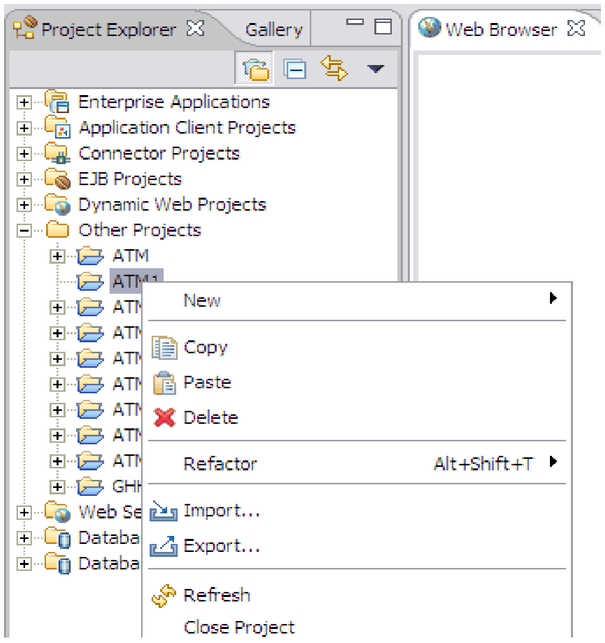
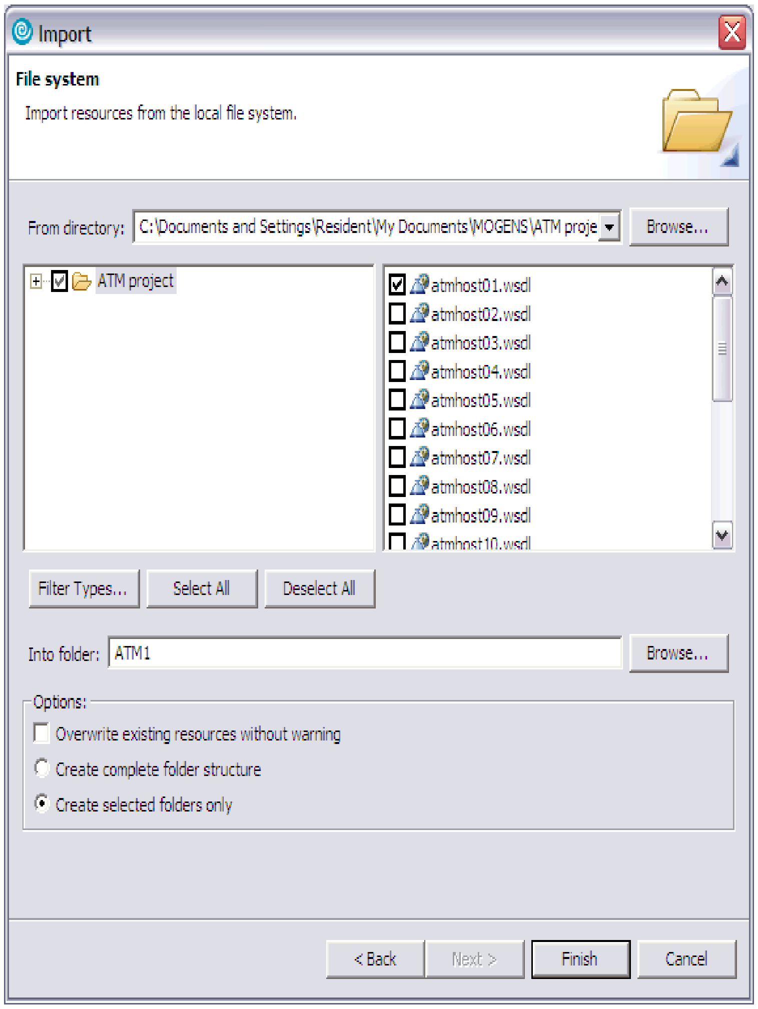


Figure 2-7 Choose project, then import

Next, we needed to import resources. As shown in Figure 2-8, we found the folder with the schema and WSDL definitions, and imported them.


Import

File system

Import resources from the local file system.

From directory: C:\Documents and Settings\Resident\My Documents\MOGENS\ATM proje

[✓] ATM project

[✓] atmhost01.wsdl
[ ] atmhost02.wsdl
[ ] atmhost03.wsdl
[ ] atmhost04.wsdl
[ ] atmhost05.wsdl
[ ] atmhost06.wsdl
[ ] atmhost07.wsdl
[ ] atmhost08.wsdl
[ ] atmhost09.wsdl
[ ] atmhost10.wsdl

Into folder: ATM1

Options:
[ ] Overwrite existing resources without warning
( ) Create complete folder structure
(●) Create selected folders only

Figure 2-8 Find and import schema and WSDL definitions

Finally, as shown in Figure 2-9, we prepared to send a request message.

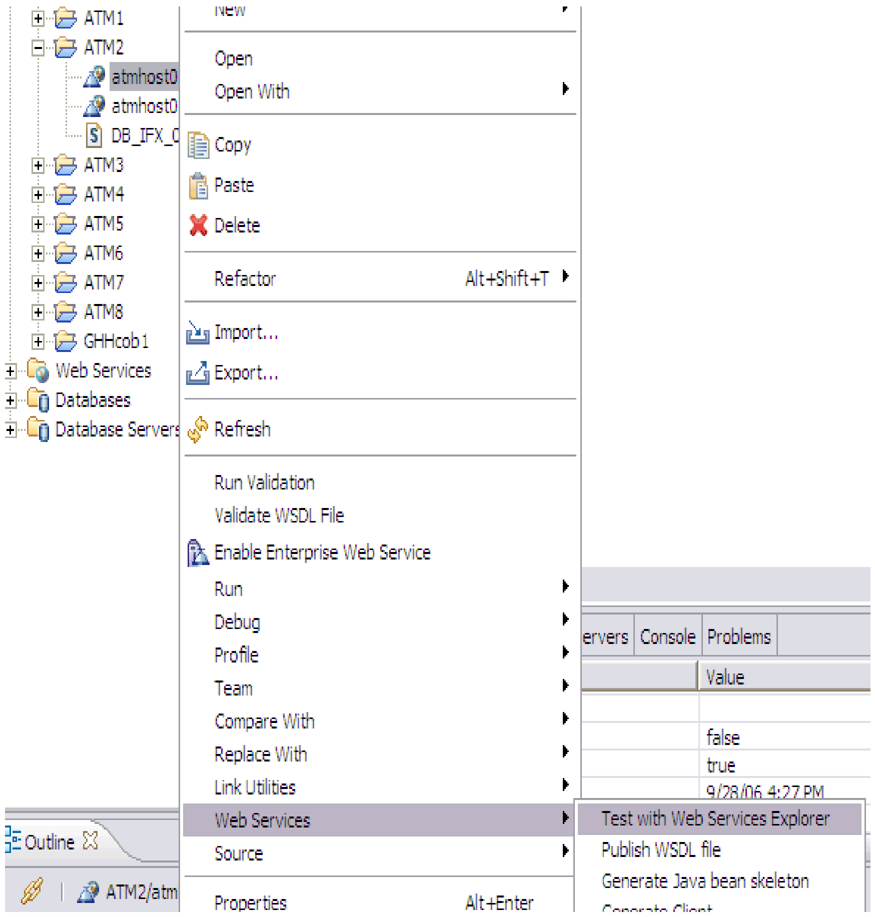
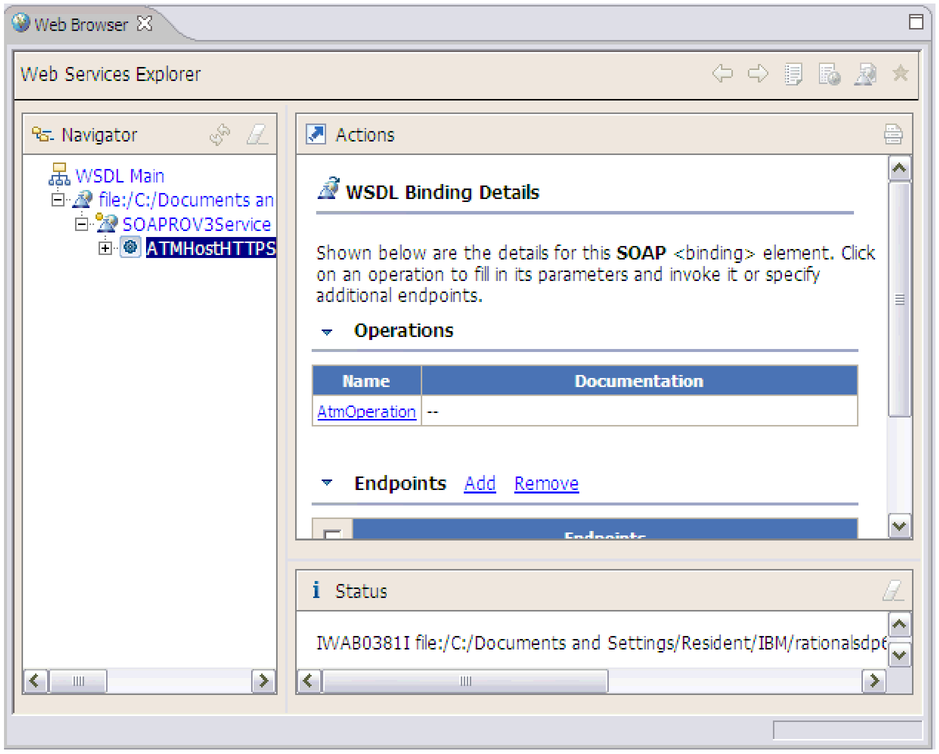


Figure 2-9 Prepare to send a request message

We used the Test with Web Service Explorer function, displayed in Figure 2-10, to send a request message to the host application, as follows:

- We right-clicked the project name in the left menu.
- We clicked Web Services.
- We clicked Test with Web Services Explorer.



WSDL Binding Details

Shown below are the details for this **SOAP** <binding> element. Click on an operation to fill in its parameters and invoke it or specify additional endpoints.

Operations

Name	Documentation
AtmOperation	--

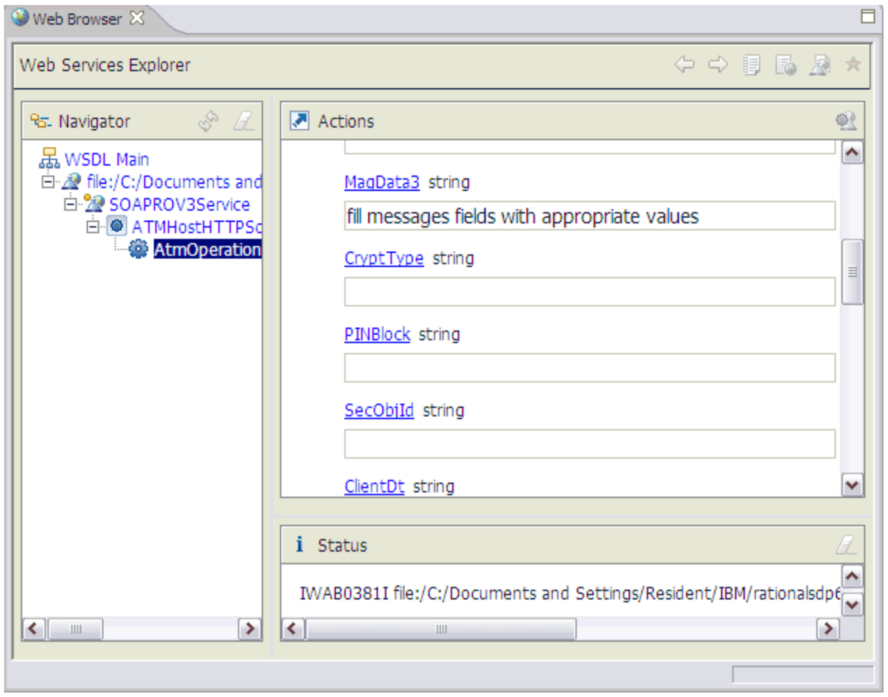
Endpoints Add Remove

IWAB0381I file:/C:/Documents and Settings/Resident/IBM/rationalsdp6

Figure 2-10   The Web Services Explorer Window

We clicked the name of the requested operation (in this example, AtmOperation).

At the Action window, shown in Figure 2-11, we could fill in the appropriate values for the request message.


Web Browser

Web Services Explorer

Navigator

WSDL Main
file:/C:/Documents and
SOAPROV3Service
ATMHostHTTPSo
AtmOperation

Actions

MagData3 string

fill messages fields with appropriate values

CryptType string

PINBlock string

SecObjId string

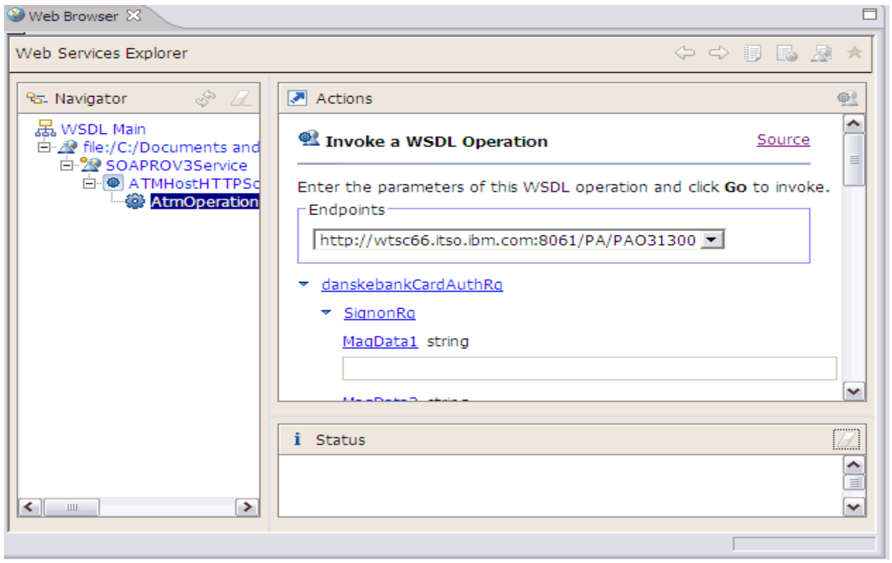
ClientDt string

Status

IWAB0381I file:/C:/Documents and Settings/Resident/IBM/rationalsdp6

Figure 2-11 Fill in the values for the request message

After filling in the message values, as shown in Figure 2-12, we clicked Source.


Web Browser

Web Services Explorer

Navigator

WSDL Main
file:/C:/Documents and
SOAPROV3Service
ATMHostHTTPSo
AtmOperation

Actions

Invoke a WSDL Operation Source

Enter the parameters of this WSDL operation and click **Go** to invoke.

Endpoints

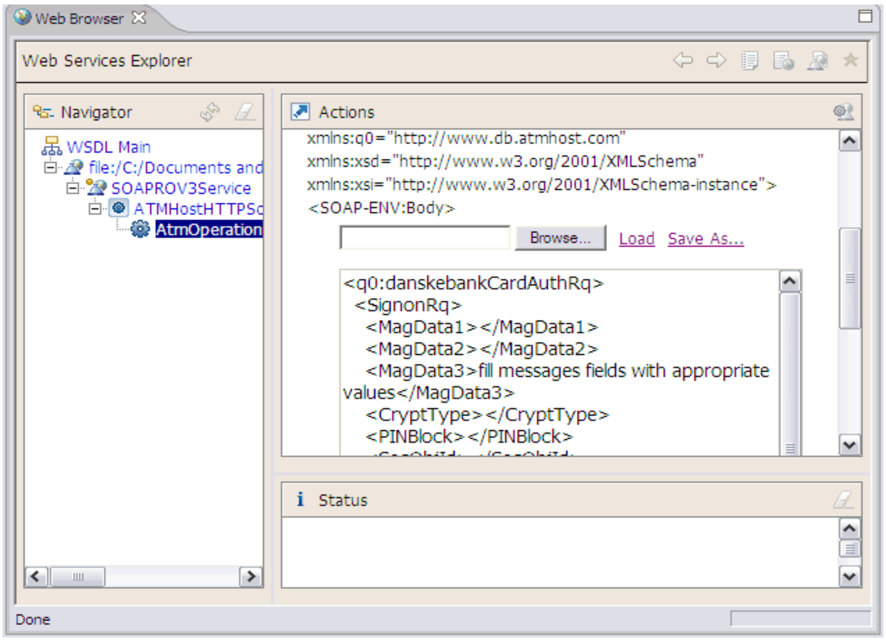
http://wtsc66.itso.ibm.com:8061/PA/PAO31300

danskebankCardAuthRq
SignonRq
MagData1 string

Status

Figure 2-12 After filling the messages values, click Source

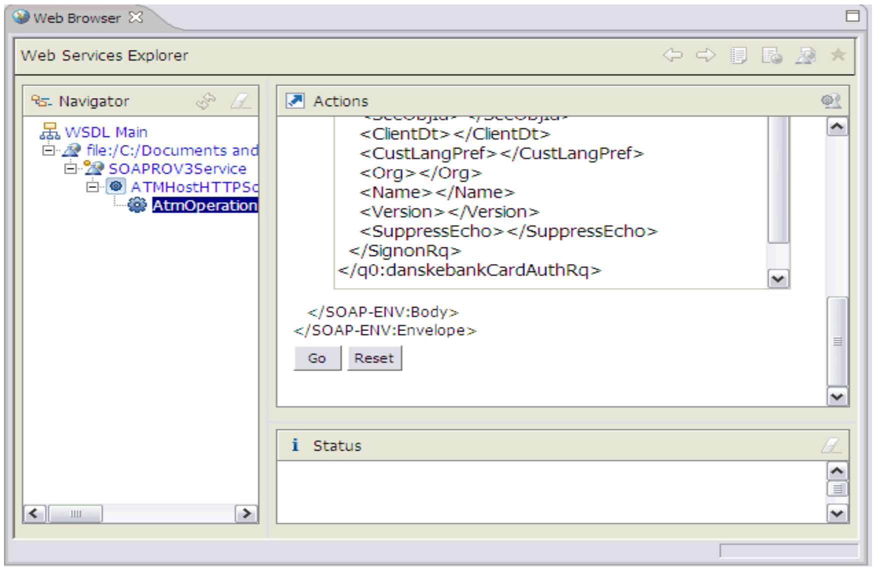
Next, as shown in Figure 2-13, we clicked Save As... to save a copy of the request values for later reuse.



```
xmlns:q0="http://www.db.atmhost.com"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Body>
<q0:danskebankCardAuthRq>
  <SignonRq>
    <MagData1></MagData1>
    <MagData2></MagData2>
    <MagData3>fill messages fields with appropriate
values</MagData3>
    <CryptType></CryptType>
    <PINBlock></PINBlock>
```

Figure 2-13 Save a copy of request values for later reuse

On the Action window, shown in Figure 2-14, we clicked Go to send the message to the server.

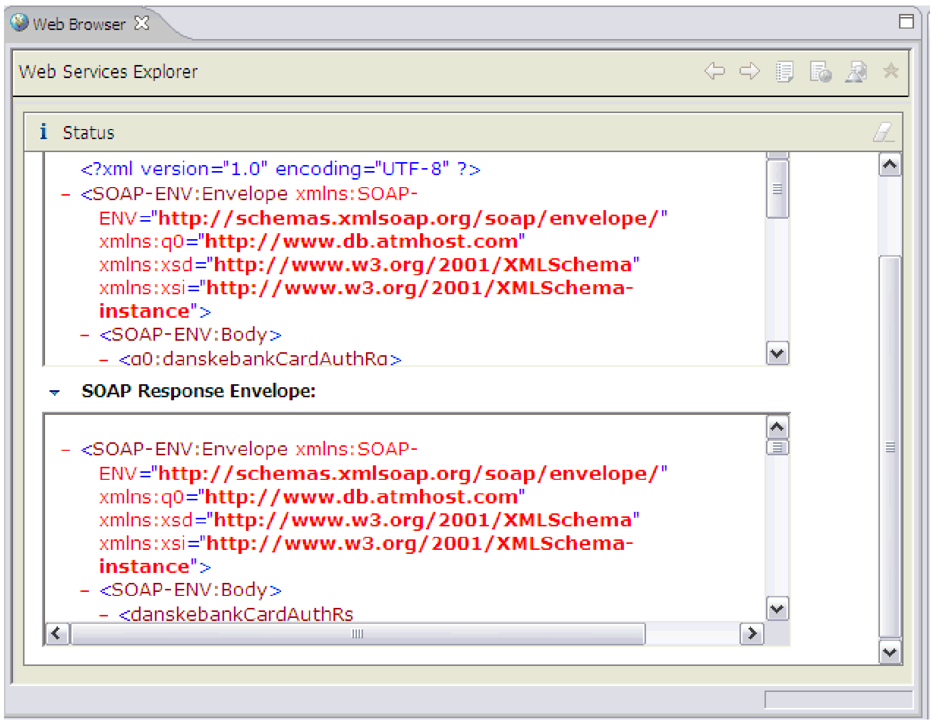


```
    <ClientDt></ClientDt>
    <CustLangPref></CustLangPref>
    <Org></Org>
    <Name></Name>
    <Version></Version>
    <SuppressEcho></SuppressEcho>
  </SignonRq>
</q0:danskebankCardAuthRq>

</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 2-14 Send the message to the server

When the response message arrived from the server, as shown in Figure 2-15, we double-clicked Status to enlarge the Status window and browsed the response message.



```
<?xml version="1.0" encoding="UTF-8" ?>
- <SOAP-ENV:Envelope xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:q0="http://www.db.atmhost.com"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance">
  - <SOAP-ENV:Body>
    - <q0:danskebankCardAuthRq>
```

```
- <SOAP-ENV:Envelope xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:q0="http://www.db.atmhost.com"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance">
  - <SOAP-ENV:Body>
    - <danskebankCardAuthRs
```

Figure 2-15 Status window with response message

2.4 Performance analysis

Table 2-1 on page 34 lists the performance data from the original ATM application (without DB2 and module calls).

PA11 represents a Card authentication request and PA12 represents a Withdrawal request.

Notice that the Container request (Get Container, Put Container) total is more than twice the Container request total initiated in the application itself; the additional requests are made by the Web Services interface.

Table 2-1 Performance data from original ATM application

Tran	**Average CPU time (in sec)**	**Average response time**	**Average container request in application**	**Average container request total**
CWXN	0.0004	0.0011		
PA11	0.0039	0.0048	71	183
PA12	0.0039	0.0046	63	167

25 input containers and 21 output containers may seem excessive for a relatively simple input request to authorize a credit card. The high number of containers, however, is a result of using IFX. The IFX schema is very flexible and built to support many types of applications in any financial business in the world. Because of that the IFX schema is generalized to a very high degree, and nearly all field definition is therefore marked as “optional”. In the conversion

from XML to CICS channels and containers, this results in a CICS structure where many single fields are placed in their own containers.

To explore the performance impact of this use of optional fields and the resulting high number of containers used, we subsequently optimized the schema definition by marking nearly all fields "required" and reduced the complexity of the schema by reducing the number of nesting levels to a maximum of 3.

Table 2-2 on page 35 shows the performance data from the optimized version of the ATM application. PA13 is the optimized version of PA11, and PA14 is the optimized version of PA12. (Refer to 2.5, "Schema versus container definitions" on page 36, for a discussion of the connection between schema definition and the use of containers.)

Table 2-2 Performance data for optimized ATM application

Tran	Average CPU time	Average response time	Average container request in application	Average container request in application
CWXN	0.0004	0.0014		
PA13	0.0028	0.0034	16	76
PA14	0.0024	0.0029	10	60

As shown in Figure 2-16, the optimized version gives a CPU reduction between 0.0011 and 0.0015 seconds, and a similar response time improvement, due to a simpler control structure in the programs and a reduced number of container API calls.

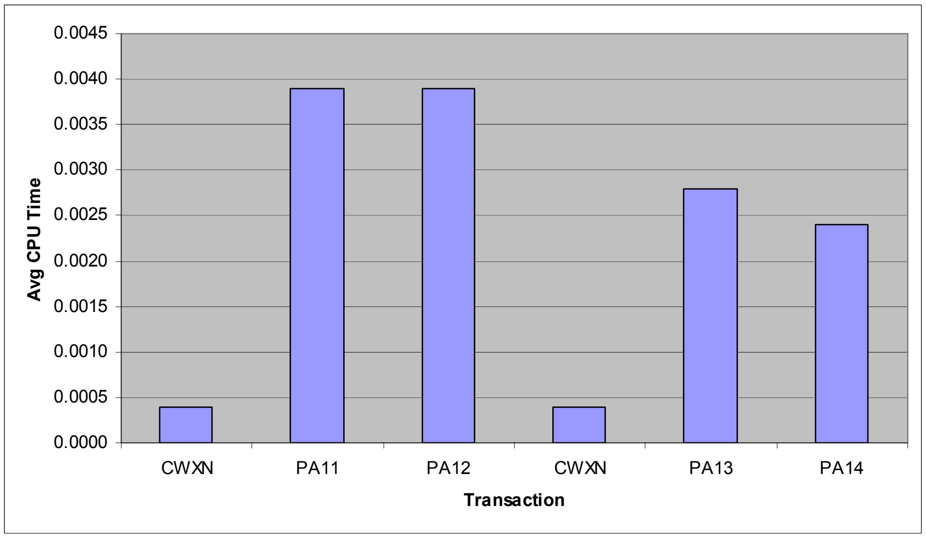


Figure 2-16 Comparison - non-optimized and optimized schema versions

The optimization performed in this example is extreme, and it may not be feasible to optimize to this degree in practice. Furthermore, the CPU savings, due to the effective container API code, is too small to justify redesigning this specific application.

We do, however, recommend that you customize the schema definitions and change the fields marked "optional" to "required" based on the 80-20 rule, as explained here:

- If the field is used in 80% or more of the requests, change the attribute to required.

- If the field is used in 20% or less of the requests, leave the attribute as optional.
- If the use of the field is in the middle area, change the definition if this will simplify development.

To control the use of a specific field that is changed from optional to required, and where the field is not required in all requests, implement one of these solutions:

- Define a null value
- Add an indicator field

The length field generated for text fields, when using mapping level 1.2, will in many situations be sufficient to determine whether a value is present.

2.5 Schema versus container definitions

Next, we provide examples showing how different constructs in the schema definition will impact the container setup in CICS Web Services.

2.5.1 Optional fields

Example 2-4 shows the definition of an address. The example, from IFX, defines only two fields as required and the rest are marked optional, using the attribute minOccurs=0. Both minOccurs and MaxOccurs default to 1.

Example 2-4 Optional fields - schema definition

```
<xsd:complexType name="PostAddr_Type">
      <xsd:sequence>
         <xsd:element ref="Addr1"/>
         <xsd:element ref="Addr2" minOccurs="0"/>
         <xsd:element ref="Addr3" minOccurs="0"/>
         <xsd:element ref="Addr4" minOccurs="0"/>
         <xsd:element ref="City" minOccurs="0"/>
         <xsd:element ref="StateProv" minOccurs="0"/>
         <xsd:element ref="PostalCode" minOccurs="0"/>
         <xsd:element ref="Country"/>
         <xsd:element ref="AddrType" minOccurs="0"/>
         <xsd:element ref="StartDt" minOccurs="0"/>
         <xsd:element ref="EndDt" minOccurs="0"/>
      </xsd:sequence>
   </xsd:complexType>
```

The resulting data areas for the CICS program are shown in Example 2-5.

The two required fields, Addr1 and Country, are placed in the PostAddr container. The other fields are allocated in separate containers, and indicator fields -num and -cont, are placed in the PostAddr container.

Example 2-5 Optional fields - container definitions

```
   01 ATRS0101-PostAddr.
           05 PostAddr.
             10 Addr1-length                  PIC S9999 COMP-5 SYNC.
```

```
          10 Addr1                          PIC X(64).

          10 Addr2-num                      PIC S9(9) COMP-5 SYNC.
          10 Addr2-cont                     PIC X(16).

          10 Addr3-num                      PIC S9(9) COMP-5 SYNC.
          10 Addr3-cont                     PIC X(16).

          10 Addr4-num                      PIC S9(9) COMP-5 SYNC.
          10 Addr4-cont                     PIC X(16).

          10 City-num                       PIC S9(9) COMP-5 SYNC.
          10 City-cont                      PIC X(16).

         10 StateProv-num                  PIC S9(9) COMP-5 SYNC.
          10 StateProv-cont                 PIC X(16).

          10 PostalCode-num                 PIC S9(9) COMP-5 SYNC.
          10 PostalCode-cont                PIC X(16).

          10 Country-length                 PIC S9999 COMP-5 SYNC.
          10 Country                        PIC X(3).

          10 AddrType-num                   PIC S9(9) COMP-5 SYNC.
          10 AddrType-cont                  PIC X(16).

          10 StartDt-num                    PIC S9(9) COMP-5 SYNC.
          10 StartDt-cont                   PIC X(16).

          10 EndDt-num                      PIC S9(9) COMP-5 SYNC.
          10 EndDt-cont                     PIC X(16).

01 ATRS0101-Addr2.
        05 Addr2-length                   PIC S9999 COMP-5 SYNC.
        05 Addr2                          PIC X(64).

01 ATRS0101-Addr3.
        05 Addr3-length                   PIC S9999 COMP-5 SYNC.
        05 Addr3                          PIC X(64).

01 ATRS0101-Addr4.
        05 Addr4-length                   PIC S9999 COMP-5 SYNC.
        05 Addr4                          PIC X(64).

01 ATRS0101-City.
        05 City-length                    PIC S9999 COMP-5 SYNC.
        05 City                           PIC X(32).

01 ATRS0101-StateProv.
        05 StateProv-length               PIC S9999 COMP-5 SYNC.
        05 StateProv                      PIC X(32).

01 ATRS0101-PostalCode.
        05 PostalCode-length              PIC S9999 COMP-5 SYNC.
        05 PostalCode                     PIC X(11).
```

```
   01 ATRS0101-AddrType.
          05 AddrType-length              PIC S9999 COMP-5 SYNC.
          05 AddrType                     PIC X(80).

   01 ATRS0101-StartDt.
          05 StartDt-length               PIC S9999 COMP-5 SYNC.
          05 StartDt                      PIC X(255).

   01 ATRS0101-EndDt.
          05 EndDt-length                 PIC S9999 COMP-5 SYNC.
          05 EndDt                        PIC X(255).
```

The data area definitions demonstrate that, because of the high degree of normalization in the schema definition, a simple address ends up in 10 separate containers. We recommend that your use optional fields with care.

Example 2-6 illustrates a reasonable optimization of the PostAddr definition.

Example 2-6 Optional fields - optimized schema definition

```
<xsd:complexType name="PostAddr_Type">
 <xsd:sequence>
   <xsd:element ref="Addr1"/>
   <xsd:element ref="Addr2"/>
   <xsd:element ref="Addr3"/>
   <xsd:element ref="Addr4"/>
   <xsd:element ref="City"/>
   <xsd:element ref="StateProv" minOccurs="0"/>
   <xsd:element ref="PostalCode" minOccurs="0"/>
   <xsd:element ref="Country"/>
   <xsd:element ref="AddrType"/>
   <xsd:element ref="StartDt" minOccurs="0"/>
   <xsd:element ref="EndDt" minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>
```

With this optimization, the code used for sending a standard address is reduced from six Put Container API calls to one. If all fields are used, ten API calls are reduced to five.

Example 2-7 illustrates the simplification in the data area definitions for containers.

Example 2-7 Optional fields - optimized container definition

```
01 ATRS0701-PostAddr.
           05 Addr1-length              PIC S9999 COMP-5 SYNC.
           05 Addr1                     PIC X(64).
           05 Addr2-length              PIC S9999 COMP-5 SYNC.
           05 Addr2                     PIC X(64).
           05 Addr3-length              PIC S9999 COMP-5 SYNC.
           05 Addr3                     PIC X(64).
           05 Addr4-length              PIC S9999 COMP-5 SYNC.
```

```
     05 Addr4                         PIC X(64).
     05 City-length                   PIC S9999 COMP-5 SYNC.
     05 City                          PIC X(32).

    05 StateProv-num                   PIC S9(9) COMP-5 SYNC.
    05 StateProv-cont                  PIC X(16).

    05 PostalCode-num                  PIC S9(9) COMP-5 SYNC.
    05 PostalCode-cont                 PIC X(16).

    05 Country                         PIC X(3).
    05 AddrType                        PIC X(80).

    05 StartDt-num                     PIC S9(9) COMP-5 SYNC.
    05 StartDt-cont                    PIC X(16).

    05 EndDt-num                       PIC S9(9) COMP-5 SYNC.
    05 EndDt-cont                      PIC X(16).

  01 ATRS0701-StateProv.
     05 StateProv-length              PIC S9999 COMP-5 SYNC.
     05 StateProv                     PIC X(32).

  01 ATRS0701-PostalCode.
     05 PostalCode-length             PIC S9999 COMP-5 SYNC.
     05 PostalCode                    PIC X(11).

  01 ATRS0701-StartDt.
     05 StartDt-length                PIC S9999 COMP-5 SYNC.
     05 StartDt                       PIC X(255).

  01 ATRS0701-EndDt.
     05 EndDt-length                  PIC S9999 COMP-5 SYNC.
     05 EndDt                         PIC X(255).
```

2.5.2 The choice compositor

The choice compositor is used to describe a choice between a number of possible elements or element groups. Example 2-8 illustrates the use of the choice compositor. The message can include AcctId or CardMagData, but not both.

Example 2-8 Choice compositor - schema definition

```
<xsd:complexType name="CardAcctId_Type">
  <xsd:sequence>
   <xsd:choice>
    <xsd:element ref="AcctId"/>
    <xsd:element ref="CardMagData"/>
   </xsd:choice>
   <xsd:element ref="AcctType"/>
  </xsd:sequence>
 </xsd:complexType>
```

The data area for CardAcctId contains indicators for the optional fields AcctId and CardMagData. AcctId and CardMagData is placed in seperate containers.

A special case of the choice compositor is where the schema definition only indicates one choice. The result is an optional field, and will be the same as if the field were defined with minOccurs=0.

Example 2-9 Choice compositor - container definition

```
01 ATRQ0201-CardAcctId.

           05 AcctId-num                    PIC S9(9) COMP-5 SYNC.
           05 AcctId-cont                   PIC X(16).

           05 CardMagData1-num              PIC S9(9) COMP-5 SYNC.
           05 CardMagData1-cont             PIC X(16).

           05 AcctType-length               PIC S9999 COMP-5 SYNC.
           05 AcctType                      PIC X(80).

         01 ATRQ0201-AcctId.
           05 AcctId-length                 PIC S9999 COMP-5 SYNC.
           05 AcctId                        PIC X(32).

         01 ATRQ0201-CardMagData1.

           05 MagData11-num                 PIC S9(9) COMP-5 SYNC.
           05 MagData11-cont                PIC X(16).

           05 MagData21-num                 PIC S9(9) COMP-5 SYNC.
           05 MagData21-cont                PIC X(16).

           05 MagData31-num                 PIC S9(9) COMP-5 SYNC.
           05 MagData31-cont                PIC X(16).

         01 ATRQ0201-MagData11.
           05 MagData1-length               PIC S9999 COMP-5 SYNC.
           05 MagData1                      PIC X(79).

         01 ATRQ0201-MagData21.
           05 MagData2-length               PIC S9999 COMP-5 SYNC.
           05 MagData2                      PIC X(40).

         01 ATRQ0201-MagData31.
           05 MagData3-length               PIC S9999 COMP-5 SYNC.
           05 MagData3                      PIC X(107).
```

2.6 Large fields

The ATM system includes text fields for sending free form text to display on the ATM terminal. The free form text is for help, advertising, guidelines, and so on.

As illustrated in Example 2-10, the two text fields in transaction PA13 were filled with 5 to 10 characters and the length field was updated accordingly.

Example 2-10 Schema definition and corresponding COBOL definition

```
<xsd:simpleType name="edtText1_Type">
  <xsd:annotation>
   <xsd:documentation>Advertising text to be displayed to the customer. May
contain HTML tags</xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="NC">
   <xsd:minLength value="0"/>
   <xsd:maxLength value="4096"/>
   <xsd:whiteSpace value="preserve"/>
  </xsd:restriction>
 </xsd:simpleType>
 <xsd:simpleType name="edtText2_Type">
  <xsd:annotation>
   <xsd:documentation>Advertising text to be displayed to the customer. May
contain HTML tags</xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="NC">
   <xsd:minLength value="0"/>
   <xsd:maxLength value="4096"/>
   <xsd:whiteSpace value="preserve"/>
  </xsd:restriction>
 </xsd:simpleType>


      10 edtText1-length                PIC S9999 COMP-5 SYNC.
      10 edtText1                       PIC X(4096).
      10 edtText2-length                PIC S9999 COMP-5 SYNC.
      10 edtText2                       PIC X(4096).
```

For this test we changed the schema definitions of the two fields to the sizes listed in Table 2-3. The corresponding programs were changed to fill the fields with data, and to set the length field. (We used 32000 bytes as the maximum, because single fields larger than 32 K are placed in a separate container.)

Table 2-3 Field size

Tran	Field size	Data length
PA15	96	96
PA16	32000	31000
PA17	4096	4096
PA18	32000	300

Table 2-4 on page 42 lists the performance impact of the various field sizes. The difference between the small sizes (96 and 4096) is insignificant, but changing to 32 K creates an overhead in both CPU time and elapsed time. Also note the extra container API calls.

In PA18, we only send 300 bytes but have a field size of 32000 bytes, and then see a result comparable to PA15 and PA17.

Table 2-4 Message size impact

Tran	Average CPU time	Average response time	Average container request in application	Average container request total
CWXN	0.0004	0.0015		
PA15	0.0028	0.0034	16	71
CWXN	0.0004	0.0016		
PA16	0.0052	0.0169	16	76
CWXN	0.0004	0.0015		
PA17	0.0030	0.0036	16	71
CWXN	0.0004	0.0016		
PA18	0.0032	0.0038	16	71

For best performance, it is important to set the correct length (and not just the maximum value) when using text fields. The larger the field, the more important it is to set the correct length.

Figure 2-17 shows the field size comparison.

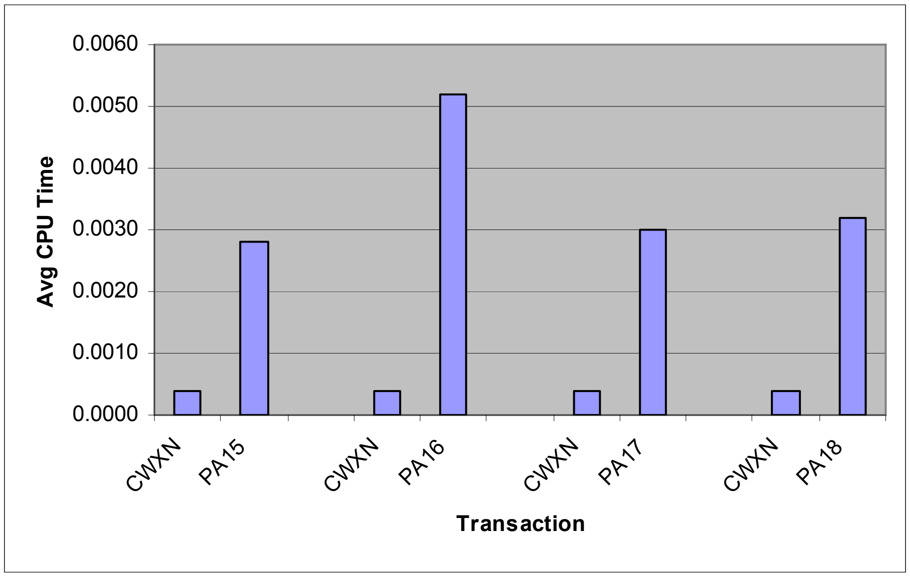


Figure 2-17 Field size comparison

2.7 Schema complexity

The original schema for the ATM application was constructed of 163 simple types and 100 complex types. The nesting level was 9 in the schema definitions and 5 in the generated COBOL data definitions.

To see how the performance is impacted by the schema complexity, we created transaction PA19 to use a modified schema similar to that used by PA13, but changed it to have a nesting level of 25, giving a COBOL nesting level of 23.

PA19 uses the same programs as in PA13, and the data sent and received is the same. The only difference is in schema complexity; see Table 2-5.

Table 2-5 Performance data for complex schema

Tran	Average CPU time	Average response time	Average container request in application	Average container request total
CWXN	0.0004	0.0014		
PA13	0.0028	0.0034	16	76
CWXN	0.0005	0.0028		
PA19	0.0036	0.0044	16	71

As shown in Figure 2-18, there was an increase in both CPU time and response time, demonstrating that simplicity, as always, is preferred.

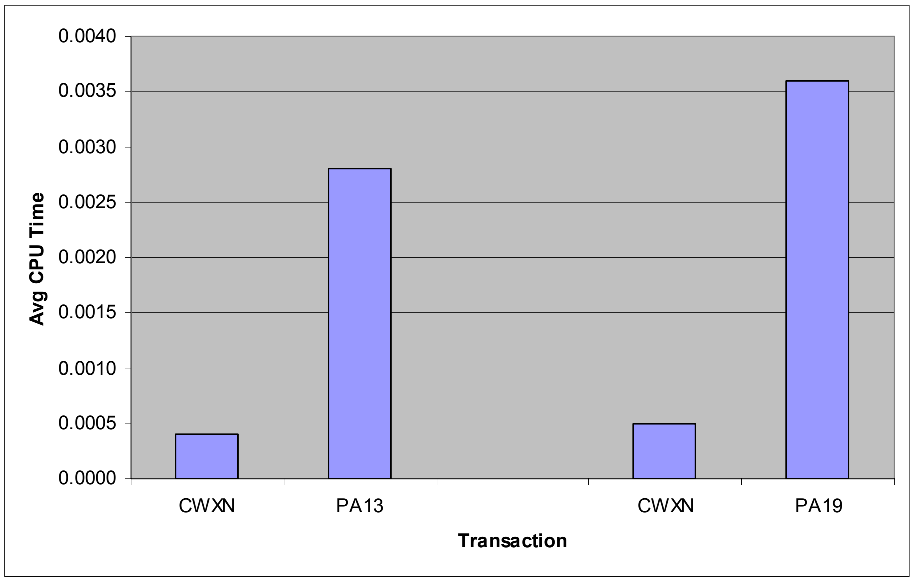


Figure 2-18 Schema complexity comparison

While generating the COBOL data structures from the complex schema definitions, we found an error in the Web Service Assistant. The COBOL structures were generated with level number 01, 05, 10. With 23 levels that gave us COBOL level numbers up to 115.

Because 49 is the highest legal level number in COBOL (except for special types), we had to manually renumber the COBOL structures to be able to compile the application programs. An APAR (PK35396) was created to correct this potential problem.

2.8 Mapping level 1.0

As previously mentioned, we found that using mapping level 1.2 gave better performance than mapping level 1.0.

We compared two sets of programs:

- PA19 and PA20 with Web Service generated under mapping level 1.2
- PA21 and PA22 with Web Service generated under Mapping-level 1.0

PA19 is identical to PA21, and PA20 is identical to PA22. We receive and send the same data, and the only programming changes were the required extra length variables used under mapping level 1.2.

Table 2-6 shows the impact of mapping level.

Table 2-6 Mapping level impact

Tran	Average CPU time	Average response time	Average container request in application	Average container request total
CWXN	0.0005	0.0028		
PA19	0.0036	0.0044	16	71
PA20	0.0035	0.0042	11	65
CWXN	0.0005	0.0031		
PA21	0.0054	0.0180	16	76
PA22	0.0039	0.0046	11	65

The performance improvement is larger between PA19 and PA 21, than between PA20 and PA22. The difference is to be found in the schema definitions for the messages used. Notice the additional container API calls, made by the Web Service interface in PA21.

Figure 2-19 illustrates the mapping level comparison.

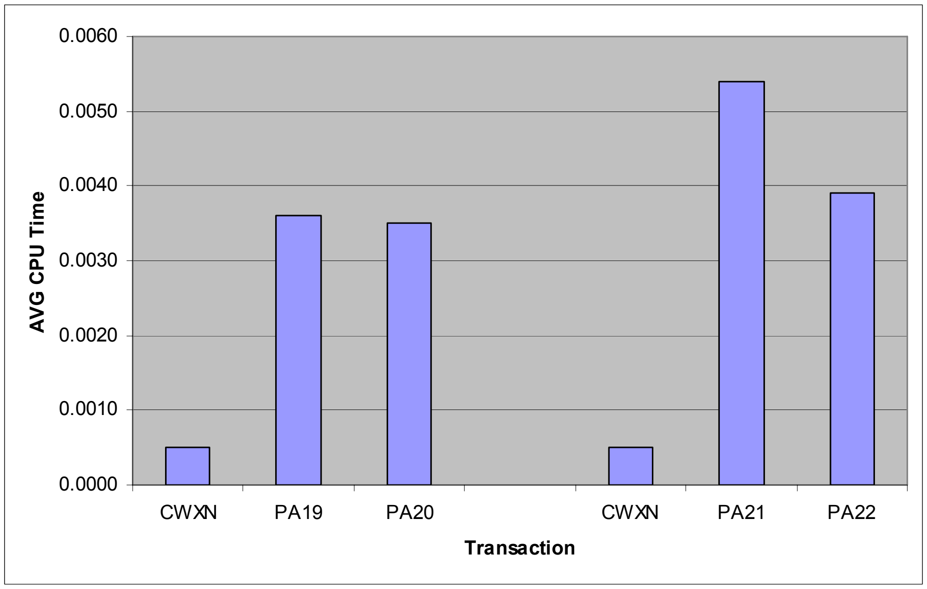


Figure 2-19 Mapping level comparison

PA19 and PA21 send 9 fields with a size of 4096 bytes, but the actual text string in the fields is only 10 to 20 bytes in length.

The request and reply messages for PA20 and PA22 contain only small fields of up to 255 bytes.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

IBM Redbooks

For information about ordering these publications. Note that some of the documents referenced here may be available in softcopy only.

- Implementing CICS Web Services, SG24-7206.

Other publications

These publications are also relevant as further information sources:

- *z/OS MVS System Management Facility,* SA22-7630
- *CICS Performance Analyzer for z/OS User's Guide,* SC34-6307
- *CICS Performance Analyzer for z/OS Report Reference,* SC34-6308
- *CICS Web Services Guide,* SC34-6458
- *WebSphere Studio Workload Simulator User's Guide,* SC31-6307

How to get IBM Redbooks publications

You can search for, view, or download Redbooks publications, Redpapers publications, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

SOAP Message Size Performance Considerations

DanskeBank ATM application performance

A look at SOAP message size performance

Example of the use of WebSphere Developer for zSeries

This IBM Redpaper publication examines performance when using different SOAP message sizes in CICS Web Services. We used a preexisting basic Web Services environment with no message handlers or header handlers. Using WebSphere Studio Workload Simulator (WSWS), we ran a number of simulations using predefined scripts. For more information about WSWS, refer to *WebSphere Studio Workload Simulator User's Guide*, SC31-6307. We then processed the CICS SMF data using CICS Performance Analyzer (CICS PA) and collated the statistics. For more information about CICSPA, see *CICS Performance Analyzer User's Guide,* SC34-6307.

This paper also reviews a customer scenario, DanskeBank. We provide an overview of the bank's ATM system, and briefly discuss the IFX framework. We detail our use of WebSphere Developer for zSeries (WD/z) to simulate request messages from an ATM, present a performance comparison between optimized and non-optimized schema versions, and cover schema versus container definitions, large fields, and schema complexity. Finally, we provide an example of the impact of mapping levels.

REDP-4344-00