z/TPF V1.1

TPF Users Group Spring 2008

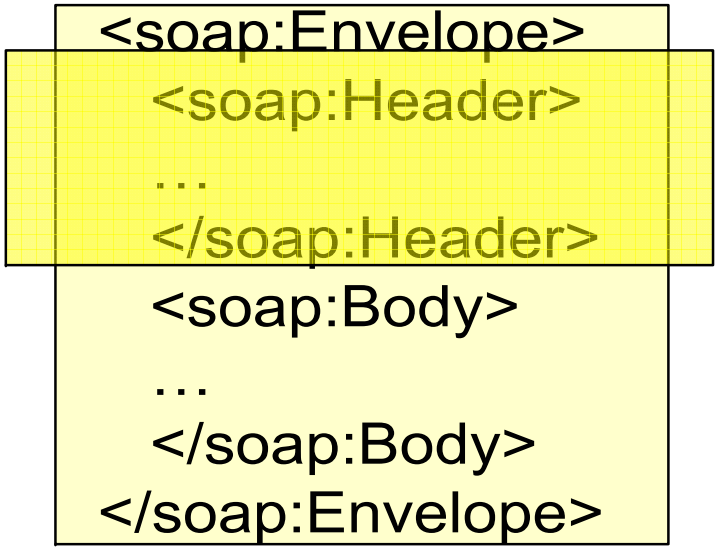
SOAP Message Handlers:
Extending the basic SOAP protocol

Jason Keenaghan
Distributed Systems Subcommittee

AIM Enterprise Platform Software
IBM z/Transaction Processing Facility Enterprise Edition 1.1.0

Extending the basic Web services support

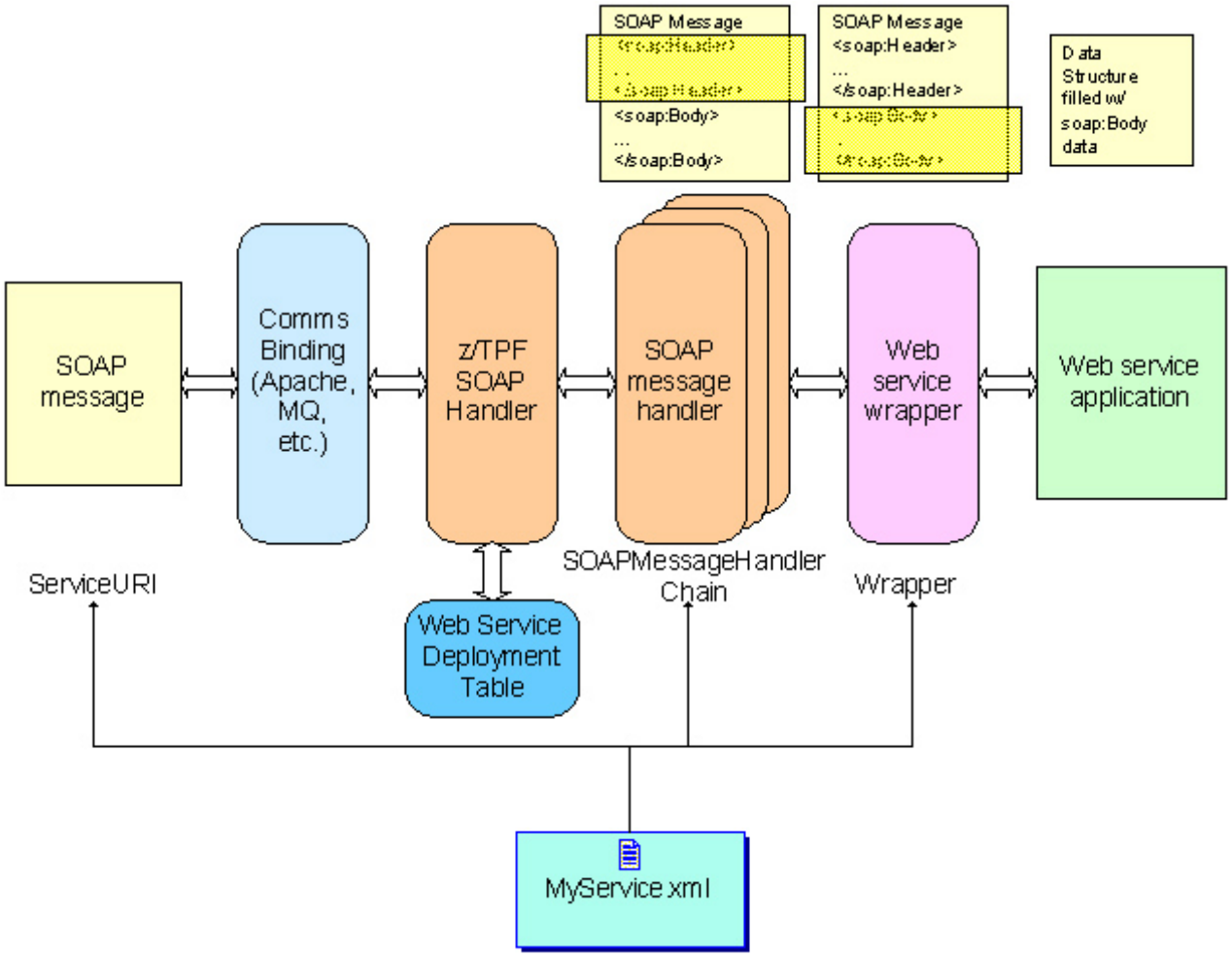
- The SOAP specifications allow for a great deal of extensibility by using the “Header” section of SOAP messages
- A large number of secondary specifications have evolved for solving common business problems; collectively, these are known as ***SOAP features*** or ***WS-****
  - WS-Security
  - WS-Addressing
  - WS-Reliable Messaging
- As of z/TPF PUT 4 (PJ32215), you can easily create reusable Web service extensions, known on z/TPF as ***SOAP message handlers***



Taking advantage of SOAP message handlers

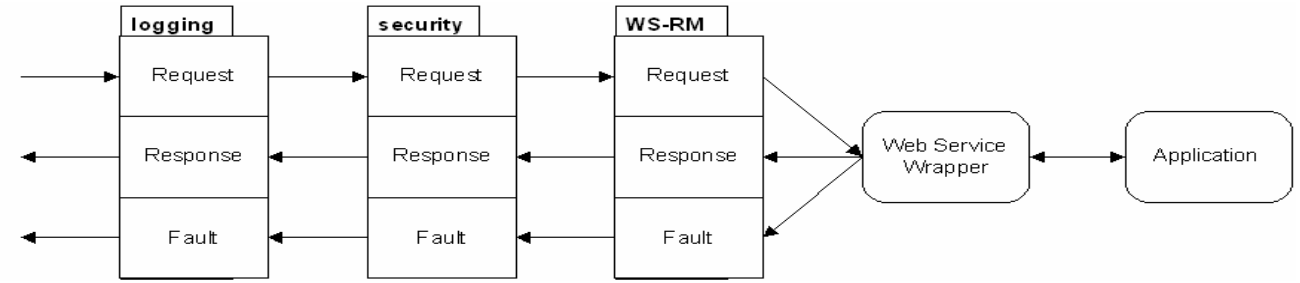
- SOAP message handlers provide a single, reusable interface for extending one or more provider Web services
  - Standard SOAP features: WS-Security, WS-Addressing, WS-*
  - User-specific functionality: request/response logging, etc.
- Creating SOAP message handlers for use on z/TPF requires 2 artifacts:
  - ***SOAP message handler deployment descriptor:*** defines the runtime characteristics of the SOAP message handler
  - ***SOAP message handler program:*** performs necessary processing using the inbound SOAP request and the outbound SOAP response
- Each provider Web service that requires a SOAP message handler to be called must include the message handler's name in the SOAPMessageHandlerChain element of its own deployment descriptor

Logical processing flow of a SOAP request



Implementing a SOAP message handler program

- SOAP message handlers are invoked by the z/TPF SOAP handler in each of the following situations:
  - **SOAP request:** called in the order they are listed in the Web service's SOAP message handler chain before calling the Web service wrapper
  - **SOAP response:** called in the reverse order they are listed in the Web service's SOAP message handler chain after calling the Web service wrapper
  - **SOAP fault:** called in the reverse order they are listed in the Web service's SOAP message handler chain



SOAP message handler interface

```
typedef enum SOAP_MSGHDLR_RET(*intfn)(enum SOAP_MSGHDLR_ACT,
                                      tpfSoapMsgCtx*);
```

- **Parameter #1:** Specifies reason for invoking SOAP message handler (received request, sending response, sending SOAP fault)
- **Parameter #2:** SOAP message context structure (defined in `<tpf/c_soapctx.h>`
  - Contains XML API handles for the SOAP request and response
  - Contains a complete copy of the Web service deployment table (WSDT) entry for the intended Web service
  - Macros available to get information about this SOAP message handler from the SOAP message context. For example:
    - `TPF_SOAP_MSGCTX_GET_HDLR_WRKAREA`
    - `TPF_SOAP_MSGCTX_SET_HDLR_WRKAREA`

Example uses for SOAP message handlers

- **logging**: Log information about SOAP requests/responses to the syslog facility
- **SNMP_alert:** Send an SNMP alert when a SOAP fault is generated in response to a SOAP request
- **msg_expiration:** Validate the WS-Security expiration timeout value before processing SOAP request and before sending SOAP response

Example #1: *logging*

- Request Flow: MSGHDLR_ACT_REQ

```
openlog ("WSLOG", LOG_PID, LOG_USER);

syslog (LOG_INFO, "Request logged for service %.32s, operation: %.32s",
        ctx->wsdt_ent.key, xptr->nodesArray[0].nodeName);

closelog ();
```

- Response Flow: MSGHDLR_ACT_RESP

```
openlog ("WSLOG", LOG_PID, LOG_USER);

syslog (LOG_INFO, "Response logged for service %.32s, operation: %.32s",
        ctx->wsdt_ent.key, xptr->nodesArray[0].nodeName);

closelog ();
```

- Fault Flow: MSGHDLR_ACT_FAULT

```
openlog ("WSLOG", LOG_PID, LOG_USER);

syslog (LOG_NOTICE, "Fault logged for service %.32s, operation: %.32s",
        ctx->wsdt_ent.key, xptr->nodesArray[0].nodeName);

closelog ();
```

Note: Complete sample is available for download at http://www-306.ibm.com/software/htp/tpf/download/ztpfsoap.htm

Example #2: *SNMP_alert*

- Request Flow: MSGHDLR_ACT_REQ
  - Do nothing
- Response Flow: MSGHDLR_ACT_RESP
  - Do nothing
- Fault Flow: MSGHDLR_ACT_FAULT
  - Setup: you will need to define your own Enterprise-specific management information base (MIB) variable that represents the particular service

```
tpf_snmp_BER_encode ();  /* BER encode the ObjectID,
                            value, binding*/

tpf_itrpc ():            /* Send the Trap to the
                            remote SNMP destination
                            managers defined in
                           /etc/snmp.cfg */
```

Example #3: *msg_expiration*

- Using a feature of WS-Security specification, validate that a SOAP request has not expired, as specified by the consumer, before processing the SOAP request or sending back the SOAP response

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="..." xmlns:wsse="..." xmlns:wsu="...">
   <SOAP-ENV:Header>
      <wsse:Security SOAP-ENV:mustUnderstand="true">
         <wsu:Timestamp wsu:Id="timestamp">
           <wsu:Created>2008-03-10T01:56:02Z</wsu:Created>
           <wsu:Expires>2008-03-10T01:56:03Z</wsu:Expires>
         </wsu:Timestamp>
      </wsse:Security>
   </SOAP-ENV:Header>
   ...
</SOAP-ENV:Envelope>
```

Example #3: *msg_expiration* (continued)

- Request Flow: `MSGHDLR_ACT_REQ`
  - Inspect the `Expires` timestamp and ensure that the message hasn't expired upon receipt

```
time_t curTime;
time_t *expireTime = (time_t *)malloc(sizeof time_t);
/* Get current time */
curTime = time(NULL);
/* Compare it with <wsu:Expires> */
nodeArray = tpf_xml_getNextElementByTagName
              (xmlPtr, TYPE_DATETIME, "Expires");
*expireTime =
  mktime(nodeArray->nodesArray[0].nodeValueDataType.nodeValueDateTime;
/* If message has expired, return fault, else save expiration in work area */
if (*expireTime < curTime)
  tpf_soap_build_fault_ext (soapFaultPtr, &faultHandle);
else
  TPF_SOAP_MSGCTX_SET_HDLR_WRKAREA (ctx, expireTime);
```

Example #3: *msg_expiration* (continued)

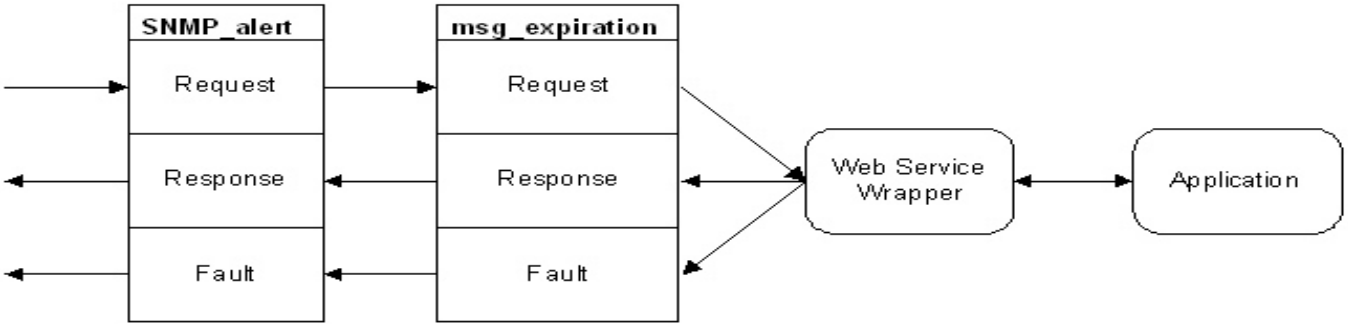
- Response Flow: MSGHDLR_ACT_RESP
  - Check the current time against the previously saved expiration time and ensure that the message hasn't expired just prior to sending the response

```
time_t curTime;
time_t *expireTime;
/* Get current time */
curTime = time(NULL);
/* Compare it with the expiration time */
expireTime = (time_t *)TPF_SOAP_MSGCTX_GET_HDLR_WRKAREA (ctx);
/* If message has expired, return fault and activate transaction rollback */
if (*expireTime < curTime) {
  tpf_soap_build_fault_ext (soapFaultPtr, &faultHandle);
  if (((pid = fork()) != -1) && (pid))
    execl ("/test.sh", lniata, NULL);  }
```

- Fault Flow: MSGHDLR_ACT_FAULT
  - Do nothing

Combining the message handlers

- **Problem:**
  - msg_expiration SOAP message handler only informs the SOAP consumer in the case of an expired message
  - Your SNMP management console must be notified of this event as well
- **Solution:** Include both message handlers in your message handler chain for the particular Web service



Even more uses for SOAP message handlers

- *billing* SOAP message handler
  - Capture system utilization statistics from ECB when request is received just before handing it off to wrapper
  - Compare with system utilization statistics from ECB after response is built
  - Update billing information based on actual system resources used to satisfy the consumer request
- Extensibility options are limitless!

Trademarks

- IBM and z/TPF are trademarks of International Business Machines Corporation in the United States, other countries, or both.
- Notes:
  - Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.
  - This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.
  - This presentation and the claims outlined in it were reviewed for compliance with US law. Adaptations of these claims for use in other geographies must be reviewed by the local country counsel for compliance with local laws.