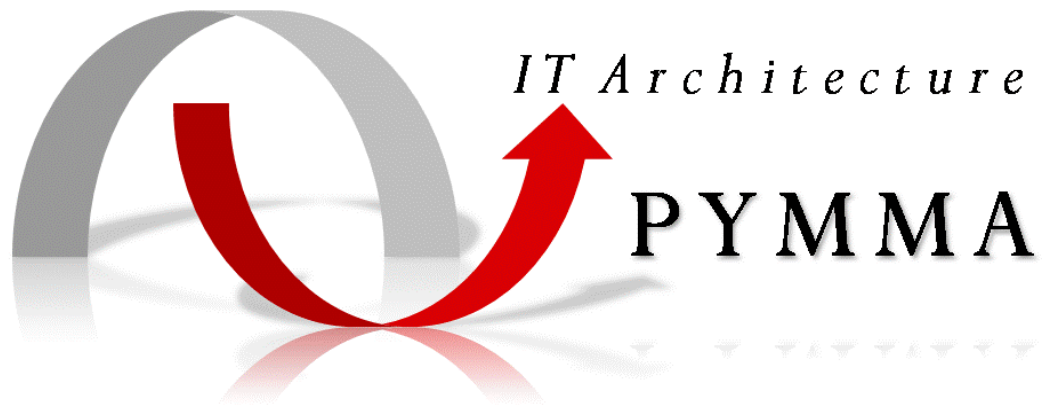


Pymma Consulting Papers



Tutorials JBI & Open-ESB

Soap Header Management with wsdl

Paul Perez & Bruno Sinkovic

Pymma (2008)

To the readers : Since my native tongue is French and my English is far from perfect, I had the choice between writing papers in an excellent French understandable by French speakers or in a very bad English understandable by a great majority of IT people. Even if I look foolish, I choose the second option to be understandable by the maximum of people. (Paul Perez)

Abstract

This paper describes the best practices for SOAP header management with Open-ESB. We noticed that open-ESB users often face difficulties when dealing with Soap Header with WSDL documents. In Fact, JBI 1.0 imposes to design services interfaces only with the abstract part of WSDL documents where nothing has been plan to support protocol characteristics like Soap Headers. Moreover, Open-ESB does not propose “hooks” or interceptor to modify the Soap Messages with low-level languages like Java or c#. The aim of this tutorial is to find an easy way to manage SOAP headers in Open-ESB, by using the abstract part of WSDL documents.

We hope that this tutorial will be pleasant to read and useful for you. Please do not hesitate to send your comments and remarks to contact@pymma.com. You will find additional details about our JBI and Open-ESB services and trainings on our web site www.pymma.com

Prerequisites

In order to enjoy reading this paper and benefit from it, you will need to be familiar with the main concepts of JBI and Open-ESB. We also assume that you are able to read and understand XML Schemas, WSDL and BPEL documents.

Content and Objectives

This tutorial contains a simple but significant example on Soap header management. With this example, we will design a new JBI composite application that offers the functionality of sending SMS messages. We call this application “**SMSSender**”.

For sending SMS, “**SMSSender**” uses services provided by a company AbcText (www.abcText.com). These services are based on Microsoft technologies. The WSDL documents provided by AbcText to send SMS are suitable as examples for the topic of this paper (Soap headers): first because they are simple and secondly because they utilise SOAP headers to transport session IDs.

Remark : This paper focused on “Soap header” management and not the development of a sending SMS component. We will propose a complete tutorial on this topic soon.

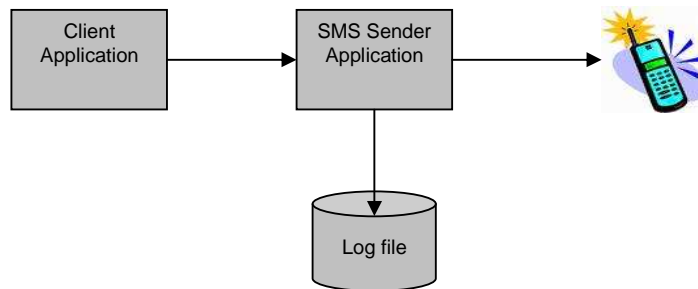
Note: Pymma Consulting has no commercial agreement or partnership with AbcText. Pymma uses the AbcText’s Services only as an illustration for this paper.

SMS Sender component specifications

We propose to start this tutorial by defining a high-level specifications for the SMSSender application.

Our main requirement is to provide a simple and easy-to-use service to send SMS. The service interface must be very simple and hide all protocol details to the end user.

As usual in SOA, the service is defined with a WSDL document. The protocol used by the client applications to invoke SMSSender's service is SOAP/HTTP.



1. A client application invokes the services of SMSSender with parameters and a message to send
2. SMS Sender attempts to send the message as an SMS into the mobile network.
3. SMS Sender logs a trace file on disk and indicates if the message was sent successfully or if it encountered errors.

Overview of AbcText services

If you have a look at the [AbcText](http://www.abctext.com) web site, you will find 7 web services, all designed to send SMS. In order to access to AbcText services, you must be registered and pay about 8 GBP to send between 50 and 100 SMS. Noted: it is worth to pay it! You will probably have lots of fun, as I did, when you will hear your mobile phone ringing and reading the message you just sent from the web.

You will find below the list of services available at: <http://abctext.com/webservices/servicesdevelopers.aspx>

Abctext Services		
Service	Operation	Description
Session	StartSession	Starts a session that enables you to interact with our web services
Session	EndSession	Ends a session immediately, meaning that you must re-authenticate with AbcText
SMS	GetCreditBalance	Returns the number of credits remaining
SMS	GetMessageRecipientStatus	Returns the status of a message for a specified recipient number and user phone message id
SMS	IsSenderIdAvailable	Returns a boolean confirming whether a SenderId is available
SMS	QuickSend	Allows you to send a single SMS message instantly to many recipients
SMS	Send	Allows you to send multiple SMS messages, each with its own optional delivery time, to many recipients. String interface for non-.Net clients.

In order to keep the tutorial simple, we use the **Quicksend** service rather than the Send service.

Technical overview of AbcText services

We provide now a technical overview of “AbcText” services. You may find useful documents on AbcText website at <http://abctext.com/webservices/servicesdevelopers.aspx>

- WSDL documents
- XML Schemas
- Soap operations traces
- VB Code

QuickSend Service Soap traces

Diving into the SOAP traces of [QuickSend](#) service provided by AbcText, we discover that for security reasons, AbcText treats a session key in the SOAP message Header. Thereby, if you do not provide this SessionKey, QuickSend will reject your request and you will not be able to send any SMS. We guess that SessionKey is associated with an Http Session in the AbcText web server.

Before invoking the QuickSend services, we need to get a session key from AbcText. In order to obtain it, we invoke the service [startSession](#) and **authenticate** with our login and password as parameters.

The figures below show the SOAP Traces for the service “startSession”. The first figure traces the Soap request, then the second traces the Soap response:

```
POST /webservices/Session.asmx HTTP/1.1
Host: abctext.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://sjmillerconsultants.com/webservices/StartSession"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<StartSession xmlns="http://sjmillerconsultants.com/webservices">
<sUserName>string</sUserName>
<sPassword>string</sPassword>
</StartSession>
</soap:Body>
</soap:Envelope>

Code 1: Soap code for StartSession request
```



```

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <StartSessionResponse xmlns="http://sjmillerconsultants.com/webservices">
      <StartSessionResult>string</StartSessionResult>
    </StartSessionResponse>
  </soap:Body>
</soap:Envelope>

```

You get the sessionKey

Code 2: Soap code for StartSession response

Note :The startSession service returns a SessionKey available for 15 minutes.

The figure below shows the SOAP traces illustrating the usage of the SessionKey. To invoke the QuickSend Service, we have to insert the SessionKey in the Soap header

```

POST /webservices/SMS.asmx HTTP/1.1
Host: abctext.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://sjmillerconsultants.com/webservices"

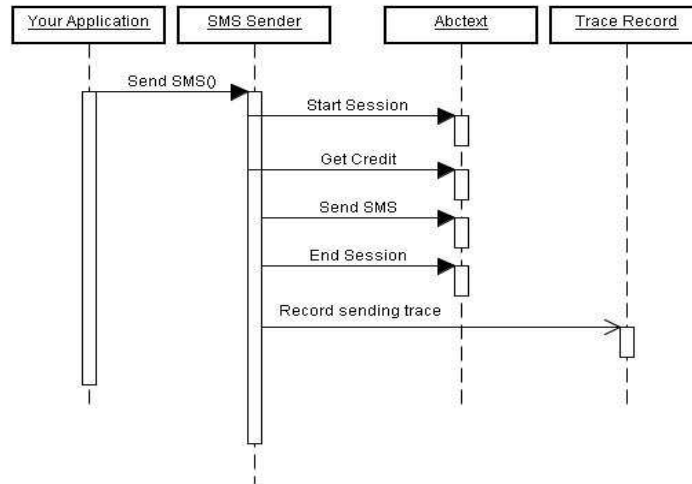
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <AuthorizationHeader xmlns="http://sjmillerconsultants.com/webservices">
      <SessionKey>string</SessionKey>
    </AuthorizationHeader>
  </soap:Header>
  <soap:Body>
    <QuickSend xmlns="http://sjmillerconsultants.com/webservices">
      <SMSRequest xmlns="http://sjmillerconsultants.com/sms/send">
        <Message>string</Message>
        <Unicode>boolean</Unicode>
        <Sender>string</Sender>
        <MessageType>Text or FlashText</MessageType>
        <Recipients>
          <Recipient>
            <Number>string</Number>
          </Recipient>
          <Recipient>
            <Number>string</Number>
          </Recipient>
        </Recipients>
      </SMSRequest>
    </QuickSend>
  </soap:Body>
</soap:Envelope>

```

Insert the sessionKey in the Soap header when sending a SMS

The Sequence diagram below summarises the calls sequence to send a SMS. In our prototype, we added an additional call to log SMSSender activities (component "Trace recorder").

To keep the tutorial simple and easy to understand, we ignored error and compensation processing in this diagram.



Code 3: Soap message for QuickSend Service request

Note: the last call "Record Sending Trace" is asynchronous.

The process is the following:

- we first invoke StartSession to get a SessionKey
- we check if there is enough credit
- we send the SMS
- Lastly, we close the Session and record the sending in a log file .

Summary

AbcText provides a Session Key during the process of identification (Start Session Services). QuickSend uses it as ID card when invoking the Abctext Server. The Session key must be inserted in the Soap message header.

We all know that a JBI component uses WSDL Abstract part to define message structure and not the WSDL concrete part. So, when SMSSender communicates with AbcText services, it uses AbcText WSDL Abstract part as reference to build the messages to AbcText as well. Unfortunately, QuickSend WSDL Abstract part has no reference to the Session Key (*Concrete part has a reference to the sessionKey*) and does not contain any parameter dedicated to store the protocol parameters.

If we want to use AbcText with Open ESB, we have an issue: On one hand, JBI messages are defined by the abstract part of WSDLs, on the other hand, WSDL abstract part does not deal with protocol parameters (ex: soap header). Because of this problem, we are not able to use the WSDL of AbcText "as is" for a open-ESB component.

In the next section of the tutorial, we will analyse the WSDL of AbcText in order to work around this issue.

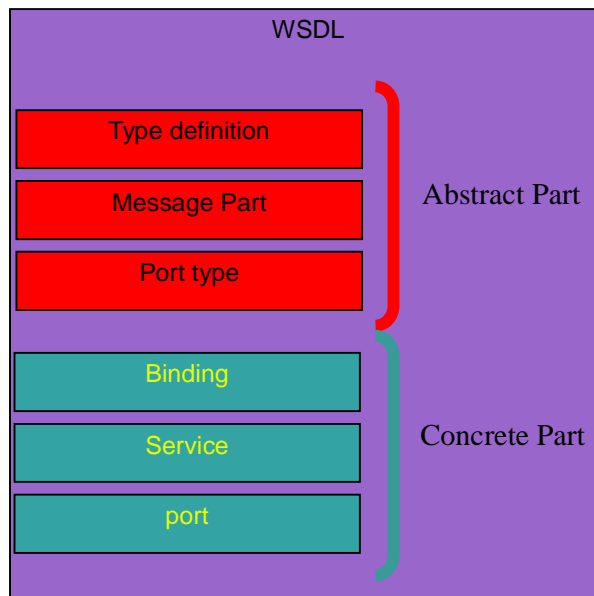
Analysis of Abctext WSDL

Now let's analyse in detail QuickSend WSDL design by AbcText.

In the previous chapter, we saw that Abctext's wsdl's are not fit for purpose with open-ESB. In order to understand why, we will first analyse AbcText's WSDLs. Then, we will propose a solution to get around this issue.

WSDL Abstract part

In a WSDL document, two parts are defined: The abstract part and the concrete part. For more information, you may want to look at the WSDL specifications ([click for information](#)).



The WSDL abstract part defines the message structures (type definition) and the interface of services (port type)

Let us have a look on QuickSend WSDL document.

In the port type definition, we note that the parameters for "QuickSend" operation are QuickSendSoapIn (*complexType SMSRequest*) and QuickSendSoapout (*complexType SMSResponse*).

```
<wsdl:portType name="SMSSoap">
  <wsdl:operation name="QuickSend">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Allows you to send a single SMS message instantly to many recipients.</documentation>
    <wsdl:input message="tns:QuickSendSoapIn" />
    <wsdl:output message="tns:QuickSendSoapOut" />
  </wsdl:operation>
```

Port Type definition: Quick Send operation

SMSRequest and SMSResponse types are described below.

```

<s:complexType name="SMSResponse">
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="UserMessageId" type="s:long" />
    <s:element minOccurs="1" maxOccurs="1" name="Credits" type="s:double" />
    <s:element minOccurs="1" maxOccurs="1" name="Units" type="s:long" />
    <s:element minOccurs="0" maxOccurs="1" name="Message" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="Recipients" type="s1:ArrayOfSMSRecipientResponse" />
  </s:sequence>
</s:complexType>

```

QuickSendSoapOut is a SMSResponse complexType

```

<s:complexType name="SMSRequest">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="Message" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1" name="Unicode" type="s:boolean" />
    <s:element minOccurs="0" maxOccurs="1" name="Sender" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1" name="MessageType" type="s1:MessageType" />
    <s:element minOccurs="0" maxOccurs="1" name="Recipients" type="s1:ArrayOfSMSRecipient" />
  </s:sequence>
</s:complexType>

```

QuickSendSoapIn is a SMSRequest complexType

Note: portTypes are similar to Java interfaces, where QuickSend would be a method and QuickSendSoapIn and QuickSendSoapOut its parameters with the respective types SMSRequest and SMSResponse.

```

public interface SMSSoap {
    public SMSResponse quickSend (SMSRequest quickSendSoapIn) ;
}

```

Port Type: Java code equivalence

We clearly see that there is no reference to the Session Key in the WSDL abstract part.

WSDL Concrete part

Let us examine now the concrete part of the WSDL of AbcText.

In the WSDL "Binding", we define the protocol for invoking QuickSend. The Binding details the protocol specifications:

```

<wsdl:binding name="SMSSoap" type="tns:SMSSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  <wsdl:operation name="QuickSend">
    <soap:operation soapAction="http://sjmillerconsultants.com/abc-services/QuickSend" style="document" />
    <wsdl:input>
      <soap:body use="literal" parts="parameters" />
      <soap:header message="tns:QuickSendSoapIn" part="headerParameters" use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
      <soap:header message="tns:QuickSendAuthorizationHeader" part="AuthorizationHeader" use="literal" />
    </wsdl:output>
  </wsdl:operation>

```

QuickSend Soap Binding

In the Soap binding WSDL expresses needs for a soap header

Part used for the soap header is named : "headerParameters"

Header Parameters

To invoke QuickSend with the Soap protocol, we use two message parts: "parameters" and "headerParameter". We use the former in order to define SOAP message body, and the latter to define SOAP Header.

In the WSDL Concrete part, AbcText informs us that soap header parameters are required.

"AuthorizationHeader" is one of them.

```
<s:complexType name="AuthorizationHeader">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="SessionKey" type="s:string" />
  </s:sequence>
</s:complexType>
```

AuthorizationHeader complex Type

AbcText WSDL in a JBI environment.

According to JBI 1.0 Specification, we know that JBI component interfaces are based on the abstract part of the WSDL associated to the component. We also know that the WSDL concrete part is never used by JBI components. From its part, Open-ESB uses WSDL concrete parts to setup binding component and not to define services interfaces.

So, In a JBI environment when SMSSender communicates with AbcText services, it uses QuickSend WSDL Abstract part as reference to build the messages to AbcText. Unfortunately, QuickSend WSDL Abstract part has no reference to the Session Key. It does not contain parameter dedicated to store the SessionKey.

If we would have to use QuickSend WSDL as defined by AbcText, we were not able to send any SMS. We draw the conclusion that we cannot use the WSDLs design by AbcText "as is" to add a session Key in the SOAP header.

Question : *Why is there no reference to Soap header parameters in the WSDL abstract part?*

I guess there are two answers for this question.

The first answer is that AbcText does not want to introduce protocol parameters in the abstract service definition. In general, it is a good practice to prevent the insertion of protocol aspects in the WSDL Abstract part.

The second answer is that AbcText uses low-level language to modify SOAP messages at the runtime. As proof, AbcText proposes code samples to develop client service with "VB.Net". We clearly see that the Visual Basic code modifies directly Soap headers. AbcText designed WSDLs for low-level development language environment where few lines of code are able to insert any parameter in the Soap.

Session Management with a WSDL document

To solve this problem, we propose a simple solution. Let's modify AbcText's WSDL and let's add a new part in the QuickSendSoapIn message in the WSDL abstract part that contains values for the Soap Header. This new message part is named "headerParameters".

Let's modify QuickSend WSDL abstract part and add a new part named "headerParameter" with the existing type "AuthorizationHeader" in the "QuickSendSoapIn" message as describes below.

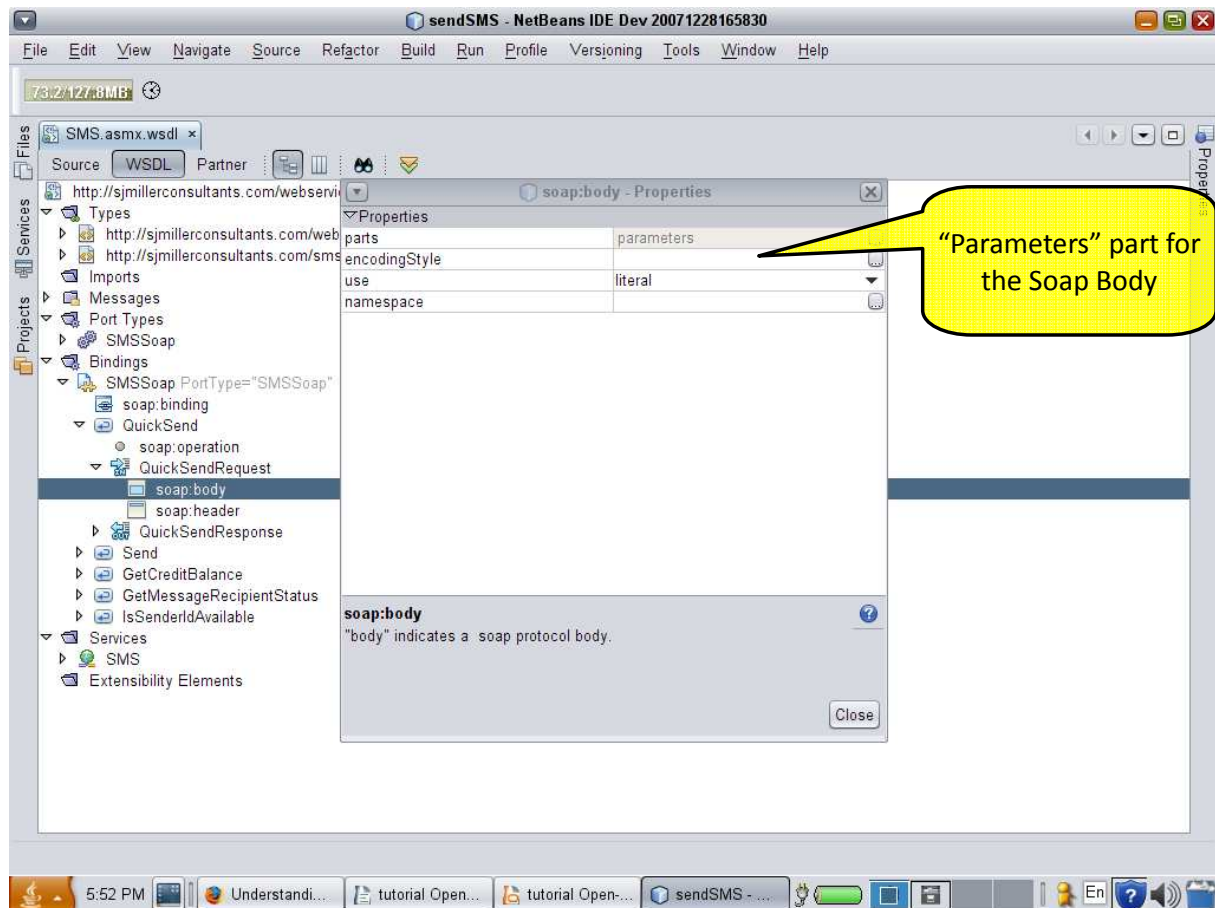
```
<wsdl:message name="QuickSendSoapIn">
  <wsdl:part name="parameters" element="tns:QuickSend" />
  <wsdl:part name="headerParameters" element="tns:AuthorizationHeader"/>
</wsdl:message>
```

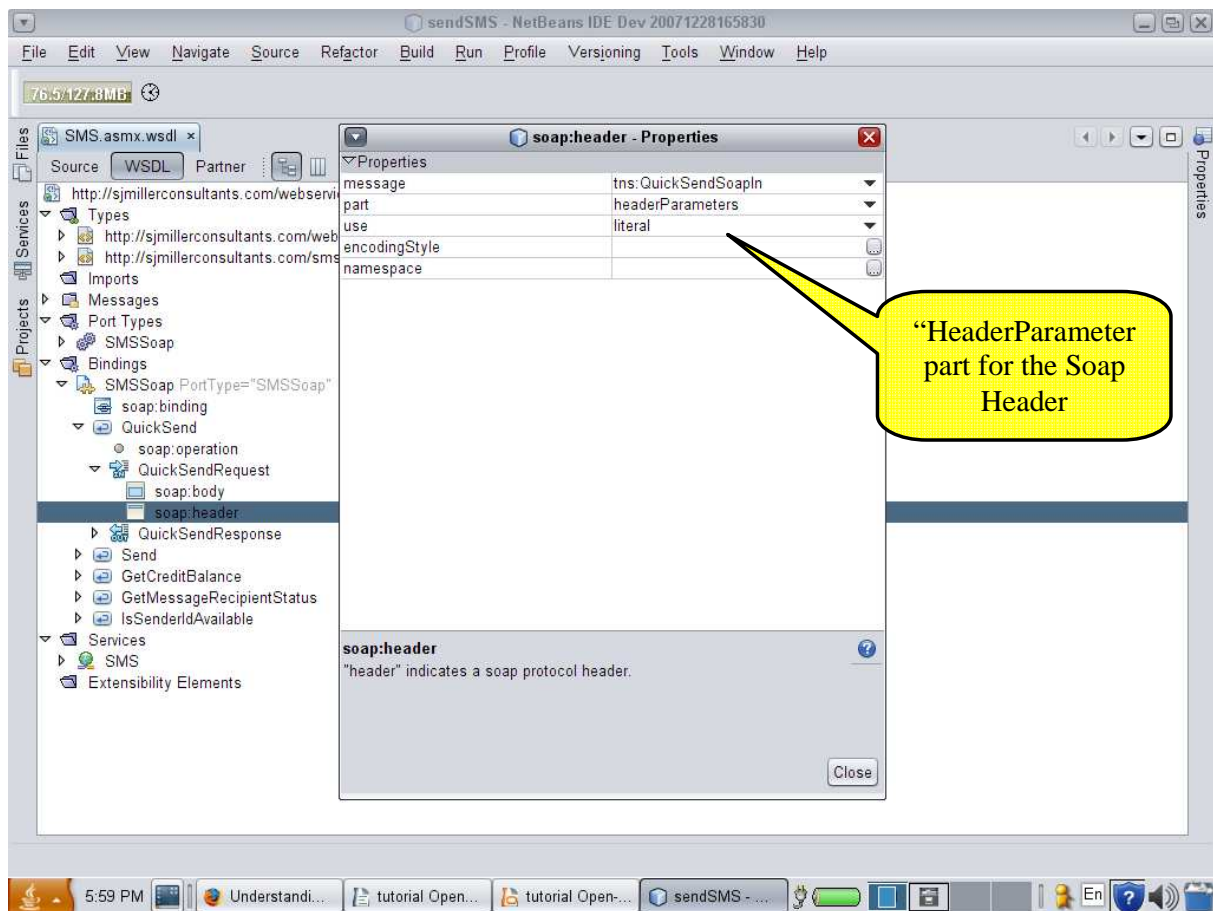
abcText WSDL Modification

Let's add a new part in the QuickSendSoapIn message

Since the part "HeaderParameter" contains the SessionKey, SMSSender component will be able to send it in the message SOAP to AbcText.

To finish up the WSDL modifications, we have just to assign "parameters" part to the Soap body and "headerParameters" part to Soap Header in the Soap Binding



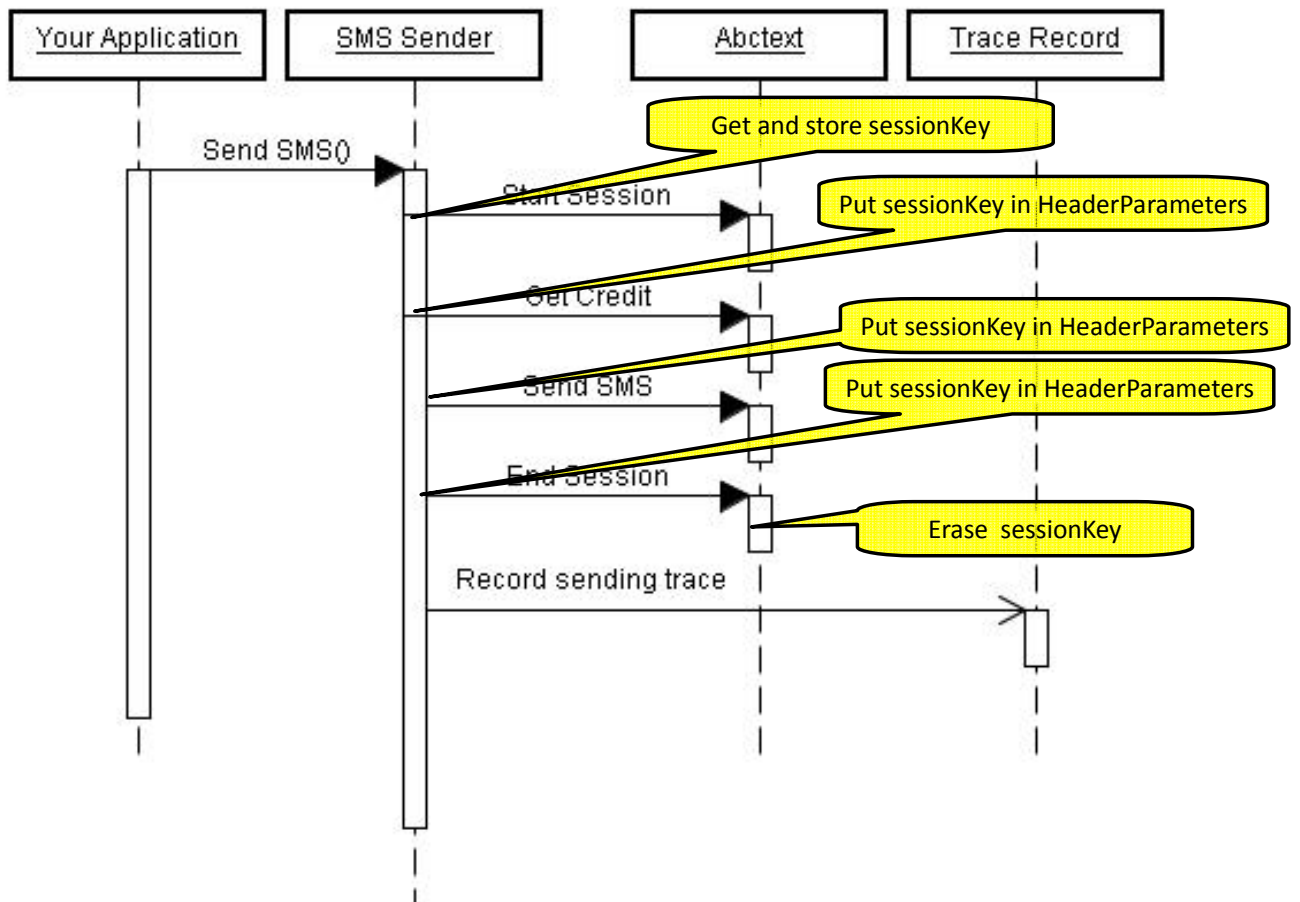


Impact on the service consumer

The requirement clearly says: *"composite application interface must be very simple and hide all protocol detail implementation to the end user"*. Since we added "HeaderParameter" part in the Quick send message, protocol dependencies appear in the composite application interface and the WSDL given to the client application is different from the original WSDL given by AbcText.

SMSSender is a Façade Pattern

Let's have a look on the diagram sequence below.



SMSSender is a stateful process orchestrator (BPEL) which is able to store sessionKey and insert it in the message part without impact on the end-user application. In this use case SessionKey management is assigned to SMSSender. By defining it as stateful process orchestrator, we hide sessionKey management and protocol detail to the service customer as required by the specification.

Conclusions

From our experience on the field, we noticed that the providers of Web Services often design their services for a specific platform or a specific language. Although this is far from a best practice for web services because it breaks the interoperability, this is a common practice.

In the example used for this tutorial, "AbcText" designed its WSDLs for low-level language applications. Since their abstract parts did not define Soap header parameters in, we were not able to use them in a JBI integration process "as is". Actually, Open-ESB does not provide an easy way to insert protocol information in the messages. In the future, it might be that WSDL appendix specifications will dedicate a section for protocol handling.

Since, we want to encapsulate AbcText services into our component, we had to redesign AbcText's WSDLs and insert an additional message part that holds header parameters. This redesign is hidden by SMSSender which acts as a stateful orchestrator and a façade pattern. Following this way, we are able to integrate AbcText's services in an Open-ESB environment.

About Pymma Consulting

Pymma Consulting is a technical architect bureau. Pymma provides expertise in the design and implementation of high performance Java EE and JBI (Open-ESB) architectures, with a high capacity of integration. Pymma was founded in 1999 and is a European company based in London (*headquarters*), with regional offices in Amsterdam and Paris. (contact@pymma.com or www.pymma.com)