SSE Project: BibleBraille Service

Luca Rinaldi

January 2016

Contents

1. Introduction. This section should clearly describe –with the help of a figure- the chosen orchestration. In particular which are the inputs and outputs of the orchestrator and of the service invocations, and how such inputs and outputs relate one another. The actual addresses of the employed remote services must be explicitly mentioned.
2. WS-BPEL implementation. 4.1 WS-BPEL processes. This section must clearly describe the main WS-BPEL process P, and it must contain readable figures illustrating (parts of) such processes. 4.2 Tests. This section must describe which tests were successfully executed. Such tests must include the case in which no fault was thrown, the case in which to_fault was thrown, and the case in which reply_fault was thrown.
3. Analysis of the WS-BPEL specification. This section should describe the workflow net modelling (the control flow of) P.

1 Introduction

The service developed is called BibleBraille, and as is name its goal is to provide a Braille translation and an audio version of specific verses of the Bible. More in detail this service have only the getVerse SOAP operation that as can be see in the WSDL file (BibleBrailleWSDL.wsdl) of the service take in input: - the name of the bible book (i.e. genesis). - the number of chapter of that book (i.e. 1). - the specific verse of the chapter (i.e. 1).

Than the service elaborate the response and send back as output: - the base64 binary representation of the braille image. - the link of a mp3 with the recording of the verse. - the original text of the verse.

the service to compute the result use three external services: - **BibleWebservice** (now on call A) a REST and SOAP bible service that retrieve a specific verse of the King James Version of the bible. The orchestrator use it to get the text of the requested bible verse, using the `GetBibleWordsByChapterAndVerse` SOAP operation, described in its WSDL (http://www.webservicex.net/BibleWebservice.asmx?WSDL) - **Text to Braille** (now on call B) a REST and SOAP service that convert a plain text in braille, returning a base64Binary string representing an image. The orchestrotor use its `BrailleText` SOAP operation decribe in its WSDL (http://www.webservicex.net/braille.asmx?WSDL) - **ESV Bible webService** (now on call C) a complete bible web service, that provide text and audio track of bible verse in Bible in contemporary English. The orchestrator use it only to get the link of audio track, using this REST resource: `http://www.esvapi.org/v2/rest/passageQuery.php/`

2 WS-BPEL implementation

The WS-BPEL of the service orchestrotor is composed in two parts, the **BibleBrailleComp** service, the real orchestration of the external service and the **BibleAudioProxy** a service proxy service for be able to asynchrounslly call the *ESV Bible webService* service, with doen't have this built in features. For now on we only speak about the main service, because this proxy doen't have any soffistecated logic, it only send back what receive by the bibleservice back to the orchestrator.

2.1 WS-BPEL processes

To achive the result the orchestrtor execute two part in parallel, the first one is the call of the service A to retrive the bible verse amd than invoke the service B to compute th braille conversion of the verse; in the second parallel part there is the asyncronous call of the service C to get the audio version of the verse and than wait for a callback.

The this last part we can have different outcome, in fact if the a callback message come before a timeout of 10 secons the service continous and the received message is analized, if not the all process is stopped and an to_fault is send back to the client. The message received by the proxy is then analysed and the url of the audio track is composed base on the id of the bible verse, if the verse serched is not found or other error shows up the process continous going but instead of

the audio url is send back an error. In this case the client receive only the text version and the braille version of the requested verse.

This behaivor it can me possible by the use of two scope, `ExternalScope` and `BibleAudioScope`, an two different event handler, in this way when the event `to_fault` is throw the hadler in the `ExternalScope` cath it and the all process is interrapted, opposite wjhen the `reply_fault` fault is throw the hadler in the `BibleAudioScope` catch it and so it doesn't interfer with the other parrallerl invokation of the service A and B.

2.2 Test

To test the BibleBraille Service four type of test are built, two with a correct result and one for the `to_fault` error and one with the `reply_fault` error. To prove correctnes the two test *correct1* and *correct2* are build with the following input:

```
<bib:getVerse>
  <book>genesis</book>
  <chapter>1</chapter>
  <verse>1</verse>
</bib:getVerse>
```

```
<bib:getVerse>
  <book>Luke</book>
  <chapter>9</chapter>
  <verse>1</verse>
</bib:getVerse>
```

this two test give the correct result of the bible verse, but unfortunately using the two used service have different version of the bible so the text is slightly different of the audio.

To simulate the `reply_fault` error another test is builded with the following input:

```
<bib:getVerse>
  <book>test</book>
  <chapter>23</chapter>
  <verse>22</verse>
</bib:getVerse>
```

Whit this input the service C return an error because does not exist any bible book called test, so the orchestrator throw the `reply_fault` error, and the output f the test is:

```
<m:getVerseResponse>
  <braille>9j/4AAQSkZJRgABAQEAYABgAAD/2wBDAAgGBgcGBQgHBwc....</braille>
  <audio>reply_fault: no audio avaible</audio>
```

```
  <text></text>
</m:getVerseResponse>
```

We can notise than the invokation of the service A and B are not stopped by the foult and the orchestrator give therir result, but actually that kind of verse is not found so the text tag is empty and the braille service return a empty image. Instead the foult fandler for the `reply_fault` add a string in the audio tag and so we can report to the client that no audio track are found for that verse.

The check fo the `to_fault` is more complicated because it have to be simulated that no answare from the proxy is received after in 10 seconds. To simulate this behainvor, was added a wait command inside the proxy BPEL that wait more than 10 seconds before invoke the callback. In this way the output of one of the before test is:

```
<SOAP-ENV:Fault>
  <faultcode>SOAP-ENV:Client</faultcode>
  <faultstring>to_fault</faultstring>
  <detail>
    <error>to_fault: request timeout</error>
  </detail>
</SOAP-ENV:Fault>
```

As we can see in this case all the orchestration process is stopped and an error message is send back to the client.

3 Analysis of the WS-BPEL specification