SSE Project: BibleBraille Service

Luca Rinaldi

January 2016

Contents

1 Introduction

BibleBraille is a service with the goal of providing Braille and an audio version of the the Bible verses.

More in detail this service provide the `getVerse` SOAP operation that, as written in the `BibleBrailleWSDL.wsdl` file, take as input the following parameter:

- the name of the bible book (i.e. genesis).
- the number of the chapter (i.e. 1).
- the specific verse (i.e. 1).

Than the service elaborate the response and send back to the client the following output:

- the base64 binary representation of an image, in witch there is the braille conversion of the bible verse.
- the URL link of an mp3 file in witch there is the recording of the verse.
- the original text version of the verse.

This service use the following three external service to compose the response, by REST and SOAP interface:

- **BibleWebservice** (now on call B) a REST and SOAP bible service that retrieve a specific verse of the King James Version of the bible. The

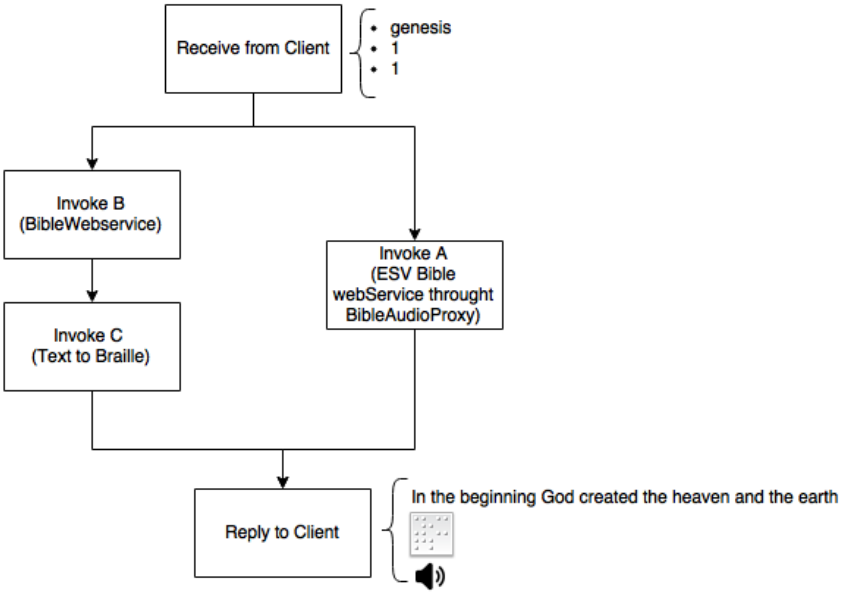

Figure 1: BibleBraille

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 C) a REST and SOAP service that convert a plain text in braille, returning a base64Binary string representing of an image. The orchestrator use its `BrailleText` SOAP operation describe in its WSDL (http://www.webservicex.net/braille.asmx?WSDL).
- **ESV Bible webService** (now on call A) a complete bible web service, that provide text and audio track of the *Contemporary English Bible*. The orchestrator use it only to get the link of the audio track, using this REST resource: `http://www.esvapi.org/v2/rest/passageQuery.php/`.

2 WS-BPEL implementation

The WS-BPEL of the service orchestrator is composed in two parts, the **BibleBrailleComp** service, the real orchestration of the external service and the **BibleAudioProxyComp** a service proxy use to asynchronously call the *ESV Bible webService* service, which doesn't have those built in features.

The proxy actually doesn't do nothing special but send back all the response of the service A, indeed all the logic needed to check the received message is moved in the main service, to avoid confusion.

The only crucial fact is that actually in the callback of the proxy there is also add the input, this is because it will be used in the main service to create a correlation between the async invocation and the callback.

2.1 WS-BPEL processes

To achieve the result the orchestrator execute two part in parallel, the first one is the call of the service B to retrieve the bible verse and than invoke the service C to compute th braille conversion of that verse; in the second parallel part there is the asynchronous call of the service A to get the audio version of the verse.

On this last part we can have different outcome: if the a callback message come after a timeout of 10 seconds the service throw a `to_fault` error and all the process is stopped and the fault is also send back to the client; if the callback message come before the timeout, the message is analyzed and the url of the audio track is composed base on the id of the bible verse. Alter the analysis if the verse a error shows up a `reply_fault` is throw, but this time the process continuous going and the only change is that instead of the audio url is send back a error. So in this case the client receive only the text version and the braille version of the requested verse.

This behavior is implemented by the use of two scope and different fault handler. Indeed there is the `ExternalScope` scope, that cover boat the flow sequence, and the `BibleAudioScope` scope that cover only the flow sequence with the A invocation. So when the `to_fault` fault is throw the fault handler in the `ExternalScope` couch it and the execution of all process is interrupted, in the other case when the `reply_fault` is throw the fault handler in the `BibleAudioScope` catch it and so it doesn't interfere with the other parallel invocation of the service A and B.

2.2 Test

To test the BibleBraille Service four type of test were design, two with a correct result and one for the `to_fault` error and one for the `reply_fault` error. To prove correctness the correctness two test *correct1* and *correct2* were 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>
```

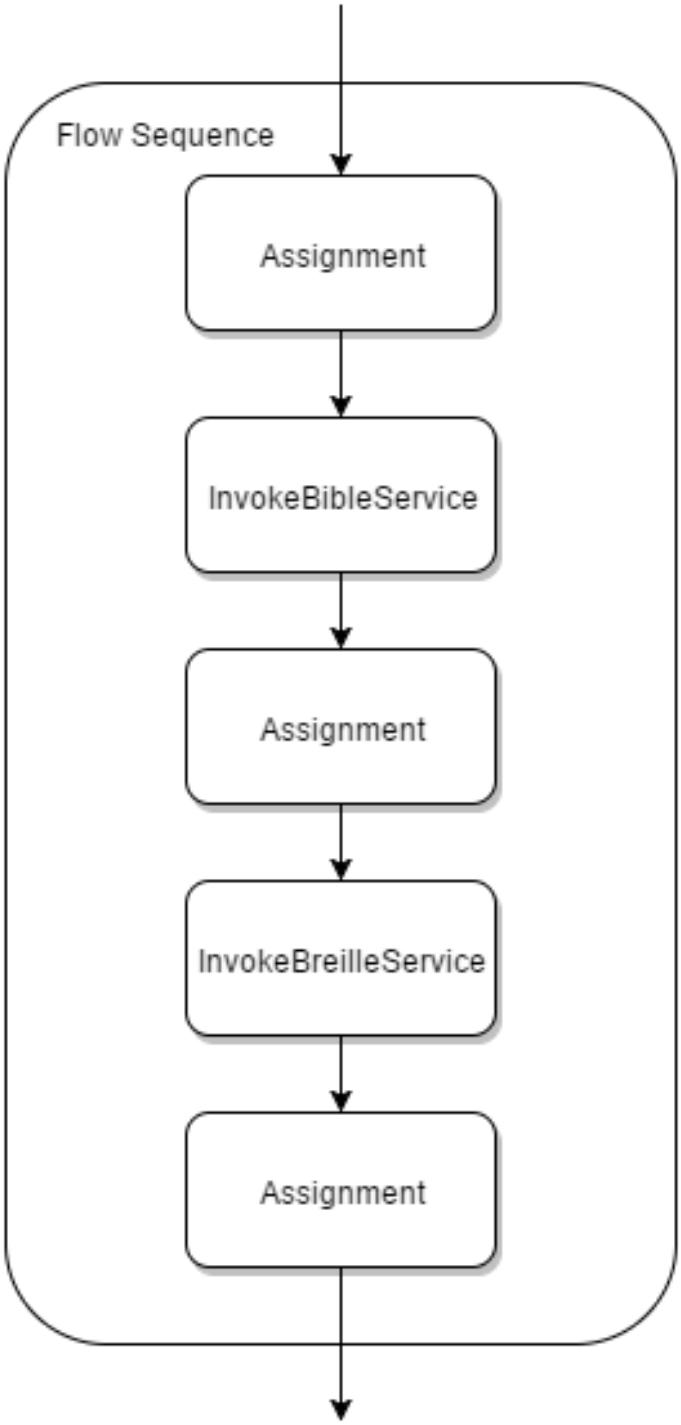

Figure 2: FlowSequece1

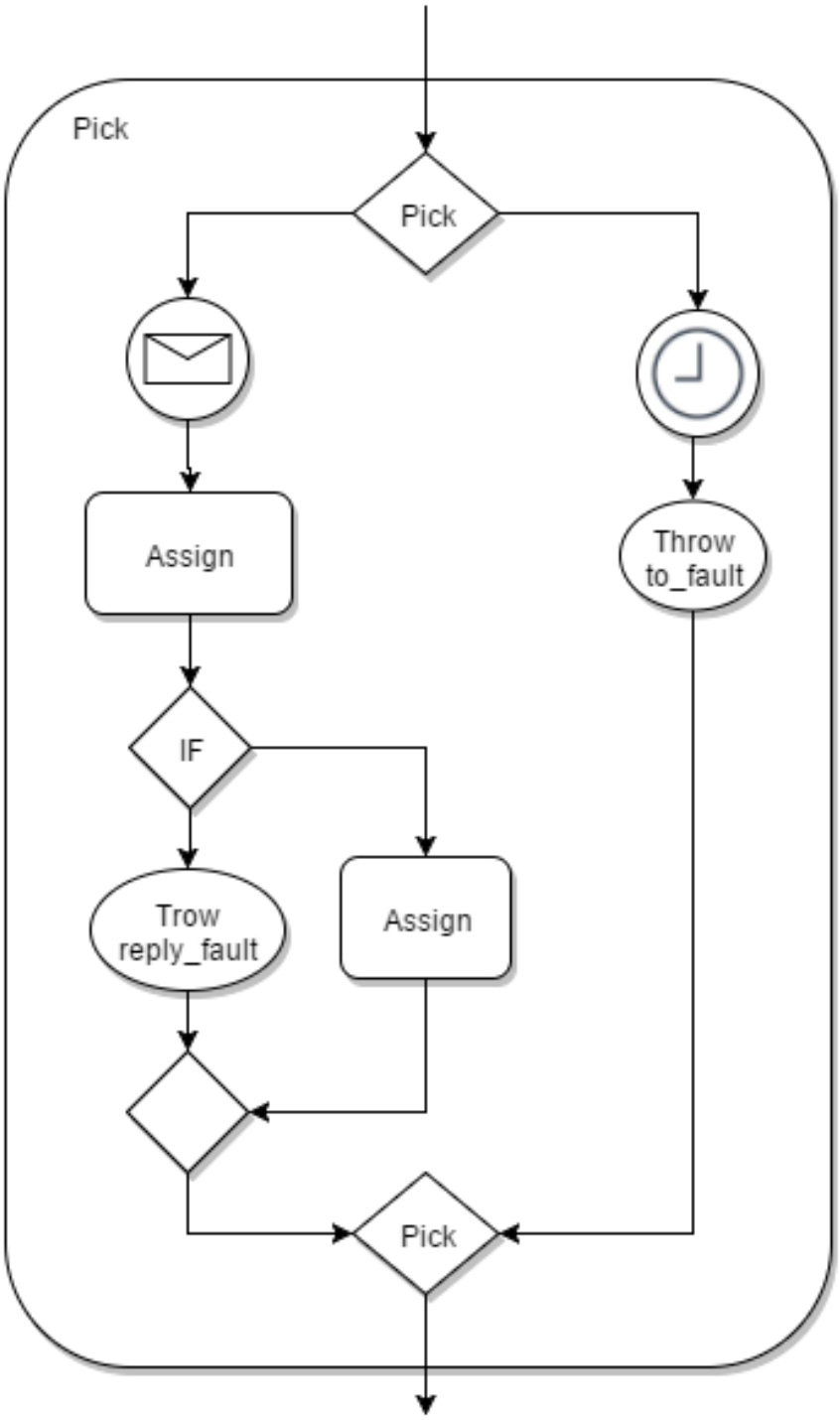

Figure 3: FlowSequece2

```
  <verse>1</verse>
</bib:getVerse>
```

this two test give the correct result of the bible verse, but unfortunately the we use service with different translation of the Bible so the meaning is the same, but the audio version of the verse is slightly different of the text and the braille version.

Instead to simulate the `reply_fault` error the following input is used, actually any input with a wrong bible book, chapter or verse can do the same:

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

We can see that in this case 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 is:

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

We can notice than the invocation of the service A and B are not stopped by the fault and the orchestrator give their result, but actually that kind of verse is not found so the text tag is empty and the braille service return a empty image.

The most difficult one is the `to_fault` because in this case we have to simulate a delay in the async answer of the proxy in such a wait the timeout if 10 second is triggered. For this reason by adding a wait command in the WS-BPEL of the proxy before the invocation we can have the following output:

```
<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

The control flow of the main WS-BPEL process is also implemented by a workflow net, and thanks to the WoPeD software it is also checked if it sounds. Actually this doesn't mean that the orchestrator does not have any problem and always terminate correctly, but it is never than less a good result.

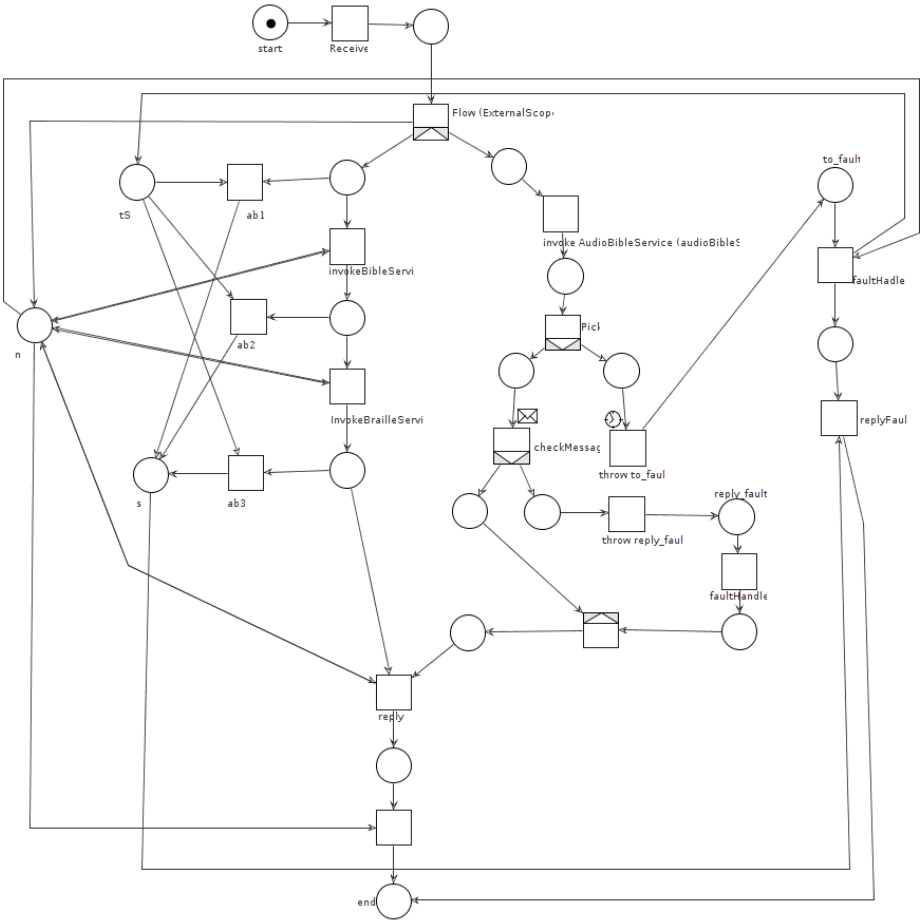

Figure 4: workflow net

The more difficult part in the implementation of the workflow net is the error handling of the two kind of fault. Actually the `reply_fault` is handled easily because it doesn't have to different this of the normal reply without fault, so an XOR join transition is added at the end of the flow, in such a way the flow correctly end either if the message received is corrected or if the `reply_fault` is throw.

In the other hand the `to_fault` have to interrupt the executing of the other parallel flow in any situation, either if no invocation are done yet, if only the invocation to service B or boat invocation are done.

To achieve this goal are added three new place called `n` (filled when no error occur), `tS` (filled when the stop procedure start) and `s` (filled when the scope stop is execution), and also three new transition, `ab1`,`ab2`,`ab3`, connected to the place related at the original scope activity. So when a token is placed in `tS` the scope stop the normal execution and the token present in the scope are absorbed. After the token is absorbed the place `s` is fill and then the fault handler can end is execution, and an error message is send back to the client.