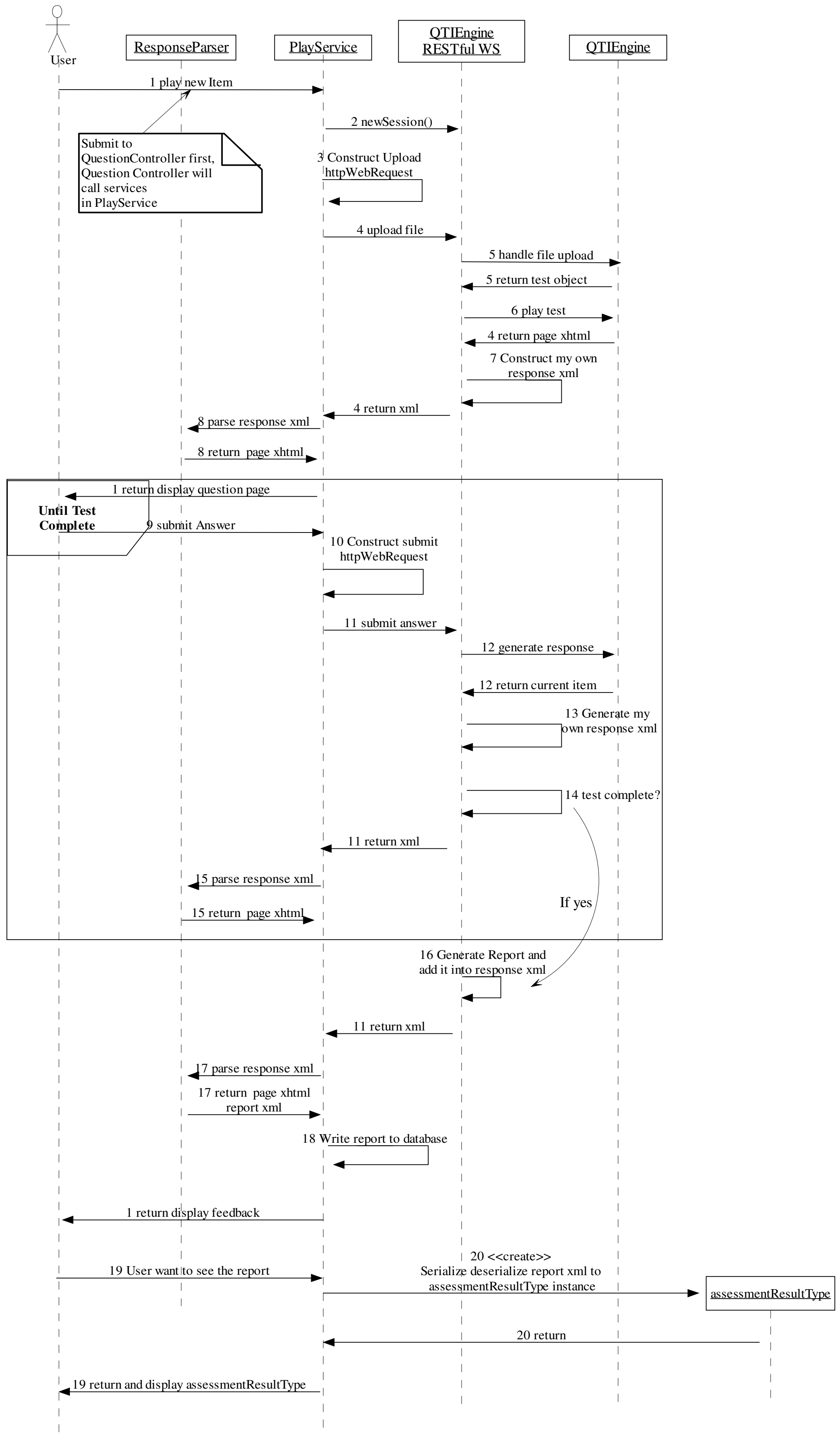
User
ResponseParser
PlayService
QTIEngine RESTful WS
QTIEngine
1 play new Item
Submit to QuestionController first, Question Controller will call services in PlayService
2 newSession()
3 Construct Upload httpWebRequest
4 upload file
5 handle file upload
5 return test object
6 play test
4 return page xhtml
7 Construct my own response xml
4 return xml
8 parse response xml
8 return page xhtml
1 return display question page
Until Test Complete
9 submit Answer
10 Construct submit httpWebRequest
11 submit answer
12 generate response
12 return current item
13 Generate my own response xml
14 test complete?
11 return xml
15 parse response xml
15 return page xhtml
If yes
16 Generate Report and add it into response xml
11 return xml
17 parse response xml
17 return page xhtml report xml
18 Write report to database
1 return display feedback
19 User want to see the report
20 <<create>> Serialize deserialize report xml to assessmentResultType instance
assessmentResultType
20 return
19 return and display assessmentResultType

Explanation

1. http://lslvm-pz3.ecs.soton.ac.uk:81/Question
   User clicks the "play" button for a test. The request will be forward to:
   http://lslvm-pz3.ecs.soton.ac.uk:81/Question/Play/19
   This url means QuestionController, play action (method), 19 is an id parameter. After processing the request, the Play method in QuestionController.cs will call playTest method in PlayService.cs.
2. ~/Services/PlayService.cs, newSession() method calls RestController.groovy, newSession() method API
   See Deployment.docx Section 3 QTIEngine Server for detail.
3. ~/Services/PlayService.cs, `public string Upload(HttpContext context, Question question)`
4. RestController.groovy, upload() method
5. In upload() method in RestController.groovy,
   For test package, applicationService.handleZip()
   Return a ContentPackage instance presents the test
6. RestController.grooy, playTest() method. This method call testService.getRenderedPage() method in QTIEngine.
   The return is xhtml page as a string
7. If no other parameters are detected (report, exit or test finished), playTest() calls RestService.createNormalResponse() method to create the response xml. This response only contains the page xhtml of the next question. If this is the last question of the test, the process will jump to 16.
8. QuestionController.cs, `private ActionResult parseResponse(string responseString, int id)`
   This method uses services of ResponseParser.cs
9. User click the submit button to submit the answer of an item to ~/Controllers/QuestionController, `public ActionResult PlayTest(int id, FormCollection collection)`
10. ~/Services/PlayService.cs, `public string playTest(HttpContext context, Test test). Write the form data and variables in query string into httpWebRequest.`
11. Send the httpWebRequest to RestController.grooy's playTest() method.
12. Similar to 6
13. Similar to 7
14. In RestController.grooy, playTest() method. Corrdinator.getTestController().isTestComplete()
15. Similar to 8

9-15 will be loop until the test is complete (14 returns true)

16. In RestController.grooy, report() method and RestService.grooy's createFinishReponse() method. Report can be generated any time during the test, but the candidate response of a item will be null if this item has been answered.
17. In QuestionController.cs, `private ActionResult parseResponse(string responseString, int id)` and ResponseParser.cs. Parse the final response xml and return the final xhtml page.
18. In PlayService.cs, `public void SaveTestResult(string reportString, Test test, DB db)`. Save the test result into database.

19. If a user clicks the “report” link in ~/UserQuestion/Index.aspx page, the request will be sent to `public ActionResult TestDetail()` in UserQuestionController.cs.
20. In TestDetail() method, the report xml will be read from database first. Then we deserialize the report xml into assessmentResultType class instances. Then the assessmentResultType will be displayed on TestDetail.aspx. Please refer to section 2.4 in deployment.aspx for detail.