Report for Data Integration project by Ayman YACHAOUI

The program I have written takes an aribitrary number of WS's provided with their definitions and realizes joins based on arbitrary number of variable matching. It is aimed to be as generic as possible and often trades performance for the completeness of the view as will be discussed. What follows is a running example of my implementation and source code comments on its details. To demonstrate its abilities, let's run through this demonstration query:

mb_getArtistInfoByName("skrillex", ?id, ?b, ?e)
#mb_getReleasesByArtistId(?id,?asin,?releaseid,?title,?date)
#mb_getReleasesByArtist(?id,?asin, ?title, ?date)
#mb_getTrackByReleaseId(?tracktitle,?duration,?trackid,?releaseid)

This query is composed of 4 steps:

1. By entering an artist name, the artist id (?id), her birth date (?b) and death date (?e) are returned.

2. The (?id) variables are cascaded down to get the albums produced by the artist. The album ASIN number (?asin), the album id (?releaseid), the album title (?title) and date of release (?date) are output.

3. The third step is redundant and used to demonstrate ability to match several variables from preceding outputs and returns the same data as for the 2nd step.

4. Using the album id (?releaseid), the 4th step outputs all the songs within it with their titles (?tracktitle), song duration (?duration), a song id (?trackid).

1 WS definitions

The .xsl definitions schema is unaltered when each .xml definition is articulated in 4 sections.

- `<headvariables/> , <call/> and <transformation/>`

  as defined and used by Pr Preda

- `<partialmerge/>`

  used in the merging of XML subcalls as discussed in section.

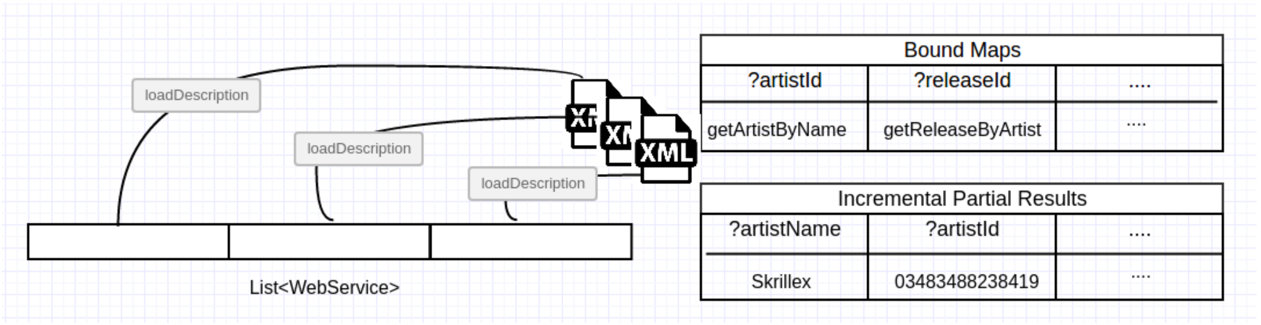
As an illustration, here's the definition for mb_getReleasesByArtist:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ws>

 <prefix name="w" value="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
 <prefix name="y" value="http://mpii.de/yago/resource/"/>

 <headVariables>
        <variable type="inout" name="?artistid"/>
        <variable type="inout" name="?asin"/>
        <variable type="inout" name="?title"/>
        <variable type="output" name="?type"/>
        <variable type="output" name="?date"/>
    </headVariables>

 <definition>
 </definition>

 <call>
  <part type="constant" value="http://musicbrainz.org/ws/1/release/?artistid="/>
  <part type="input" variable="?artistid" class="singer" separator="+" />
  <part type="constant" value="?asin="/>
  <part type="input" variable="?asin" class="uk" separator="+" />
  <part type="constant" value="?title="/>
  <part type="input" variable="?title" class="uk2" separator="+" />
 </call>

 <transformation file="mb_getReleasesByArtistId.xsl"/>

 <partialmerge container_tags="metadata,release-list"/>

</ws>
```

2 Query plan

2.1 Class variables

The main static structures maintained throught the engine processing of a query are :



?artistId	?releaseId	....
getArtistByName	getReleaseByArtist	....

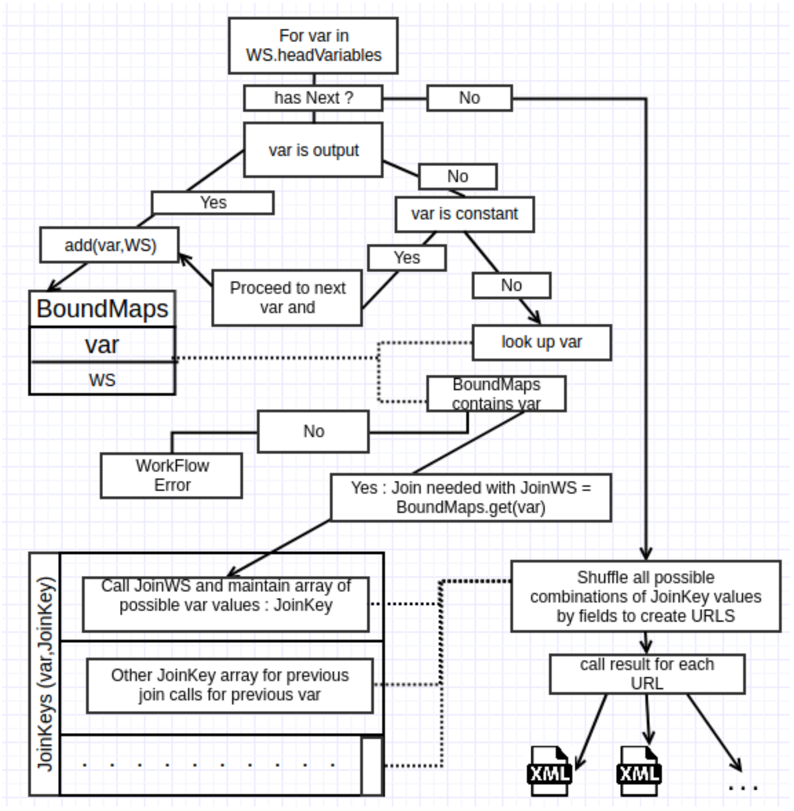
?artistName	?artistId	....
Skrillex	03483488238419	....

- An **Array** of web services is filled using the *loadDescription* method using the .xml descriptions.
- For every webservice varible, an entry is created in the **BoundMaps** TreeMap which references the WS needed for an eventual join.
- At the end of each step of the query, an ArrayList of String[ ] **incrementalPartialResults** is created storing the result rows that match the current step so far in the query.

2.2 Execution flow

2.2.1 Getting the data

For each step, we call the WS with the appropriate URLs, the following chart sums up this process:


For var in WS.headVariables
has Next ?
No
var is output
Yes
No
var is constant
add(var,WS)
Yes
No
Proceed to next var and
BoundMaps
var
WS
look up var
BoundMaps contains var
No
WorkFlow Error
Yes : Join needed with JoinWS = BoundMaps.get(var)
JoinKeys (var,JoinKey)
Call JoinWS and maintain array of possible var values : JoinKey
Other JoinKey array for previous join calls for previous var
Shuffle all possible combinations of JoinKey values by fields to create URLS
call result for each URL
XML
XML
...

It is worth noting that in the case of joining several columns for a single WS, the URLs are generated so as to cover every possible combination of needed field values. One could note that this dramatically slows down the data getting phase since it's going to collect mostly empty call results but this ensures that **join columns are independent**. And can even come from different API's !

2.2.2 Joining the data

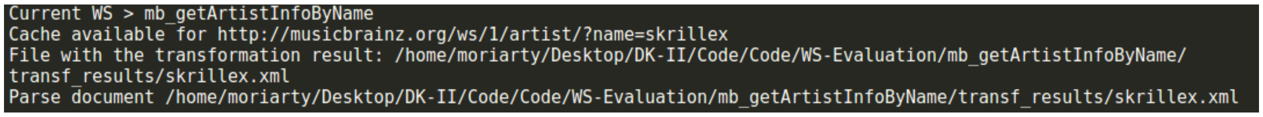
We then proceed to filling **incrementalPartialResults**. This is done iteratively at the end of *Getting the data* phase of each WS. We distinguish 2 starting points:

- The WS calls resulted in several XML partial calls, we merge them into a single one, which brings us to the second starting point.
- The WS call resulted in a single xml of call result, we transform it using *getTransformationResult*

The transformed file is **listOfTupleResults** that is used to join **incrementalPartialResults**. For every row1 in **incrementalPartialResults**, its fields are compared to every row2 in **listOfTupleResults**, if they match, row2 is appended to row1 and maintained in a newer version (thus incremental) PartialResults.

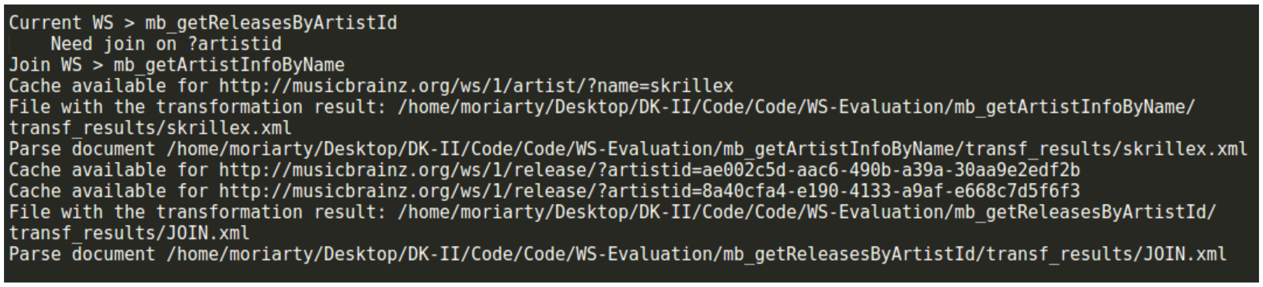
3 Running example

We comment here some of the console output that tracks the execution of the query plan:
The first webservice **GetArtistByName** results in a single .xml call result.



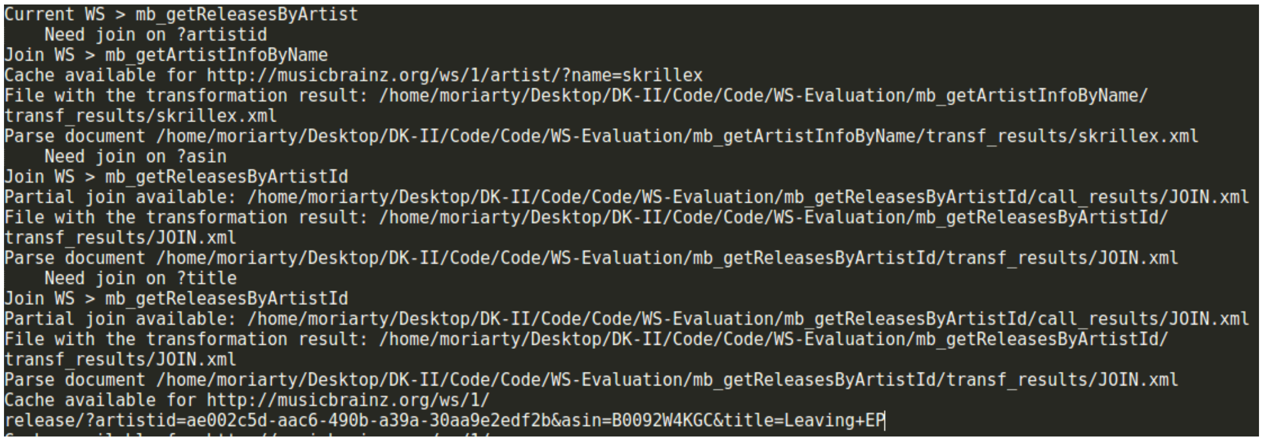
```
Current WS > mb_getArtistInfoByName
Cache available for http://musicbrainz.org/ws/1/artist/?name=skrillex
File with the transformation result: /home/moriarty/Desktop/DK-II/Code/Code/WS-Evaluation/mb_getArtistInfoByName/
transf_results/skrillex.xml
Parse document /home/moriarty/Desktop/DK-II/Code/Code/WS-Evaluation/mb_getArtistInfoByName/transf_results/skrillex.xml
```

After the result of the first webservice are joined with the (intially empty thus appended) **incrementalPartialResults**, The second webservice **GetReleaseByArtistId** is called. It needs the *?artistid* field which it gets from calling the transformed result from WS1. Two artist go by that name and thus spawn two URL calls. These will be merged than transformed and be a basis if WS2 is called for its results later before being joined to the existing **incrementalPartialResults**, maintaining only valid rows:



```
Current WS > mb_getReleasesByArtistId
    Need join on ?artistid
Join WS > mb_getArtistInfoByName
Cache available for http://musicbrainz.org/ws/1/artist/?name=skrillex
File with the transformation result: /home/moriarty/Desktop/DK-II/Code/Code/WS-Evaluation/mb_getArtistInfoByName/
transf_results/skrillex.xml
Parse document /home/moriarty/Desktop/DK-II/Code/Code/WS-Evaluation/mb_getArtistInfoByName/transf_results/skrillex.xml
Cache available for http://musicbrainz.org/ws/1/release/?artistid=ae002c5d-aac6-490b-a39a-30aa9e2edf2b
Cache available for http://musicbrainz.org/ws/1/release/?artistid=8a40cfa4-e190-4133-a9af-e668c7d5f6f3
File with the transformation result: /home/moriarty/Desktop/DK-II/Code/Code/WS-Evaluation/mb_getReleasesByArtistId/
transf_results/JOIN.xml
Parse document /home/moriarty/Desktop/DK-II/Code/Code/WS-Evaluation/mb_getReleasesByArtistId/transf_results/JOIN.xml
```

The third web-service **GetReleaseByArtist** outputs the same data as WS2 and is used here to demonstrate the join key shuffling that occurs. Indeed **GetReleaseByArtistId** needs as input the *?asin* and *?title* which it gets from WS2 for which they are rather defined as outputs.



```
Current WS > mb_getReleasesByArtist
    Need join on ?artistid
Join WS > mb_getArtistInfoByName
Cache available for http://musicbrainz.org/ws/1/artist/?name=skrillex
File with the transformation result: /home/moriarty/Desktop/DK-II/Code/Code/WS-Evaluation/mb_getArtistInfoByName/
transf_results/skrillex.xml
Parse document /home/moriarty/Desktop/DK-II/Code/Code/WS-Evaluation/mb_getArtistInfoByName/transf_results/skrillex.xml
    Need join on ?asin
Join WS > mb_getReleasesByArtistId
Partial join available: /home/moriarty/Desktop/DK-II/Code/Code/WS-Evaluation/mb_getReleasesByArtistId/call_results/JOIN.xml
File with the transformation result: /home/moriarty/Desktop/DK-II/Code/Code/WS-Evaluation/mb_getReleasesByArtistId/
transf_results/JOIN.xml
Parse document /home/moriarty/Desktop/DK-II/Code/Code/WS-Evaluation/mb_getReleasesByArtistId/transf_results/JOIN.xml
    Need join on ?title
Join WS > mb_getReleasesByArtistId
Partial join available: /home/moriarty/Desktop/DK-II/Code/Code/WS-Evaluation/mb_getReleasesByArtistId/call_results/JOIN.xml
File with the transformation result: /home/moriarty/Desktop/DK-II/Code/Code/WS-Evaluation/mb_getReleasesByArtistId/
transf_results/JOIN.xml
Parse document /home/moriarty/Desktop/DK-II/Code/Code/WS-Evaluation/mb_getReleasesByArtistId/transf_results/JOIN.xml
Cache available for http://musicbrainz.org/ws/1/
release/?artistid=ae002c5d-aac6-490b-a39a-30aa9e2edf2b&asin=B0092W4KGC&title=Leaving+EP
```

This process can go on indefinitely for an arbitrary number of WS's. After the 4th and last WS is processed, the results (which are stored in **incrementalPartialResults**) are printed in tabular form.