Query Engine for Web Service Compositions

Université Paris-Sud
Télécom ParisTech

Anderson Carlos Ferreira da Silva

1. Introduction

Web Services are Web Applications to exchange data between different systems that may or may not be in distributed environments (XML Web Services, n.d.). The retrieval of information from Web Services is not particularly difficult but depending on the Web Service, some effort is necessary to perform calls. A call is a request to a Web Service, this task is more difficult if the target Web Service has some constraints such as the number of calls per second.

A Web Service Composition can be understood as a collection of calls to Web Services to achieve a request (Claro, Albers, & Hao, 2006). This report has the objective of presenting an implementation of a Query Engine for Web Service Compositions.

2. Materials and Methods

2.1. MusicBrainz

MusicBrainz is a public, open and free online music encyclopedia (MusicBrainz - The Open Music Encyclopedia, n.d.). For this project, we chose a Web Service provided by MusicBrainz that manipulates its own database via HTTP. The architecture of this Web Service follows the REST principles and all content is served in XML (Development - MusicBrainz, n.d.).

To create a Web Service Composition, we chose four (4) resources on MusicBrainz Web Service that represent entities on MusicBrainz Database: artist[1], release[2], recoding[3] and label[4]. Each entity contains several attributes and some of them allow joins between entities. For this work, we took a sample of four (4) attributes of each entity and we transformed them in variables, the variables present in Table 1 allow a Web Service Composition.

Resource	Variable			
artist	?artistId	?artistName	?beginDate	?endDate
release	?albumId	?albumName	?releaseDate	?artistId
recording	?songId	?songName	?duration	?albumId
label	?labelId	?labelName	?beginDate	?albumId

Table 1. The variables used to map part of entity attributes.

The URL to access the MusicBrainz Web Service is: https://musicbrainz.org/ws/2/. To perform a call, it is mandatory to append the resource name but the arguments are optional, for example: https://musicbrainz.org/ws/2/release?label=47e718e1-7ee4-460c-b1cc-1192a841c6e5.

2.2. Maven

Apache Maven is a software for project management that facilitates the management of dependencies for Java Projects. It was chosen to manage all resources on this project.

3. Implementation

Our implementation approach conducts several integrity verifications before executing a query and adds some mechanisms to retry if the execution fails at some point. To simplify our implementation, we only manipulated *String*s and stored them in Java collections.

[1] An entity that produces audio-visual material (https://musicbrainz.org/doc/Artist)
[2] Release of a product on a specific period and region (https://musicbrainz.org/doc/Release)
[3] An audio that can be released as track (https://musicbrainz.org/doc/Recording)
[4] Organization engaged to marketing and publish music recordings (https://musicbrainz.org/doc/Label)

3.1. Query

The input of the Query Engine is a *String* and takes the following format:

P(?title, ?year)<-getInfo rmation(Name, ?id, ?year)#getDetail(?id, ?title, ?job)

The query is composed of two parts: a *head* and a *body*. The *head* and the *body* are separated by two symbols: "<-". If the query does not contain these symbols or if the query does not contain a *head* and a *body* an error will be raised.

The *head* and the *body* of a query are composed of *expressions*. The *head* contains only one *expression* and *body* contains a least one *expression*. An *expression* is composed by a *function name* and its *arguments*. An *argument* is a *variable* if it starts with a *question mark* ("?"), otherwise, is a *constant*. Table 2 shows how the inputs of a Query Engine are stored in memory.

Input	Example	Data Structure
Query	*p(?x, ?y)<-f(C, ?x)#g(?x, ?y)*	Map.Entry<Expression, List<Expression>>
Query Separator	<-	
Query Head	*p(?x, ?y)*	Expression
Query Body	*f(C, ?x)#g(?x, ?y)*	LinkedList<Expression>
Function Separator	#	
Function	*f(C, ?x)*	Expression
Function Name	*F*	String
Function Arguments	*(C, ?x)*	LinkedList<Element>
Argument Separator	*,*	
Constant	*C*	Element
Variable	*?x*	Element

Table 2. The data structures that hold the inputs.

3.1.1. Query Integrity

To check for the query integrity, the following checks were added to the source code:

- at least one query separator ("<-") but not more than one
- at least one *argument* for the *expression*
- the *head* and *body* of query are present
- at least one *variable* on each *expression* has to match (Cartesian Product is not allowed)
- the number of *arguments* in the *expression* is the same from the Web Service description
- all *variables* in an *expression* cannot have the same name

If one of these conditions is not met, an error is raised.

3.2. Executor

If the query is valid and well formed, the executor of the Query Engine takes action. For the first *expression* on the *body* of the query, the executor creates a *relation* by calling the Web Service. For any subsequent *expression*, the executor performs a join operation, rembering that Cartesian Product is not allowed here for a Web Service Composition. The join operation is performed looking at the attributes (*elements*) in common between the current *relation* and the current *expression*. It is also possible to perfom a selection during the join operation. In this case, the *argument* is not an input for the *function* but a condition to keep or discard the results (*tuples)* served by the Web Service. If an error is raised during the execution, an implementation of Exponential Backoff Algorithm controls the flow of calls to a Web Service. The final operation is retried.

4. Results

We tested our Query Engine passing the following query as argument:

```
P(Frank Sinatra, ?an, ?st, ?ln)<-
        mb_getArtistInfoByName(Frank Sinatra, ?id, ?ab, ?ae)#
        mb_getAlbumByArtistId(?id, ?ar, ?aid, ?an)#
        mb_getSongByAlbumId(?aid, ?sd, ?sid, ?st)#
        mb_getLabelByAlbumId(?aid, ?lid, ?ln, ?lb)
```

This query retrieves all songs and albums of any artist that contains the names “Frank Sinatra”. The result was 918 tuples in less than 2.5 seconds using less than 1 MB (Megabyte) of memory. The following excerpt shows how the query result is presented:

```
(Frank Sinatra; Sing and Dance With Frank Sinatra; The Continental; Columbia)
(Skank Sinatra; Barcoded; How Deep; RMD Recordings)
```

5. Difficulties

To implement this Query Engine, we faced some challenges to perform calls, retrieve and process data from Web Services. We found some errors such as the wrong number of inputs and some content served in different formats rather than XML. Until we corrected them on our code, we could not perform calls to Web Services and apply some operations such as joins on the results. Another issue was the encoding of the file served by MusicBrainz. It uses UTF-8 with support for miscellaneous symbols. However, on Microsoft Windows Distributions, without the correct character set and encoding, characters such as *em dash (“–”)*, or *long dash* are not correctly interpreted. Without a modification on how the files are persisted, our Query Engine cannot read the content served by some Web Services on Microsoft Windows Distributions.

6. Conclusion

This project allowed us to understand how to create a call composition and how to execute it. It was not difficult to create the conjunctive queries but evaluate and execute them was quite hard without a query optimizer. For this project, we only implemented a query executor and query validator that does not allow Cartesian Product. The results show that our Query Engine works well and is operational. A possible extension of this project is the implementation of a query optimizer.

7. Bibliography

Claro, D. B., Albers, P., & Hao, J.-K. (2006). Web services composition. In *Semantic Web Processes and Their Applications* (pp. 205-234). Springer. Retrieved from https://pdfs.semanticscholar.org/60a4/0b6ee9ad79e4a4e4e20cb29575683169411c.pdf

Development - MusicBrainz. (n.d.). Retrieved from MusicBrainz: https://musicbrainz.org/doc/Development

Maven – Welcome to Apache Maven. (2018, 02 10). Retrieved from Apache: https://maven.apache.org/

MusicBrainz - The Open Music Encyclopedia. (n.d.). Retrieved from MusicBrainz: https://musicbrainz.org/doc/Development/XML_Web_Service/Version_2

XML Web Services. (n.d.). Retrieved from W3Schools : https://www.w3schools.com/xml/xml_services.asp