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 system in distributed environments or not (XML Web Services, n.d.). To retrieve information from Web Services is not painful in nowadays. However, depending of the Web Service a minimum effort it's necessary to perform calls. A call is a request to Web Service, this task more difficulty if the target Web Service has some constraints such as the number of calls per second.

A Web Service Composition can be understood as 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, free and 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.

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 is mandatory to append the resource name and the arguments are optional, example:

https://musicbrainz.org/ws/2/release?label=47e718e1-7ee4-460c-b1cc-1192a841c6e5

A rich documentation explaining how to use MusicBrainz Web Service is available on MusicBrainz Website (https://musicbrainz.org/doc/Development).

2.2. Maven

Apache Maven, commonly called Maven, is a software for project management and comprehension (Maven – Welcome to Apache Maven, 2018). To make the management of dependencies for Java Projects easier. Maven was chosen to manage all resources to this project.

[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. Implementation

This section describes details of our implementation such as the inputs and data structures used. Our approach realizes several integrity verifications before to execute a query and adds some mechanisms to retry if the execution fails in some point. To simplify our implementation, we only manipulate *String*s and we store them in Java collections.

3.1. Query

The input of 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 by two parts a *head* and a *body*. The *head* and the *body* are separated by two (2) 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 in both cases.

The *head* and the *body* of a query are composed by *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* can is a *variable* is starts with a *question mark* ("?"), otherwise is a *constant*. The Table 2 summarizes the how store the inputs of a Query Engine.

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 to hold the inputs.

3.1.1. Query Integrity

Consitency1 raise error if there is not "<-" symbols or more than one. Consitency2 raise error if there empty clausules ",". Consitency3 raise error if the query doesn't have head. Consitency4 raise error if the query doesn't have body. Consitency5 Cartesian Product is not allowed (raise error if there is no join (any variable match). Consitency6 raise error if body of the expression has less or more variable than the description. Consistency7 raise error if body of the expression has two or more equal variables names

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 excutor creates a *relation* calling the Web Service. It's implicity that executor checks the number of inputs and the arguments for the *function*. 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 the attributes (*elements*) in common between the current *relation* and the current *expression*. It 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 there is an implementation of Exponential Backoff Algorithm to control the flow of calls to a Web Service.

4. Result

We tested our Query Engine on a Unix Distribution with Java 8 installed passing the fallowing 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)
```

The result was 918 tuples in less than 2.5 seconds using less than 1 MB (Megabyte) of memory. Part of the results can be in the Figure 1.

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

Figure 1. Results of query to retrieve the all songs, albums of any artist that contains the names "Frank Sinatra".

5. Difficulties

To implement this Query Engine, we faced several challenges to perform calls, retrieve and process data from Web Services. The first main problem was to understand and correct the source code provided, we found some errors such as the wrong number of inputs and some content served in different formats rather than XML. These errors prevented us to perform calls to Web Services and apply some operation such as joins on the results. The second main problem was the file encodings, the MusicBrainz encode the served content using UTF-8 with support to miscellaneous symbols. However, characters such as *em dash ("–")*, or *long dash*, are not well supported by Microsoft Windows Distributions if the correct character set and encoding are not defined. So, 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 allow us to understand how to create a call composition and how to execute it. It was not painful to create the conjunctive queries but evaluate and execute them it was quite hard without a query optimizer. For this project, we only implemented a query executor and partial query validator that not allow Cartesian Product. The results shows that our Query Engine works pretty well and is operational. A possible extension of this project is the implementation of 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

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