Anderson Carlos Ferreira da Silva

Université Paris-Sud  
Télécom ParisTech

Query Engine for

Web Service Compositions

# 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.

# Materials and Methods

## 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]](#footnote-1), release[[2]](#footnote-2), recoding[[3]](#footnote-3) and label[[4]](#footnote-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](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>).

## 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.

# 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.

## 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 the symbol “?”, 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

### 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

## Query Evaluator

To evaluate the query we use the structured query and create a first

# Results

# Conclusion

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