Assignment 1 - Chapter 5

TU Delft Web Data Management Course 2014

Maria Voinea *4317602*

Stijn Pieper *4037952*

June 1, 2014

1 Introduction

This project is for the web Web Data Management (WDM) course. We chose to test the use of eXist as an XML database as described in Chapter 5 of the WDM book[1]. The code for the three actual applications, developed by Maria Voinea, can be found on *https://github.com/an3m0na/wdm-xdatabase*. They all require installation on an instance of eXist-db[**exist**]. The XML resources of the applications are included in folders named *collection* in the application roots.

2 Running XPath in eXide

The selectors under **Normal** are applied on the document extracted with $doc('movies/movies.xml')$, those under **Refs** are applied to $doc('movies/movies_refs.xml')$, and those under **Both** can be applied on both documents.

1. All title elements.

```
(: Both :)
//title
```

2. All movie titles (i.e., the textual value of title elements).

```
(: Both :)
//movie/title/text()
```

3. Titles of the movies published after 2000.

```
(: Both :)
//movie[year/text() >= 2000]/title/text()
```

4. Summary of “Spider-Man”.

```
(: Both :)
//movie[title/text() = 'Spider-Man']/summary/text()
```

5. Who is the director of Heat?

```
(: Normal :)
//movie[title/text()='Heat']/director/
    concat(first_name/text(), ' ', last_name/text())
(: Refs :)
//artist[@id = //movie[title/text()='Heat']/director/@id]/
    concat(first_name/text(), ' ', last_name/text())
```

6. Title of the movies featuring Kirsten Dunst.

```
(: Normal :)
//actor[first_name = 'Kirsten' and last_name='Dunst']/
    parent::movie/title/text()
(: Refs :)
//movie[actor/@id = //artist[first_name = 'Kirsten' and
    last_name='Dunst']/@id]/title/text()
```

7. Which movies have a summary?

```
(: Both :)
//movie[summary]/title/text()
```

8. Which movies do not have a summary?

```
(: Both :)
//movie[not(summary)]/title/text()
```

9. Titles of the movies published more than 5 years ago.

```
(: Both :)
movie[year-from-date(current-date()) - year/text() >= 5]/
    title/text()
```

10. What was the role of Clint Eastwood in Unforgiven?

```
(: Normal :)
//movie[title/text() = 'Unforgiven']/
    actor[first_name = 'Clint' and last_name = 'Eastwood']
    /role/text()
(: Refs :)
//movie[title/text() = 'Unforgiven']/actor[@id =
    //artist[first_name = 'Clint' and last_name = 'Eastwood']/@id]/
    @role/string()
```

11. What is the last movie of the document?

```
(: Both :)
//movie[last()]/title/text()
```

12. Title of the film that immediatly precedes Marie Antoinette in the document?

```
(: Both :)
//movie[title/text() = 'Marie Antoinette']/preceding::movie[1]/title/text
```

13. Get the movies whose title contains a "V" (hint: use the function contains()).

```
(: Both :)
//movie[contains(title/text(), 'V')]/title/text()
```

14. Get the movies whose cast consists of exactly three actors (hint: use the function count()).

```
(: Both :)
//movie[count(actor) = 3]/title/text()
```

3 Running XQuery in eXide

The xml collections are stored in the movies directory. All instructions are preceded by the variable definitions:

```
let $ms:=doc("movies/movies_alone.xml"),
    $as:=doc("movies/artists_alone.xml")
```

1. List the movies published after 2002, including their title and year.

```
for $x in $ms//movie
return <movie>{$x/title} {$x/year}</movie>
```

2. Create a flat list of all the title-role pairs, with each pair enclosed in a "result" element.

```
return <results>
{
for $x in $ms//movie/actor
return  <result>
            {$x/parent::movie/title}
            <role>{$x/@role/string()}</role>
        </result>
}
</results>
```

3. Give the title of movies where the director is also one of the actors.

```
for $x in $ms//movie
where $x/director/@id = $x/actor/@id
return $x/title
```

4. Show the movies, grouped by genre.

```
for $x in distinct-values($ms//movie/genre)
return <group genre="{$x}">
{
    for $y in $ms//movie
    where $y/genre = $x
    return $y
}
</group>
```

5. For each distinct actor's id in $movies_alone.xml$, show the titles of the movies where this actor plays a role. Variant: show only the actors which play a role in at least two movies.

```
for $x in distinct-values($ms//actor/@id)
(: For full list remove following condition :)
where count($ms//movie[actor/@id = $x]) > 1
return <actor>{$x},
{
    for $y in $ms//movie
    where $y/actor/@id = $x
    return $y/title
}
</actor>
```

6. Give the title of each movie, along with the name of its director. Note: this is a join!

```
for $x in $ms//movie
let $y := $as//artist[@id = $x/director/@id]
return <movie>
        {$x/title}
        <director>
            {$y/concat(first_name, ' ', last_name)}
        </director>
        </movie>
```

7. Give the title of each movie, and a nested element ¡actors¿ giving the list of actors with their role.

```
for $x in $ms//movie
return <title>{$x/title/text()}
    <actors>
    {
        for $y in $x//actor
        return <actor>
            {$as//artist[@id = $y/@id]/concat(first_name,
                ' ', last_name)} as {$y/@role/string()}
            </actor>
    }
    </actors>
    </title>
```

8. For each movie that has at least two actors, list the title and first two actor, and an empty "et-al" element if the movie has additional actors.

```
for $x in $ms//movie
let $a := $x//actor[1],  $b := $x//actor[2]
where count($x//actor)>=2
return <result>
        {$x/title}
        <actor>
        {$as//artist[@id = $a/@id]/concat(first_name,
            ' ', last_name)} as {$a/@role/string()}
        </actor>
        <actor>
        {$as//artist[@id = $b/@id]/concat(first_name,
            ' ', last_name)} as {$b/@role/string()}
        </actor>
        {if ($x//count(actor) > 2) then <et-al/> else()}
    </result>
```

9. List the titles and years of all movies directed by Clint Eastwood after 1990, in alphabetic order.

```
for $x in $ms//movie[director/@id =
        $as//artist[last_name = 'Eastwood'
        and first_name='Clint']/@id]
where $x/year >= 1990
order by $x/title
return <result>{$x/title} {$x/year}</result>
```

4 Movie Application - XMovie

This application is a very simple Web application that allows to search some information in an XML document, and displays this information in a user-friendly way. The final version of this application is given in the $wdm - xmovie$ directory. Screenshots can be found in the appendix.

4.1 Functionality

The user is presented with an input form where he can fill in the details for searching the movie database:

```
Title
Director
Actors
Year
Summary keywords
```

After submitting, the user is given a result list to the right of the form. Once a result is selected, the full details of the movie are shown.

4.2 Implementation

This application was developed in the eXist-db[**exist**] IDE (eXide), but does not rely on the eXist-db application architecture. It consists of an HTML file with CSS and JavaScript resources (Bootstrap framework[2]). AJAX calls are made to the eXist REST framework in order to get information.

An XQuery file contains the code to return both types of results (movie list and movie details) based on the HTTP request parameters. The XML result is then transformed using an XSLT file (through the $_xsl$ parameter mentioned in the REST documentation). The resulting data is in HTML format and can be easily appended to the HTML DOM for display. This extra conversion step was chosen as a means of both experimenting with the markup language, as well as separating the server-side logic from the visualization of the results.

5 Shakespeare Application - XOpera

The application's purpose is to allow the user to browse through a play in order to analyze its content, read some specific parts and find related information. The final version of this application is given in the $wdm - xopera$ directory. Screenshots can be found in the appendix.

5.1 Functionality

The analysis of the requirements led to the design of the application structure illustrated in $Figure1$. The rounded rectangles represent the different application views (or "pages"

in a traditional sense).

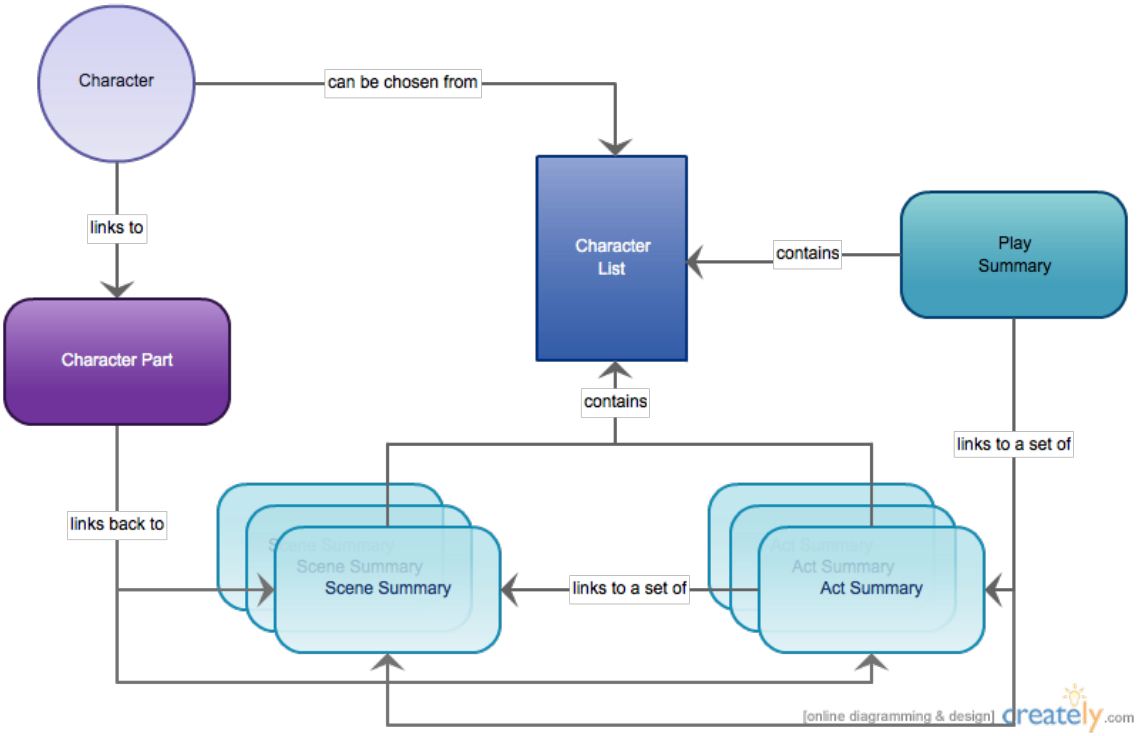

Figure 1: XOpera - Application structure

The play summary view contains the author, the list of characters and their description, and the outline of the play (structure in acts and scenes). The act summary view contains the list of speakers in the act and the list of contained scenes. The scene summary view contains the description of the scene, the list of speakers in that scene, and the scene content (lines and stage directions of all the characters). The character part view shows all the lines of a character, categorized in acts and scenes.

The navigation bar contains a dropdown list for choosing the play. Thus, any play summary is accessible from everywhere. The characters and speakers represent links to the specific character part. On each character part view, the names of acts and scenes are links to the associated act and scene views. Each outline also contains links to the actual acts and scenes.

For ease of use, especially on a mobile platform, all bulky text is contained inside collapsible panels which can be expanded on demand.

5.2 Implementation

Since this particular application contains a considerable amount of interconnected links, the XQuery application structure was used in development. The main HTML file con-

tains only the data templates. The application logic contained in *modules/app.xql* is responsible for returning custom HTML to fill those templates.

The main template function is shown in *Figure*2. It reads the HTTP request paramaters and shows the appropriate view according to them. The content generating functions are responsible for outputing the actual HTML for the corresponding views (*Figure*3).

```
declare %templates:wrap
 function app:main($node as node(), $model as map(*)) {
   let  $play := xs:string(request:get-parameter("play", "")),
        $act := xs:string(request:get-parameter("act", "")),
        $scene := xs:string(request:get-parameter("scene", "")),
        $char := xs:string(request:get-parameter("char", ""))
   return
        if($play) then
            let $play_node := $app:plays[descendant::PLAYSUBT/text() = $play]
            return if ($char) then
                        local:display-part($play_node, $char, $act, $scene)
                   else if ($act) then
                            if ($scene) then
                                local:display-scene($play_node, $act, $scene)
                            else (local:display-act($play_node, $act))
                        else local:display-summary($play_node)
        else local:display-home()
};
```

Figure 2: XOpera - Main template function

6 Music Application - XPartiture

The basic requirement is to be able to store XML music sheets in eXist, and to display the music on demand. Optional functionalities include extracting useful information from the XML documents, inputing melodies from a microphone and rendering music based on the score. The final version of this application is given in the $wdm-xpartiture$ directory. Screenshots can be found in the appendix.

6.1 Functionality

The application first displays a list of the MusicXML files in the database. On choosing a file, the user is shown a loading screen while it is processed using the Lilypond[3] software. The user is presented with the details of the score, as extracted from the XML file (including the lyrics). Three tabs can be activated. The first displays the PDF showing the music sheet. The second shows the score in Lilypond code. The last offers a primitive player for listening to the score.

```
declare function local:display-home() {
    <div class="jumbotron">
            <div class="container">
                <h1>Welcome to XOpera!</h1>
                <p>Just choose a play from the above navigator menu to start!</p>
            </div>
    </div>
};

declare function local:display-act($play, $desired_act) {
  let $act := $play//ACT[position() = xs:int($desired_act)]
  return
    <div class="container">
        <h4>{$act/TITLE/text()}</h4>
        <div class="accordion panel-group" id="accordion_prologue">
            <div class="accordion-group panel panel-default">
                <div class="accordion-heading panel-heading">
                  <h4 class="panel-title">
                      <a class="accordion-toggle collapsed"
                        data-toggle="collapse"
                        data-parent="#accordion_prologue"
                        href="#collapse_prologue">
                        Prologue
                      </a>
                  </h4>
                </div>
                <div id="collapse_prologue"
                    class="accordion-body collapse panel-body">
                    <div class="accordion-inner">
                    {local:display-speech($act/PROLOGUE/SPEECH)}
                    </div>
                </div>
            </div>
        </div>
        <div class="accordion panel-group" id="accordion_personas">
            <div class="accordion-group panel panel-default">
                <div class="accordion-heading panel-heading">
```

Figure 3: XOpera - Content generating functions

6.2 Implementation

Since XQuery and JavaScript did not permit running a system command to use the Lilypond scripts, the application architecture consists of two separate projects: Java server and HTML client.

The Java server uses Spark[4] to run basic REST services locally. It makes RPC calls to the XML database through the Java packages provided by eXist-db. *Figure*4 shows the structure of this project.

The HTTP client makes AJAX calls to the Java server in order to get all the information. Since the Same-origin Policy forbids calls between the two servers, the response had to be wrapped with JSONP. Bootstrap and *spin.js*[5] are used for the interface. The final rendering of the MIDI files (received in base64 format) is done by *MIDI.js*[6]

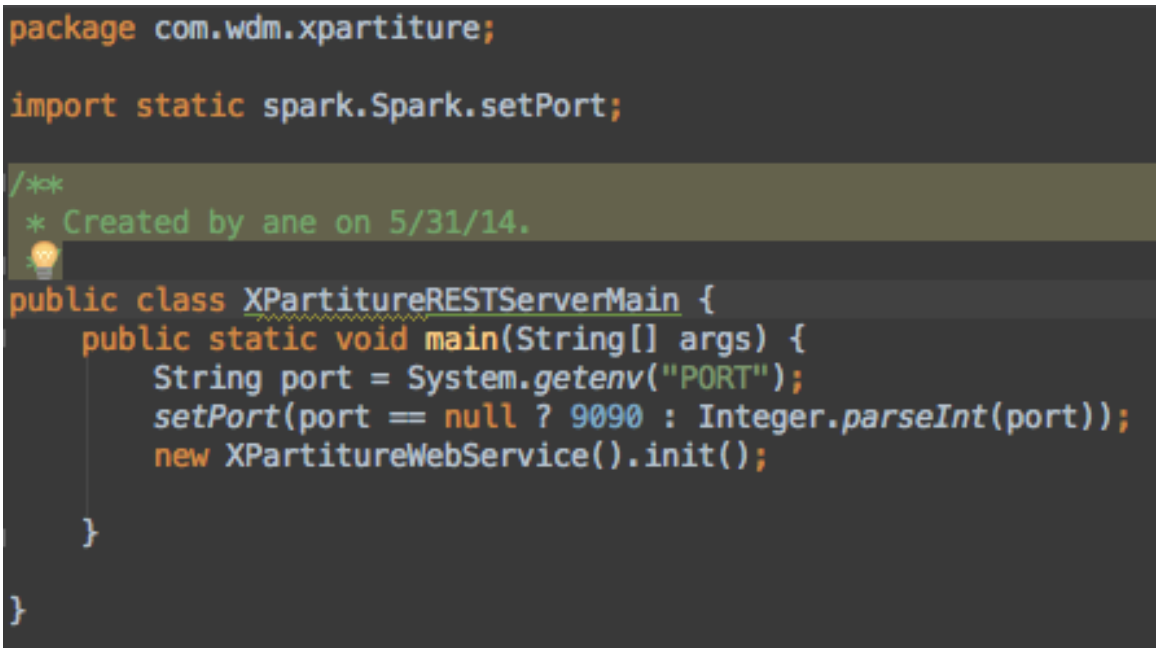

Figure 4: XPartiture - Java application structure

and *jasmid* [7].

References

[1] `http://webdam.inria.fr/Jorge/`. [Online; accessed May 2014].

[2] `http://getbootstrap.com`. [Online; accessed May 2014].

[3] `http://lilypond.org`. [Online; accessed May 2014].

[4] `http://www.sparkjava.com`. [Online; accessed May 2014].

[5] `http://fgnass.github.io/spin.js/`. [Online; accessed May 2014].

[6] `http://mudcu.be/midi-js/`. [Online; accessed May 2014].

[7] `https://github.com/gasman/jasmid`. [Online; accessed May 2014].

7 Appendix

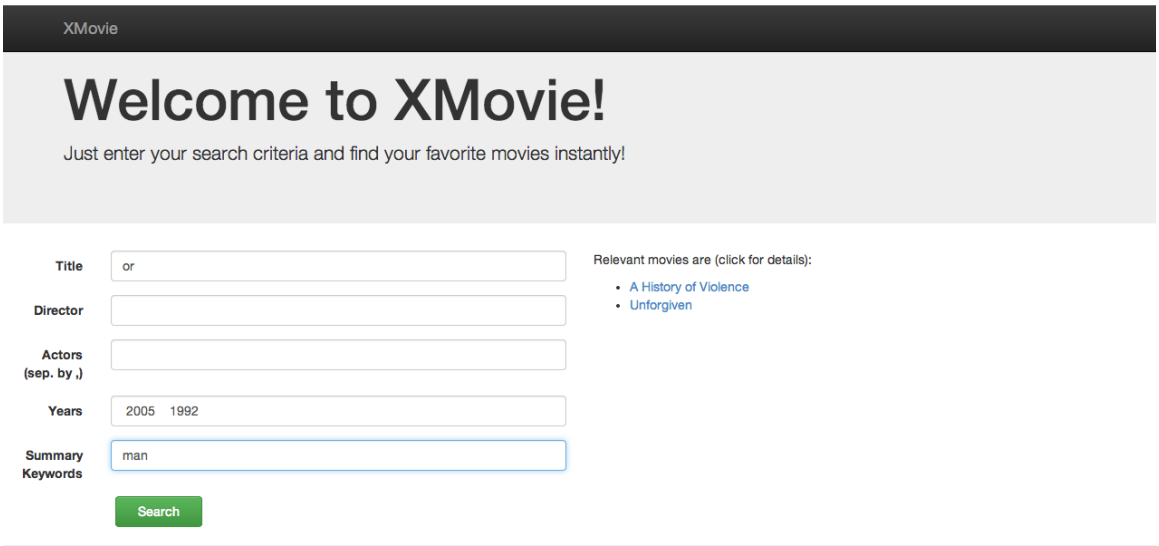

Figure 5: XMovie - Search results

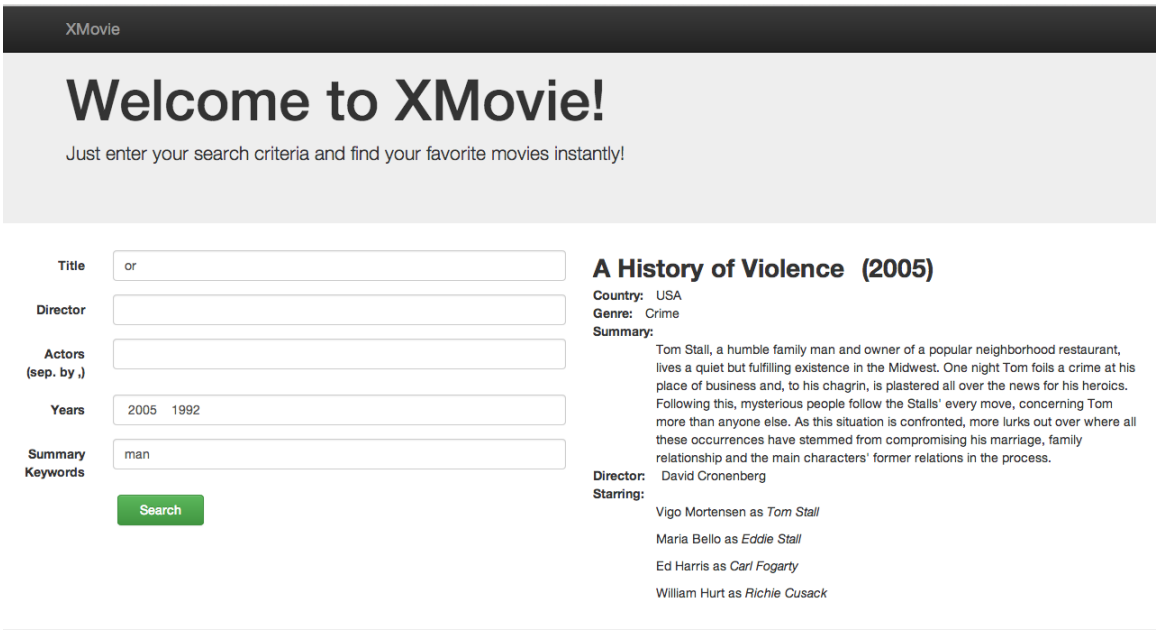

Figure 6: XMovie - Movie details

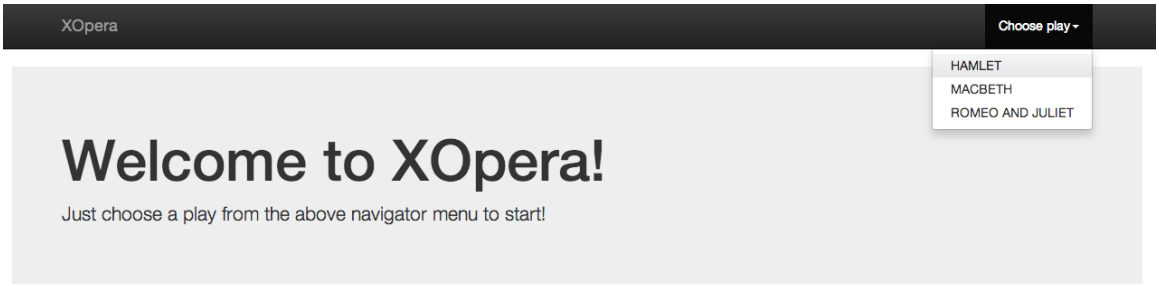

Figure 7: XOpera - Choosing a play

Figure 8: XOpera - Play summary

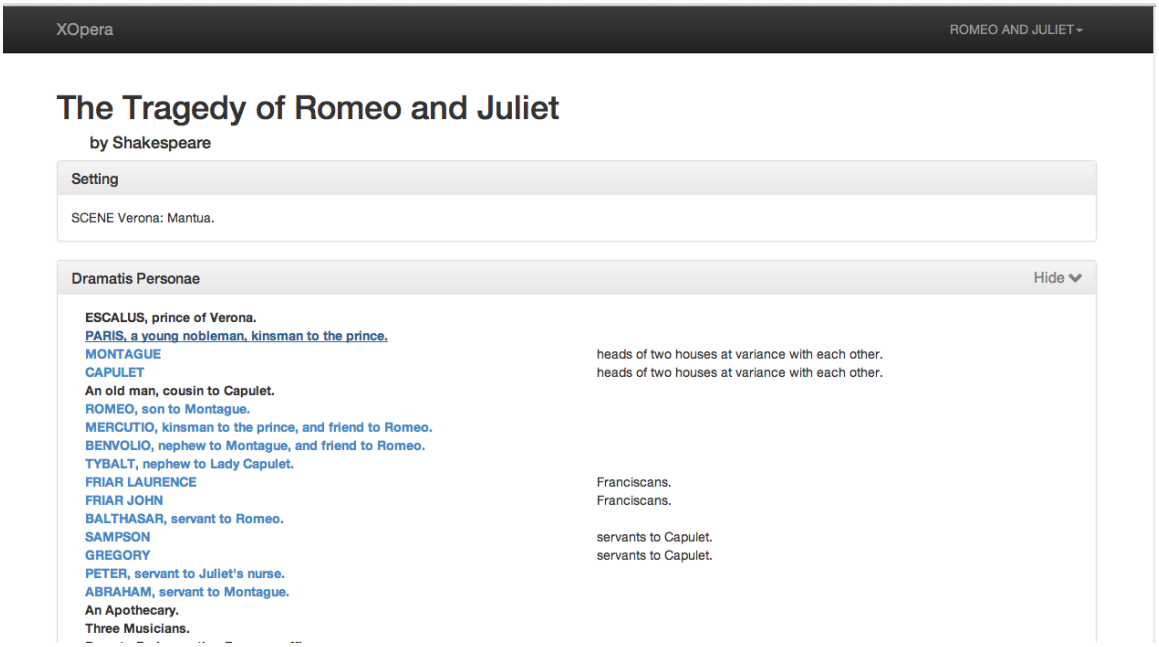

Figure 9: XOpera - Play summary characters

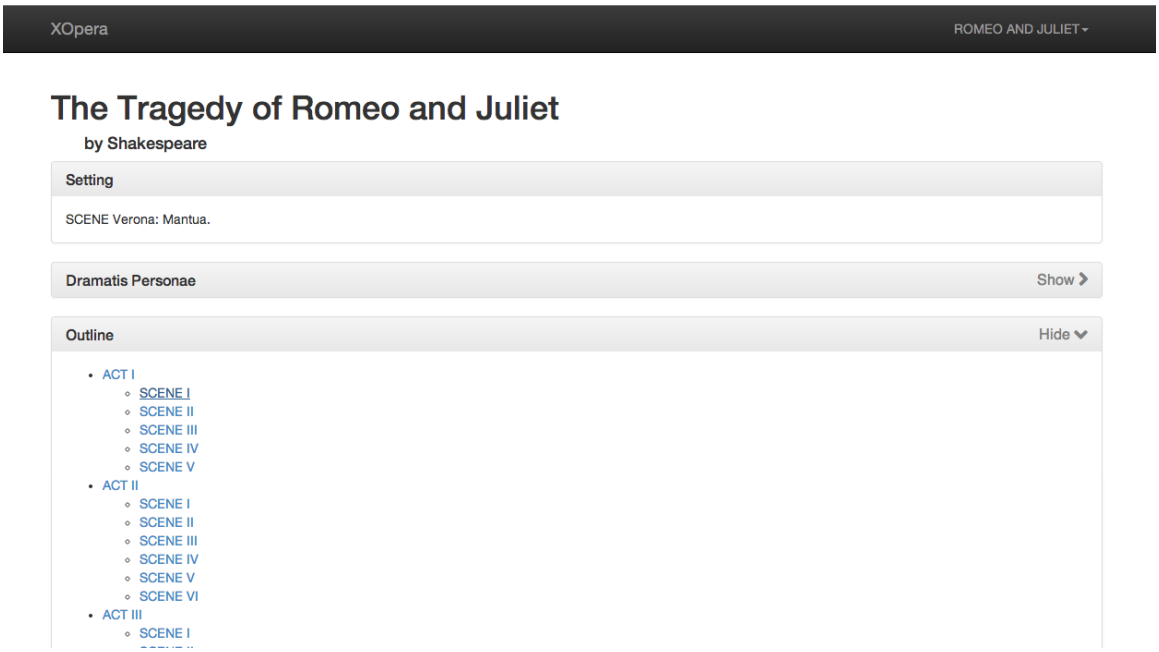

Figure 10: XOpera - Play summary outline

Figure 11: XOpera - Act summary

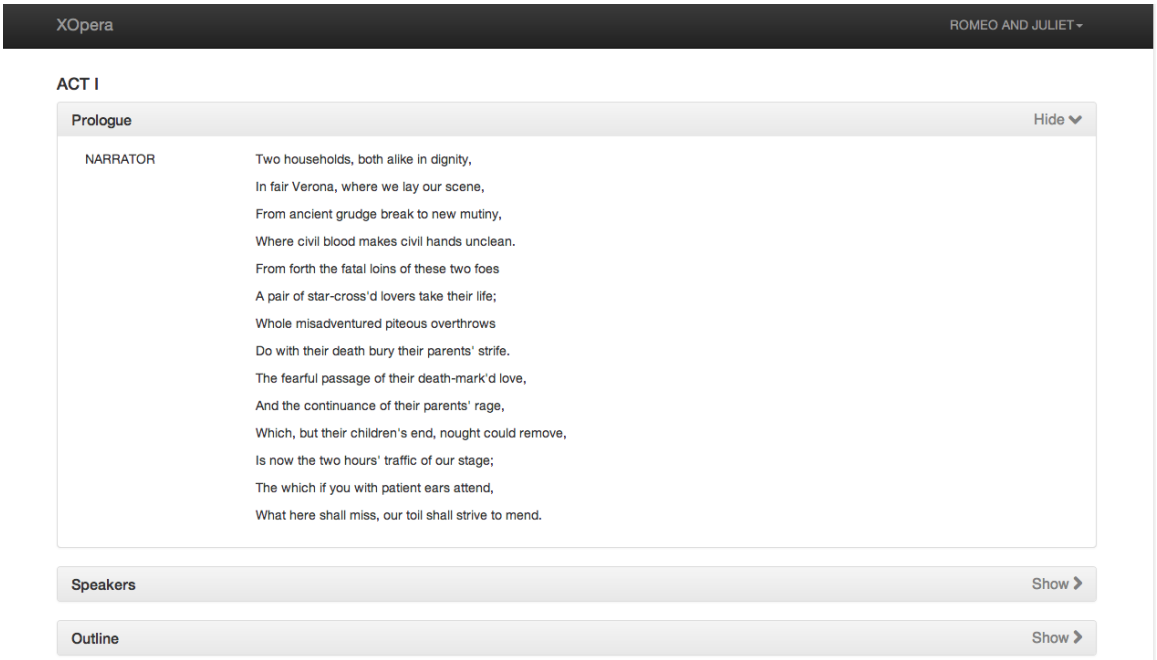

Figure 12: XOpera - Act summary prologue

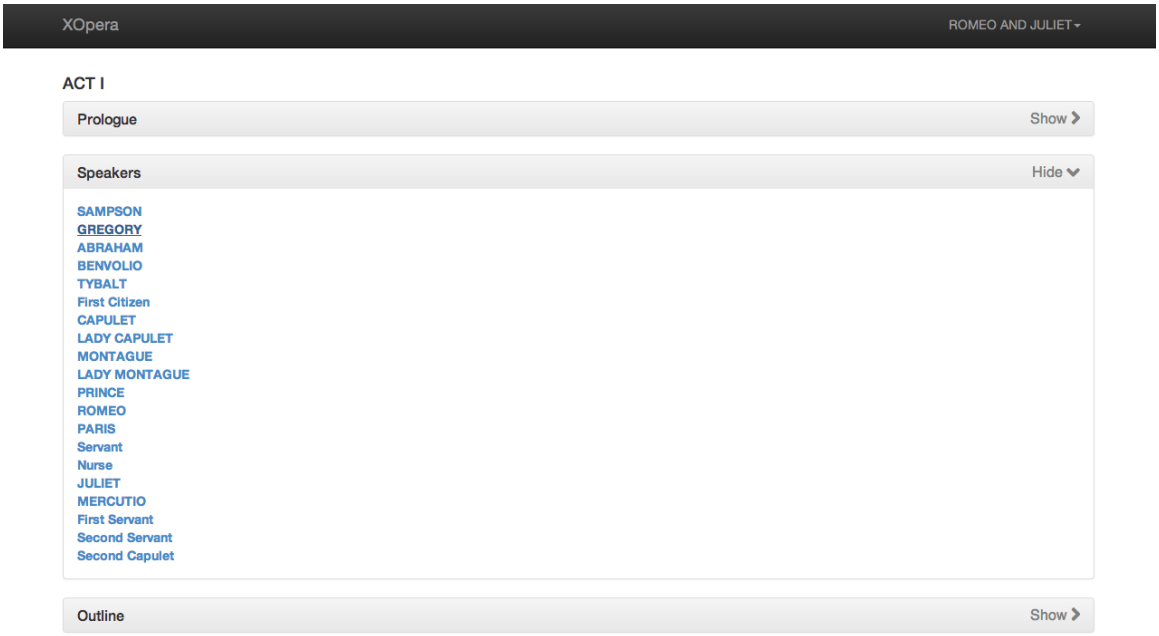

Figure 13: XOpera - Act summary speakers

Figure 14: XOpera - Act summary outline

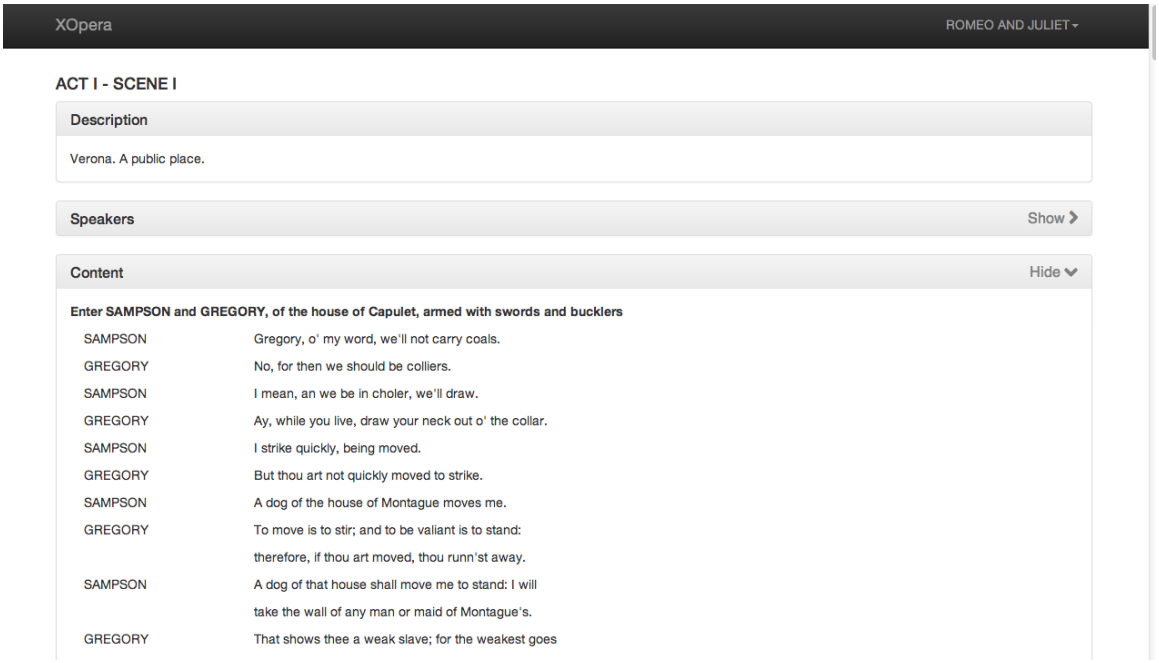

Figure 15: XOpera - Scene summary

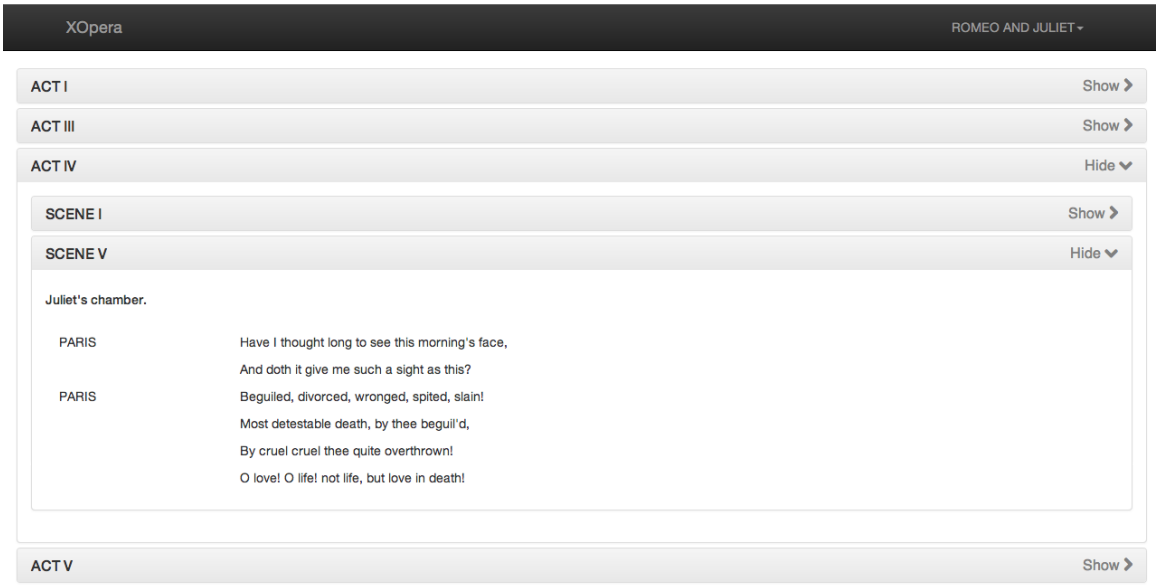

Figure 16: XOpera - Character part

Figure 17: XOpera - MusicXML file archive

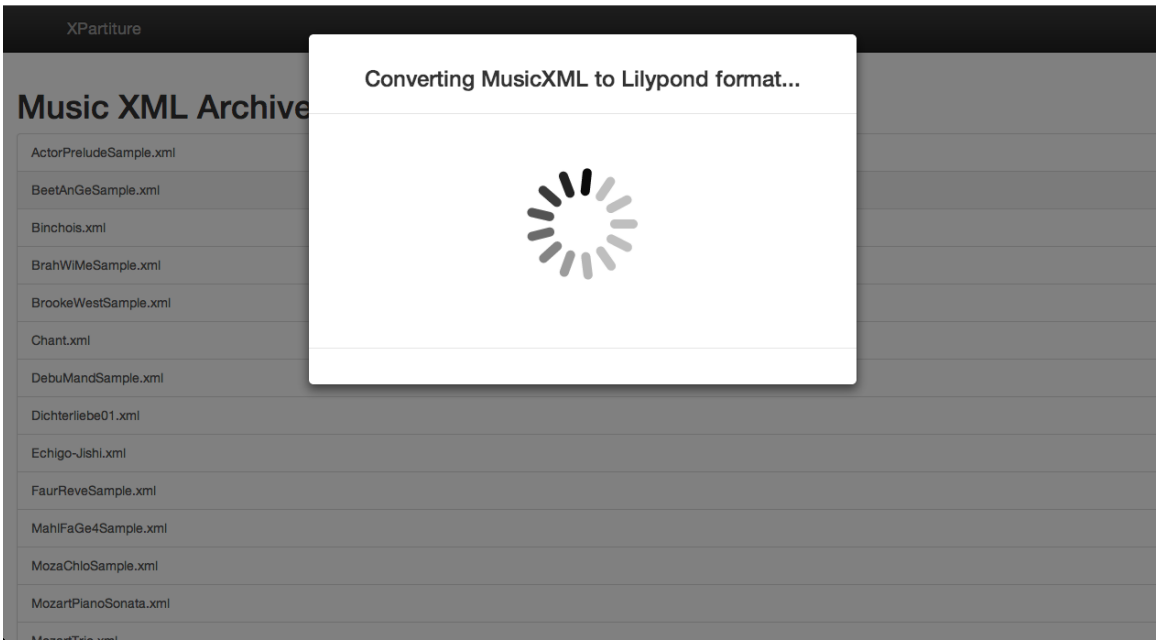

Figure 18: XOpera - Awaiting XML parsing and conversion

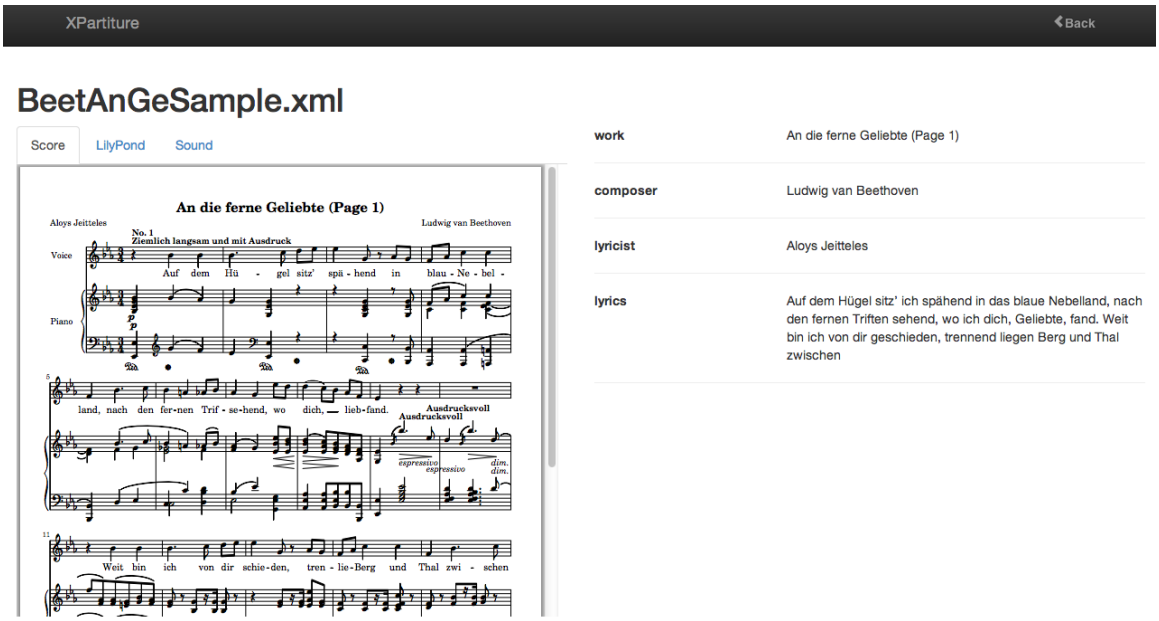

Figure 19: XOpera - Details and music score as PDF

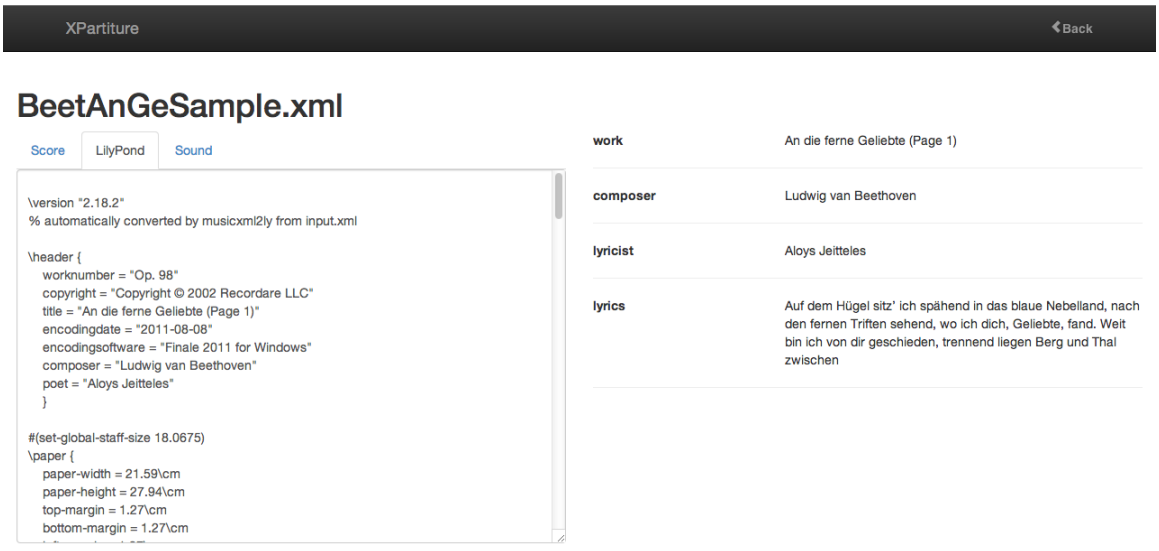

Figure 20: XOpera - Lilypond score

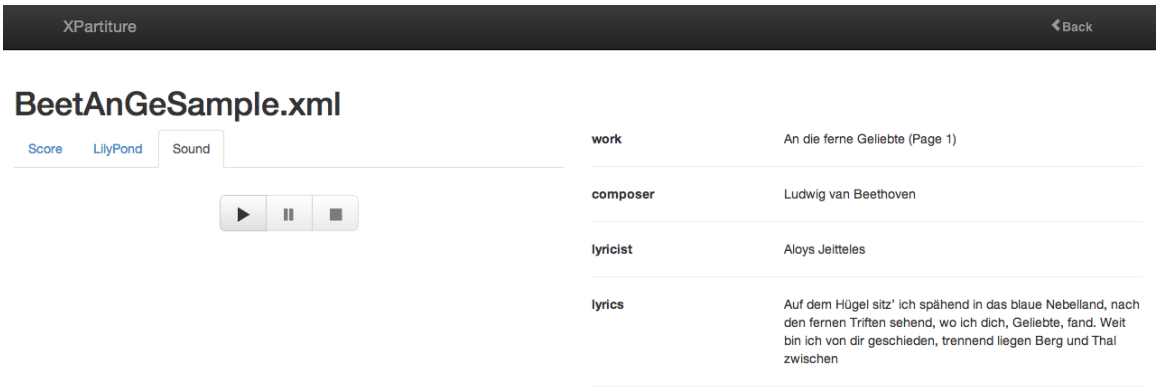

Figure 21: XOpera - Midi file player