Encoding and Navigating Fauvel:

XML and Search Capabilities for *Digital Fauvel*

Alison Y. Chang

Mentor: Rebecca Fiebrink, Princeton University

Princeton University Soundlab – *Digital Fauvel*

Mentors: Rebecca Fiebrink & Anna Zayaruznaya

Undergraduates: Alison Y. Chang, Jamie Chong, Brendan Chou

Humanities Scholars: Jamie Greenberg, Jenna Phillips, Michael Scott Cuthbert

1. Introduction to Fauvel

The *Roman de Fauvel* is a 14th century satirical medieval poem about a horse named Fauvel who corrupts France. The mix of poetry, images, and music in this important early multimedia object is both beautiful and intriguing, and the questions raised in its analysis are highly pertinent to both modern computing and humanities. However, to study this 18" x 26" manuscript, it is currently necessary to consult at least half a dozen sources at once: a translation from old French to modern French, another from French to English, modern notation for polyphonic music objects, another for monophonic music, explanation of the images, and more. Understanding *Fauvel* consequently includes unnecessary juggling of all of these resources, hindering the process of studying the work or reading it as a story, as *Fauvel*'s creators had intended.

Our objective is therefore to create a digital version of the *Roman de Fauvel* on the Samsung SUR40 (previously the Microsoft Surface 2.0), a large, 40" screen size, multi-touch tabletop. Our *Digital Fauvel* brings together these existing resources without diminishing the experience of viewing the original manuscript in its true size, color, and style. Over the course of this summer, we aimed to build an interface that allows users to view pages of *Fauvel* with added ability to zoom, pan, or display different translation overlays, with a functioning sidebar that would allow for text search and the beginnings of studying music objects. Working with both computer scientists and musicologists, building off of prior research on XML encoding and UI design, and balancing the efficiency needed for computation with the functions emphasized by medieval scholars, we strove to create a *Digital Fauvel* that would not only stay true to but enhance the design and effects of the original *Roman de Fauvel*.

2. Prior Research

As of the beginning of summer 2013, the *Digital Fauvel* team had acquired the Microsoft Surface 2.0 but had not begun coding. However, several Princeton University undergraduate computer science students had conducted preliminary research for the creation of *Digital Fauvel*.

2.1 Data Representation for the Digital Fauvel – Alan Thorne, '13

Alan Thorne consulted existing research on digital text representation (namely the Text Encoding Initiative, or TEI[1]) to propose an XML based encoding scheme for *Fauvel* [1]. He divided the encoding into two parts, content and layout, with the former indicating the content of each page (i.e., poetry/music/image objects) and the latter specifying the location of each object on that page. Thorne's work served as the basis for our creation of four documents: three content XMLs (original text, modern French translation, and English translation) and one layout. As I designed a standard for the content XMLs, I often worked directly with Thorne to incorporate newly discovered elements into the existing structure in a way that makes sense both to musicology scholars and computer scientists (see **3.1** for details).

[1] For more info, read http://www.wwp.brown.edu/outreach/seminars/tei.html.

2.2 Interaction Design for Medieval Multimedia – Andrew Callahan, '13

Andrew Callahan explored interface design for *Digital Fauvel* [2], referring to the fundamentals of human-computer interaction (HCI) techniques and existing programs written for the Microsoft Surface. Intermittently getting feedback from musicology graduate students, he created and refined paper prototypes, observing the natural actions they would use to perform common tasks like navigation, search, and translation, taking note of any confusing or missing functionality. His design of a main window that displays openings (two-page spreads) of the *Fauvel* manuscript with a sidebar on the right side that houses operations like search was the springboard for the UI design that we implemented this summer; my work in particular was on the sidebar, namely the search functionality (see **3.3**).

3. Search Capabilities

My development of search capabilities for *Digital Fauvel* can be broken down into three sections: the encoding of the three Content XML files (**3.1**), the writing of back-end code for search functionality (**3.2**), and the design and implementation of a front-end UI for the sidebar (**3.3**), specifically for the "Search" application tab.

3.1 Encoding of XML Content Files

The original text of *Fauvel* (poetry, music lyrics, and image captions) and modern French translations of most original text were available in an OCR-enabled PDF of the Armand Strubel edition [3], while the English XML was created from Eliza Zingesser's translation [4]. The process of creating content files involved writing a Java program that took in the text, which was copied and pasted from Strubel or Zingesser, and reformatted it into XML form, identifying each object correctly, counting line numbers, etc. Although the basic structure of encoding had been laid out by Alan Thorne (see **2.1**), the process included many challenges that arose at the intersection of *Fauvel* as a digital project versus *Fauvel*, the subject of musicology, particularly in terms of organizing and categorizing the objects of *Fauvel* in a way that makes sense to users from both fields.

While the English translation included poetry only and was therefore quite simple, Strubel's edition included text about almost every object in *Fauvel*. The guiding principle behind the design of the encoder program for the latter, TextReader.java, was to make the human's job as simple as possible, to allow for people to copy and paste from the PDF without a high understanding of how the program works to interpret the text. In some cases, the computer would be able to interpret objects based on the content of the text itself; a short line starting with "Fo" (folio) indicated the start of a new page, and a line beginning with the name of a music genre (of which there are seventeen types) indicated the start of a music object. However, for most objects, the computer needed certain cues to identify each line of text as belonging to poetry, music lyrics, or image caption. Consequently, the human would also need to add in simple cue-lines at the start and end of each object (i.e. "STARTPOEM", "ENDMUSIC", or "ENDFOLIO").[2]

[2] See the Instructions.txt document in our repository: https://code.google.com/p/digitalfauvel.

The following sections describe tricky or interesting challenges that have implications reaching beyond *Fauvel*, shedding light on the questions that arise when representing a humanities object in a digital form.

3.1.1 Columns and objects

While XML encoding requires a strict object hierarchy, regulated by the nesting of nodes, a manuscript and its objects have much looser conceptual and layout hierarchies. In many instances, elements that we, as humans, can easily describe and understand on a single page do not easily fit into the structure typically expected by a computer reading from XML. As a result, the encoding process involves making decisions about how to best represent *Fauvel* in a way that reflects our natural perception of a manuscript, while organizing the elements into a hierarchy understandable by technology as well.

For example, it is natural to think of a page as a container that aggregates columns; in *Fauvel*, each page has three: *a*, *b*, and *c*. It is tempting to then think of a column as aggregating objects—poetry, music, or images—but this organization is quickly rendered unsuitable by the numerous objects in *Fauvel* that span multiple columns or even multiple pages. One potential solution is to regard a "column" as an attribute, with an object appearing in columns a and b containing the attribute `column="ab"` or `"column ="a,b"`, both of which create complications when searching for all objects that, for example, appear in column "a"; the values of "ab" and "a,b" would not match the search query. Alternatively, a column could be a tag:

```
<object>
   <column>a</column>
   <column>b</column>
</object>
```

This approach, while less concise, would perhaps be the most appropriate if we wished to include "column" information in the Content XML, as preferred by musicologists like Anna Zayaruznaya. However, after further discussion with Thorne and Fiebrink, I have concluded for the time being that it would be best to ignore columns in the Content XMLs and indicate them in the Layout XML instead.

3.1.2 Drop caps encoding

As mentioned earlier, we split the encoding of *Fauvel* into two types of XML: content and layout. For the most part, this division is easy enough to grasp. However, in the case of drop caps, there is no simple boundary between content and presentation. Making a decision about where and how to encode drop caps therefore involves exploration of what information we (as scholars) expect from an XML encoding, and how we intend to *use* that information.

Drop caps are an intriguing element of medieval manuscripts, a larger, decorated initial character marking the start of a new idea in poetry or a new voice or stanza in music. Two reasons for including drop caps in an XML encoding of *Fauvel* would be allowing potential study of drop caps only and incorporation of drop caps into our translation overlays. In terms of translations, we decided that it did not make sense to include drop caps, especially since the first letter of a line of text is often different in each translation.

We chose to encode drop caps for the original text, but many questions remained as to *how*. The dual nature of a drop cap letter as both a visual element and a textual one complicates its inclusion in the drop cap (<dc>) and poetry text (<l>, for line) tags; would the text be split between the two tags, or could an embellished letter belong to both? Furthermore, what kind of text has the drop cap property: a single letter, the entire word, or the entire line? The varying sizes of drop caps that appear in *Fauvel*—most are four lines of poetry tall (see Figure 1), but some are two (Figure 2)—add further complication. Are these different types of drop caps, or is "height" an attribute worth indicating for every encoded drop cap? Many answers to these questions depend on underlying classifications of drop caps as art versus text, categories that are easy for us to comprehend and negotiate in a humanities sense but harder to reconcile when we are forced to draw clear cut lines for technology.

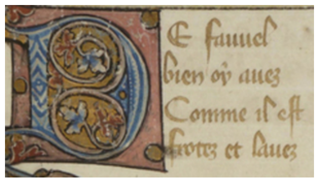


Figure 1: Drop cap (four lines tall)

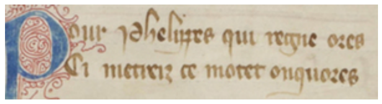


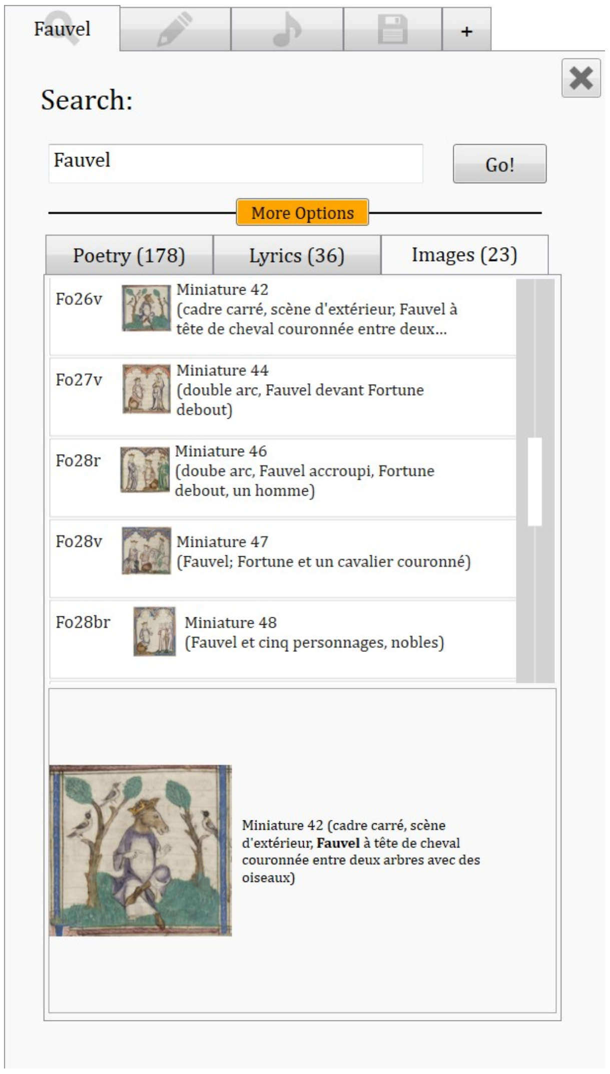
Figure 2: Drop cap (two lines tall)

3.1.3 Counterparts

As mentioned briefly in *3.1.1*, many of the music objects in *Fauvel* stretched across multiple pages. While this is a simple enough concept for human minds, it proves to be trickier in a digital sense. Since each object's ID tag includes its page number, a music object that spans more than one page could either have one tag and exist as a single part or be broken into several parts, each with a unique tag. In the latter case, we would then need a way to identify that several tags are actually referring to parts of the same object. Albeit more complicated, the second option would more accurately represent the structure of the music object in *Fauvel*, so I created a special Counterpart object that had a primary String "name" (the first tag) and a list of other names (the other tags) along with methods to match alternative tags with the primary one. The lyrics within the music object would also need to be split up into parts; although Strubel did not indicate these page breaks in his edition, I referred to actual images of *Fauvel*'s folios to divide up the original text and used my best judgment for the other translations. Since the structure of the manuscript itself was to include whatever fit on that page and then switch to a new one, there were many lines or even words that split across page breaks. This break introduced a new issue of whether the word should be included within the tag of the first page, the second, or both, keeping in mind the risks of losing searchability.

3.2 Search functionality: Back-end code

The biggest challenge in developing back-end code (written in Visual Studio using C#) for text search was striking a balance between efficiency and meeting user expectations created by popular search engines like Google. Although the user was searching for text that could appear in poetry, music lyrics, or image captions, we decided that the search result should also include other context clues like line numbers, page numbers, and translations. Results appear as a list, and users can select promising results to view in a "closeup" section (Figure 3). Selecting the result in the "closeup" prompts the opening of a new tab in the main UI, flipped to the page of that result (Figure 4).



Fo26v Miniature 42 (cadre carré, scène d'extérieur, Fauvel à tête de cheval couronnée entre deux...

Fo27v Miniature 44 (double arc, Fauvel devant Fortune debout)

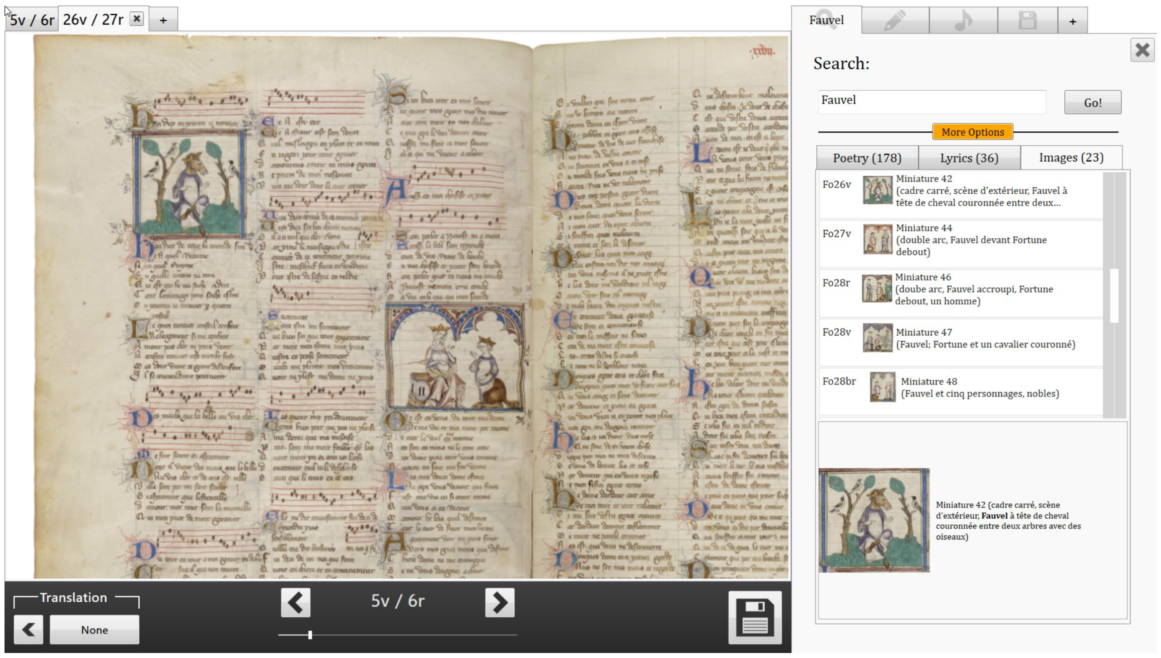
Fo28r Miniature 46 (doube arc, Fauvel accroupi, Fortune debout, un homme)

Fo28v Miniature 47 (Fauvel; Fortune et un cavalier couronné)

Fo28br Miniature 48 (Fauvel et cinq personnages, nobles)

Miniature 42 (cadre carré, scène d'extérieur, **Fauvel** à tête de cheval couronnée entre deux arbres avec des oiseaux)

Figure 3: The search tab



Search:

Fauvel

Go!

Fo26v	Miniature 42 (cadre carré, scène d'extérieur, Fauvel à tête de cheval couronnée entre deux…
Fo27v	Miniature 44 (double arc, Fauvel devant Fortune debout)
Fo28r	Miniature 46 (doube arc, Fauvel accroupi, Fortune debout, un homme)
Fo28v	Miniature 47 (Fauvel; Fortune et un cavalier couronné)
Fo28br	Miniature 48 (Fauvel et cinq personnages, nobles)

Miniature 42 (cadre carré, scène d'extérieur, **Fauvel** à tête de cheval couronnée entre deux arbres avec des oiseaux)

Figure 4: Opening of new tab from search result closeup

3.2.1 Thumbnails

As seen in Figures 3 and 4, a thumbnail is associated with each search result. The details of making thumbnails involved two major considerations: which part of an object the thumbnail should depict, and the efficiency of programmatic thumbnail creation. Our decisions required us to predict the information a user would want and need to see in order to determine whether a particular search result was worth further exploration.

In terms of the actual thumbnail content, music objects were the most complex because of their tendency to span columns or pages. In the layout XML, this meant that a single music object could have several sets of coordinates. The question was *which* set of coordinates we would use for a result, especially if the search query only appeared later in the object. Our final decision was to use the first thumbnail for each object, opting to show the user the more recognizable beginning of the object instead of an ambiguous image from the later part of the object.

Since our program would now require only one thumbnail per object, our second issue was simplified: we could create thumbnails ahead of time using a set of command line image editing tools called Imagemagick.[3] Due to the extremely high quality of our scans (each 5250 by 7350 pixels), each thumbnail originally took two seconds to create. Now, we could return more than ten results every second, a vast improvement.

3.2.2 Multi-word search

Although searching for a continuous multi-word phrase is simple, it is much trickier to search for multiple words that appear in the same vicinity in any order. How close together do words need to appear to qualify as part of a single search result? The

[3] http://www.imagemagick.org/script/index.php

answer depends on whether someone searches a phrase like "Fauvel Fortuna" to find a small section of text in which both words appear, or if the user is really searching for sections of the poem in which both Fauvel and Fortuna are key characters. If the user desires the latter, in which search words are more like keywords or tags than actual text search queries, the search breadth would need to be much wider (i.e., allowing for words to appear 10 lines apart instead of 3–5), or perhaps passages of text would need to have the added attribute of "keywords" and "tags".

Another question is the percentage of search words that must appear in a given section of text to qualify as a result. For example, if a user searches for five words, does he or she only expect results with all five? At the other extreme, would inclusion of results matching only one word out of five clutter the screen and add unnecessary delay? To keep processes fast while guiding the user towards similar (and more fruitful) queries, I also considered populating the search result list with full matches only but including a list of similar searches, each indicating the number of results that search would find. Professor Fiebrink and I discussed these potential approaches, and we concluded that the final decision would require extensive user testing, observing what people using *Digital Fauvel* expect and prefer from the search application. For the time being, I limited our returns to results that include *all* of the search words within a five-line span, allowing for future adjustment.

3.3 UI design prototyping and implementation: Sidebar and Search

Following Andrew Callahan's style and ideas for the main *Digital Fauvel* UI (see **2.2**), I created a series of prototypes for the sidebar and the search application with pencil and paper and later in WPF[4]. After working to refine the designs by getting feedback from both humanities scholars and computer scientists, I incorporated the code into our main UI, adjusting as necessary to account for the touch events and other demands of the Microsoft Surface.

The first step was to decide how to fit search, annotation, music studying, and other capabilities into the same sidebar. Although the Surface has a very large screen, we did not want to clutter it and risk overwhelming users with too many buttons and sections. However, we wanted the different functions to be easily discoverable and usable. After creating a UI for the entire sidebar, my next responsibility was to design a UI for the search application and then combine it with the back-end search capabilities (see **3.2**).

[4] Windows Presentation Foundation. See http://msdn.microsoft.com/en-us/library/aa970268.aspx.

3.3.1 "New tab" sidebar design

I settled on a sidebar design inspired by "Google Chrome" (see Figure 4), with the default screen set to a "new tab" canvas on which each application is neatly displayed as an icon. Tapping on an icon opens a new tab of that kind, and users could easily adjust the number of tabs open for each application. This solution proved to not only use screen space efficiently but also indicate that the addition of new functionalities for *Fauvel* would be easy to develop and incorporate, an important message to put forth, as many other coders will continue our work on the *Digital Fauvel* project, and we hope to set a well-designed but widely applicable precedent for other digital humanities projects in the future as well.

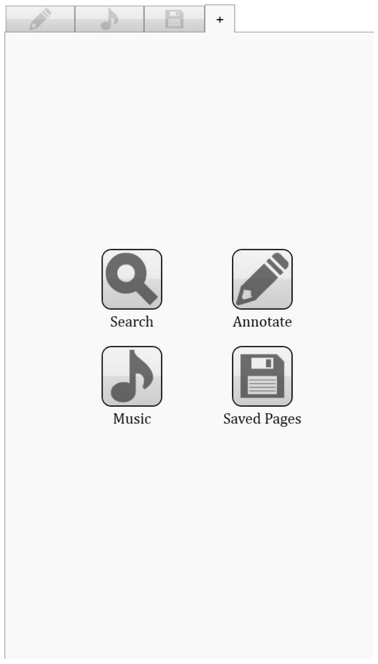


Figure 4: "New tab" in Sidebar

3.3.2 Advanced search

For the search application, one major debate was the display of advanced or extended search options. I wanted to keep the default design (Figure 5) as simple as possible but minimize confusion and optimize discoverability. After trying out various word-less buttons such as a black, triangular down arrow (intended to open up an extended options panel), user feedback led me to change the design to a short, wide, rectangular "More Options" button that would open up additional user preferences for search settings (Figure 6).

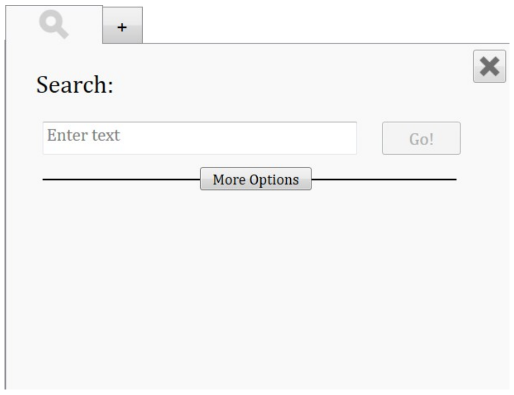


Figure 5: Simple search

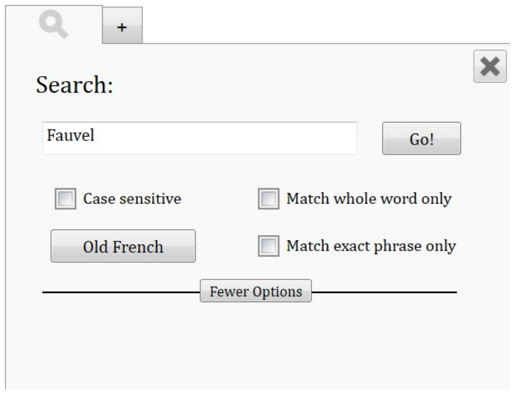


Figure 6: Advanced search

3.3.3 Search suggestions

Many measures were taken to avoid user confusion, especially the erroneous conclusion that a certain query has no matches when the return of zero results was actually due to a user's "mistake". For example, if any default settings have been changed, after minimizing the "extended options" panel, the "More Options" button glows a different color (see Figure 3) to attract attention and prompt users to check their current settings. If the search result tab control is open to a tab with no results but another tab *does* have results, it automatically switches to the latter, to avoid a user reading "0" and assuming that refers to all three genres. Basic hints and error messages have been added to "0" result tabs, such as information about music lyrics only being available in original or modern French (but not English). Eventually, I hope that the search application will be further developed so that low or no return search queries will prompt suggestions (see *3.2.2* for similar discussion), perhaps with each suggestion automatically launching a new search upon user selection.

4. Conclusion

Building upon Callahan and Thorne's prior research for *Digital Fauvel*, our team's role this summer was to lay the groundwork not only for future *Fauvel* scholars and contributors but also for other groups of musicologists and coders who wish to explore similar cross-disciplinary projects involving different manuscripts or other significant works in the humanities. All of our code (the Java programs used to encode XML files, our main C# code for the Surface, and many documents detailing our thought processes and methodology throughout) is available in a Google code repository[5], in hopes of inspiring others to contribute to our project, adapt it for other technology, or apply it to a different manuscript. As we designed the UI and structured the code, we therefore strove to keep things flexible, recording the challenges we faced in encoding a multimedia object in XML and allowing easy addition of another language's translation of *Fauvel* or a new sidebar application.

Digital Fauvel lives at the intersection of two disciplines: computer science and musicology. As a devoted musician newly discovering a passion for computer science, I am particularly intrigued by the exciting questions stirred up by *Digital Fauvel*'s creation as the demands and expectations of the two fields conflict and interact. The application of computer science to musicology complicates and challenges our concepts of technology and how it can be used to help us in both ordinary and high-level endeavors. Just as the *Roman de Fauvel* still serves as an important example of early multimedia work, with writing, art, and music coming together to create an incredible story, perhaps the *Digital Fauvel* will serve as a notable example of cross-disciplinary studies for centuries to come.

[5] https://code.google.com/p/digitalfauvel/

References

1. Thorne, Alan. *Data Representation for the Digital Fauvel*. Independent work report, Princeton University, Spring 2013.

2. Callahan, Andrew. *Interaction Design for Medieval Multimedia.* Independent work report, Princeton University, Spring 2013.

3. Strubel, Armand. Editor. *Le Roman De Fauvel.* Paris: Librarie Général Française, 2012.

4. Zingesser, Eliza. Translator. *Le Roman De Fauvel.* Paris: Librarie Général Française, 2012.