CS310: Computer Science Project

'An interactive web-application for music notation'

The *'Notate'* System

Progress Report

Abstract

This document details the progress of the Notate music-notation system as the first term of development draws to a close. The current stage of development is evaluated against the schedule proposed in the original specification and any deviation from the intended management of the project is explained and justified. Some alterations to the specification and a revised plan for the remainder of development are presented also.

November 27, 2007

Contents

1 Scheduled Project State **2**

2 Progress **2**
2.1 Research . . . 2
2.1.1 System Overview . . . 2
2.1.2 Server-side System . . . 2
2.1.3 Client-side System (The Score Editor) . . . 3
2.1.4 Score Data Format . . . 3
2.1.5 Score Rendering . . . 4
2.1.6 Musical Symbols . . . 4
2.2 Analysis & Design . . . 4
2.2.1 Server-side . . . 4
2.2.2 Client-side . . . 5

3 Evaluation of Progress **6**

4 Project Management **7**
4.1 Issues Encountered . . . 7
4.1.1 Volume of Research . . . 7
4.1.2 Analysis . . . 7
4.1.3 Design Methodologies . . . 7
4.2 Further Specification Alterations . . . 7
4.2.1 Quality of Editor Score . . . 7
4.2.2 Aim Change: Community-based System . . . 8
4.2.3 GUIDO Noteserver Integration . . . 8
4.3 Remainder of Development . . . 8
4.3.1 Timeline . . . 8
4.3.2 Tasks . . . 9

A Original Specification **10**

1 Scheduled Project State

Below is an excerpt from the timeline of development, as proposed in the original specification (see the Appendix for a full copy of the specification document).

	Time Period	**Expected Stage of Development**
Term 1	Weeks 1-2	Project Specfication
Deadline	**Thursday Week 2**	**Project Specification submitted**
		Begin Main Iteration
	Weeks 3-4	Phase 1: Analysis
	Weeks 5-8	Phase 2: Design
	Week 9	Work on Progress Report
Deadline	**Monday Week 10**	**Progress Report submitted**
	Week 10 and over Christmas	Phase 3: Implementation

Figure 1: Timeline proposed in the specification

This initial plan aimed for the design stage of development to be completed by the end of the first term. The progress report itself was intended to be a deliverable that would indicate the finalisation of all analysis and design project tasks. It should also mark the end of any further modifications to the specification itself (in terms of objectives and methods).

Therefore the project should now be at such a state that it is possible to immediately begin implementing the initial version ("Main Iteration") of the *Notate* music notation system.

2 Progress

Among the first tasks to be completed were the setting up of a project development blog[1] and a *Subversion* repository where all documentation, research notes and prototype code may be found. See the development blog for details about the repository.

2.1 Research

I have undertaken research and analysis over the last term, much of it specifically aimed at understanding what technologies already exist out there that are relevant to this system. The most important issues that have come up are detailed below.

2.1.1 System Overview

It became obvious fairly quickly that there was a natural modularisation of the system, dictated by the current nature of a user's interaction with the Internet: The **server-side** system and the **client-side** ('score editor') system. Although there is no requirement to formally separate these two components, it was decided early on that they should be considered as distinct, encapsulated systems. This, more structured approach, makes development easier to manage and will be simpler to test and debug. The communication between the two will be well understood and designed so that integration will be seamless from the users' point-of-view.

During this early stage of research and analysis, a very basic prototype (`Prototype-01`) was created to give some idea of the possible look and feel of the editor and how it might interact with a server. It was also useful for familiarisation with Javascript and HTML coding while researching implementation options.

2.1.2 Server-side System

Research was completed regarding what should be used to develop and implement the *server-side* system. Firstly it was decided that an *SQL* database should be maintained on the server providing an easy-to-develop-on and secure location to store (at least) all personal information for each user account. It is worth noting that,

[1] The development blog may be found at http://blogs.warwick.ac.uk/notate

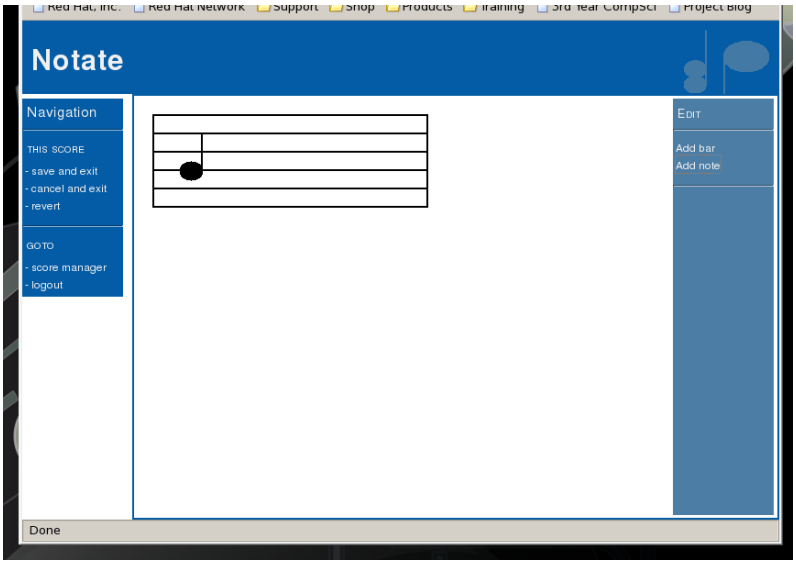

Figure 2: Screenshot of `Prototype-1`

although it would be possible to store users' actual scores in the database, it would be preferable to store them as separate files to allow for easy conversion to other formats for export (possible future extension of the project) and to keep the two halves of this system as encapsulated as possible.

Implementation Options: A selection of frameworks/languages were considered for the server-side system, amongst them *PHP*, *Ruby on Rails* and *Python*. For the most part it would have made little difference which of the most popular options I decided to use but there were several unique features of *Ruby on Rails* (*RoR*) that made it most appropriate for this project:

- **Ease-of-use:** Due to its nature *RoR* promotes succinct code and also would integrate seamlessly (and automatically) with the underlying database, even offering abstraction from SQL in most cases.
- **Agility:** Including automatic testing support and the ability to rollback system/database changes. Although my project management approach is not specifically agile, this is still an advantage which will make it easy to add further functionality in a quick, robust manner.
- **Popularity:** *RoR* is growing in popularity at the moment and the fact that it has been around for a couple of years now shows that it might not be purely hype. It is worth becoming familiar with this technology as it may be a large part of the future of web-development.

2.1.3 Client-side System (The Score Editor)

There are many options available for this side of the project, for example platforms such as *Adobe Flash* and *Silverlight*). However, the decision has been made to implement the Editor in *Javascript*. Although it would potentially impose constraints when it comes to more advanced functionality (beyond that currently suggested in the "Optional Features" section of the specification), it would have the unique property of being supported on practically every single browser-operating system combination available to users (Microsoft *Silverlight* does not currently have support on Linux for example). Javascript would run unnoticed by the average user. This is very much in the spirit of this project, whereas requiring users to install and become familiar with third-party plug-ins is not.

2.1.4 Score Data Format

Having decided to keep each score saved in a standalone file, it was necessary to decide the file format. Some research and development has already taken place about this subject[2] ,and there are several well-established

[2] "Algorithms and Data Structures for a Music Notation System based on GUIDO Music Notation", 2002, Vom Fachbereich Informatik der Technischen Universitat Darmstadt genehmigte, http://deposit.ddb.de/cgi-bin/dokserv?idn=967375975&dok_var=d1&dok_ext=pdf&filename=967375975.pdf

formats available; *GUIDO*, *NIFF*, *MIDI*, *MusicXML*, *LilyPond* to name the most prevalent few. The most promising (and recent) of these is MusicXML, which is the only one which has been designed specifically for the notation of music and is complete. Others are incomplete when it comes to the actual manuscript representation. The other advantage of MusicXML is that it is XML, and has been designed to be compatible with, and encourage the development of music-based web-applications[3]. MusicXML also stores all the data MIDI is able to, so would be sufficient if audio-playback was implemented.

There are two variants of *MusicXML*: *partwise* and *timewise*. As it is more intuitive and commonplace, *partwise* will be the variant stored by the system.

2.1.5 Score Rendering

Research into the presentation of a musical score onto a page (or screen) has been extensive (see footnote 2 on prevous page) and, for a score to be of a publishable quality the process of deciding the layout is complex. For this reason the editor will not attempt to match this high quality - it will maintain a simple a layout that is sufficient for the task. There is the possibility however of adding an option to the user of converting their score to *GUIDO* and sending it to the *GUIDO Noteserver*[4] to be converted to a high-quality manuscript.

The basic layout of musical symbols however will follow standard notation practice[5].

2.1.6 Musical Symbols

The set of musical symbols that will be supported by the system has been formally declared, in accordance with the aims of the initial version of *Notate*, as laid out in the specification[6].

2.2 Analysis & Design

2.2.1 Server-side

Overall Architecture The aforementioned overall structure of the system is illustrated in Fig. (3). Notice there are two lines of communication between the server and the client systems: One for when an entire score is loaded into the editor or saved to the server, and another - more continuous - auto-save communication system by which the editor informs the server about each change made to the score as it happens. This is implemented using Ajax.

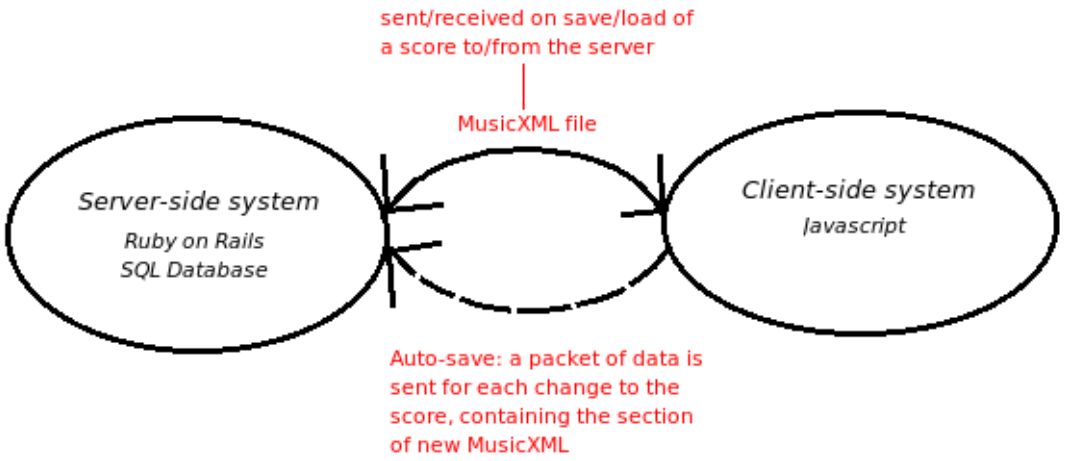

Figure 3: Overall System Architecture

[3]For a rundown of the advantages of MusicXML see the tutorial and FAQ online at http://www.musicxml.org/xml/tutorial.html
[4]Details of the GUIDO noteserver may be found at its homepage: http://www.noteserver.org
[5]"Standard Music Engraving Practice", Broido, Music Educators Journal, Vol. 52, No. 4. (Feb. - Mar.,1966), pp. 52-56+213
[6]This document may be found at the project development blog

Note the two components to the server-side system: SQL Database and the Ruby on Rails application:

Database Schema Ignoring the tables that RoR will generate in the database for itself (specific user session data for example), the database only needs to contain a personal data about each user-account. The schema for this database may be found in fig. (4).

User			
Attribute name	**Data Type**	**Constraints**	**Description**
userid	number	Primary Key	Unique id number automatically generated and associated with each account
first_name	string	required	First name of the user
last_name	string		Surname of the user
email	string	required	Email address of the user
password_hash	string		

Figure 4: Database Schema

2.2.2 Client-side

Object-Oriented Programming Although *Javascript* is not purely an object-oriented language, it is simple to simulate OOP with it and this is how the system will be designed and implemented.

Data Structure A variety of options have been considered for the design of the in-browser, *Javascript* data structure. Among them,

1. **XSLT:** In theory, since all of the scores are being received by the editor in XML format, it should be possible to develop an XSLT describing to the browser how the MusicXML could be interpreted as musical symbols in the correct places. This would use the MusicXML itself as the data structure. After some experimentation, however, it turns out that this approach is not, currently, practical as there is no easy, efficient and non-browser-specific way to enable the user to edit this XML without the change being sent via the server.

2. **Simulating MusicXML:** That is, setting up *Javascript* 'objects' that are similar to the structure of the tags in MusicXML (e.g. A 'Part' object that contains many 'Measure' objects). This is a good solution because it allows for a good integration of the data structure with the underlying MusicXML file so that a user's modifications to a score may be translated back to the underlying MusicXML (which may be sent back to the server) in a straightforward manner. Also, the structure of *partwise* MusicXML is an intuitive choice for the data as it reflects the structure of a manuscript.

The system will be designed in accordance with this second option, which is reflected in diagram (5). The objects in this design are: *Part*, *Staff*, *Measure* and *Voice*. It is not an exact simulation of MusicXML, which suffers from some constraints due to the hierarchical nature of XML.

The Voice objects will contain a doubly linked-list of *Voice-Symbol* objects (see fig. (6)), which make up each strand of music in each Measure. The Measure objects will also be stored in a doubly linked list within each Staff. The reason for choosing linked lists is because the most common requirements of the system will be to insert and delete symbols in a voice, insert and delete measures in a staff and search through the voice-symbols in a given measure (for validation of time-signatures). Linked lists will be extremely efficient for these needs. It will also not be necessary to search a list by some index (a big weakness of linked lists) because a reference to each object will be kept in the relevant HTML DOM element that the users will interact with. Therefore it will always be possible to go directly to the Javascript object being edited by the user.

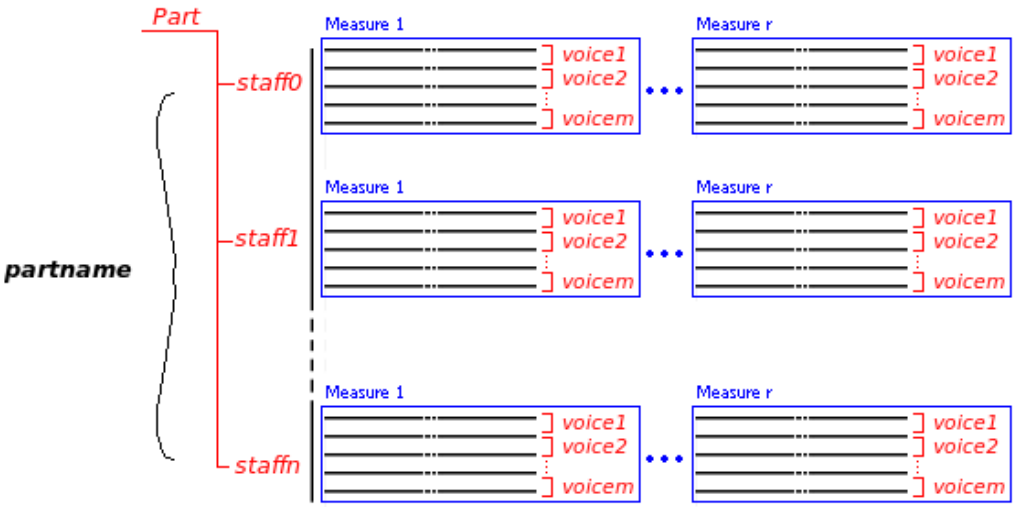

Figure 5: Strucure of Javascript Objects in the Editor

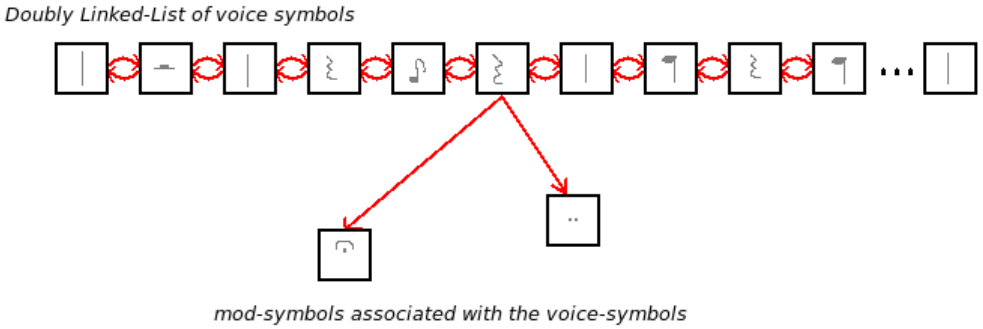

Figure 6: Doubly Linked-List of Voice Symbols in a Voice Object

During the development of the design of the client-side system, two further prototypes were developed in order to experiment with XSLTs and the implementation of a linked-list in Javascript. The code for these prototypes is available in the project repository.

3 Evaluation of Progress

The project is currently behind schedule. It was expected that the design of the main iteration would be complete by the end of the first term, but there are still design tasks left to be done, including:

1. Design the server-side, RoR system in accordance with its Model-View-Controller paradigm.
2. Obtain the 68 musical symbol graphics needed by the system
3. Complete design of client-side system

This progress report was intended to mark the end of the design phase. However, in the original specification I had scheduled the deadline for the report as a week later than it actually is. This error has effectively shortened my design phase by a week which explains why it is not complete.

Current Project State Comparing current progress against the specification, development is drawing to the end of the design phase. A few more weeks are required to complete the design of the system before the specified implementation section may begin.

4 Project Management

4.1 Issues Encountered

4.1.1 Volume of Research

The original development schedule underestimated the amount of research that would be needed to effectively design this system. It was turned out that it was necessary to investigate what had been researched and developed in the past, what data structures had been proved to work with scores and also to thoroughly educate myself about web-development, with which I have had little experience.

A decision was made to extend the Research & Analysis phases by a week (into week 5), removing a week from the time given to design work in term one. This decision was made at the beginning of week 5, and was the only modification to the overall specification timeline.

This is a significant factor of why the Design phase of development is having to run over into the next term. All in all (also considering the error in scheduling the progress report deadline) two weeks have been removed from design time this term.

4.1.2 Analysis

The specification demanded that a requirements analysis and/or a use-case analysis be produced for the system during analysis. During research and analysis this task was not completed for several weeks, as it did not seem relevant to my work at the time. I eventually decided that it would not be necessary to produce such a document as I knew well enough what the system was going to do to be able to design it, and so the general list of features in the specification was sufficient.

4.1.3 Design Methodologies

One of the obstacles I faced when starting to consider the design of the system, was acertaining what would be appropriate for each of the systems. The decision was made to essentially use object-orientation in the design of the server-side, *Javascript* system, and so some form of class design was sufficient for that system. However, more research was required into the *Ruby on Rails* system to understand its Model-View-Controller architecture so that the server-side application may be designed effectively.

This was a contributing factor in the extension of the Analysis & Research phase of development by a week and explains why design time is overflowing to next term. It also shows that the less ambiguious the development schedule is, the higher productivity will be.

4.2 Further Specification Alterations

During research, it became apparent that certain changes to the specification of the system would benefit both developer and user. The alterations considered are detailed below.

4.2.1 Quality of Editor Score

It soon became clear that the mathematics behind publishable-quality music scores is not trivial and it would be inappropriate for the javascript code to attempt to achieve such high standards within the editor. It would be difficult to design and implement and would create noticeable slowdown to the user compared to a simpler approach. It would also, most likely, make the users' interface with their score less intuitive.

For this reason, the client-side editor system will be required to display the score in a simple way, fixing values (such as the number of measures to a line) to a constant value (g. 4). Specifically, the system will not attempt to resize the width of measures dynamically as the user ads symbols to the scor - it will be fixed at all times. Although this will create aesthetically unbalanced music, it will be easier and more intuitive for the user to edit, which is the overriding aim of the editor system.

4.2.2 Aim Change: Community-based System

The proposal was to change the aim of the system so as to place the emphasis on creating a web-community of users where there would be the sharing of sheet-music among user profiles. Users would be allowed to view, comment on and perhaps even edit each other's work.

The advantages and drawbacks of adopting this aim change were considered in the fourth week[7]. While aiming for this system would improve the originality, and potential popularity of the project, it would also require much more development work, as there would be far more objectives to be implemented in the initial version. Also these objectives would not necessarily pose a sufficient challenge for the Third Year Computer Science Project module.

It was decided by the beginning of week 5 that the aim of the project would remain unchanged for these reasons. However the idea is considered as being a possible eventual, future aim of the system. The nature of *Ruby on Rails* would make the design and implementation of a user-based web-community, as an upgrade to the initial version fairly straightforward.

4.2.3 GUIDO Noteserver Integration

The GUIDO noteserver[8] provides the facility to send musical scores (encoded in the GUIDO format) to the server where they will be automatically converted to publishable-quality images of the manuscript. It would be possible to provide a facility to the user to request such an image, and the system would automatically convert the MusicXML to GUIDO, send it away to be converted and return to the user the result.

Due to the requirement for the design of a MusicXML-GUIDO converter program, it was too late in the design stage of the project to encorporate this objective into the initial version. However, it stands as an optional feature, should there be time to perform a further iteration.

4.3 Remainder of Development

Due to design phase remaining unfinished, the timeline for the remainder of development needs to be revised to allow for more design time before implementation begins. Following is a new, revised development scedule for the remainder of the project and details of the specific tasks left to perform.

4.3.1 Timeline

	Time Period	**Expected Stage of Development**
Christmas		Design work
Term 2	Weeks 1-2	Complete Design phase
	Weeks 3-6	Phase 3: Implementation
	Weeks 7-10	Phase 4: Testing
	Weeks 7-9	Prepare presentation
Over Easter		Phase 4/5: Testing & Refactoring
		Optional Iterations (as time allows)
		Work on final report
Term 3	Weeks 1-2	Work on final report
Deadline	**Thursday Week 2**	**Final Report submitted**

[7]The proposal document may be found at the development blog: http://blogs.warwick.ac.uk/files/notate/specchange1.pdf
[8]The homepage of the GUIDO noteserver may be found at `http://www.noteserver.org`

4.3.2 Tasks

Although tasks were specified in the original specification document, it is helpful to refine them into more specific jobs left in each phase. This can be done because, having performed research, I understand more now about the design methods that are appropriate for the platforms the system is being developed on.

1. **Design**
    - 1.1 Design of *Javascript* objects
    - 1.2 Event-based javascript functions (user interaction) in editor
    - 1.3 Design of server-side models, controllers and views
    - 1.4 Test cases for server-side system and client-side system
    - 1.5 Design the integration of the *Javascript* files with *RoR* system
2. **Implementation**
    - 2.1 Create the musical symbols required
    - 2.2 Code the client system
    - 2.3 Code the server-system
    - 2.4 Integrate the two systems
3. Testing
    - 3.1 Apply test cases
    - 3.2 Generate & perform usability tests
    - 3.3 Specify resulting bug corrections
    - 3.4 Recommend design changes
4. Refactoring
    - 4.1 Correct bugs
    - 4.2 Re-perform relevant tests

Further Iterations Any further iterations, if there is time for them, will proceed as proposed in the specification (see Appendix).

A Original Specification