An Integrated MIR Programming and Testing Environment

Jörg Garbers
Department of Computer and Information Sciences
Utrecht University
garbers@cs.uu.nl

Abstract

The process of shaping a music information retrieval algorithm is highly connected with implementing it and testing suitable parameterizations. Often music information retrieval scientists do not have a programmer at hand and must implement their experimental setup themselves. This paper describes an integrated tool setup *OHR* consisting of the music (analysis) systems *OpenMusic*, *Humdrum* and *Rubato* and a system for form based parametrization and comparison of algorithms. These packages and their programming environments provide the scientist with frameworks and existing libraries for implementing and testing algorithms. They differ in the programming languages that they support and in the type of testing user interfaces that they allow the scientist to build easily. The systems and their components are integrated by using their scripting languages. We sketch an example of the integrated use of these systems.

Keywords: Computational music analysis, development environments, scientific programming, software integration

1. Introduction

The process of shaping a music information retrieval algorithm is highly connected with implementing it and testing it. Algorithm implementation is usually done in one or more general purpose programming language(s) with or without the help of an integrated programming environment and with respect to existing libraries. Effective testing however requires the scientist him/herself to build a time consuming graphical or text based interface or to embed his/her algorithm into existing testing systems. That is where experimenting environments come in that provide us with ways of parameterizing algorithms, of applying them to data and of visualizing results.

We give a short overview of the systems *OpenMusic*, *Humdrum*, *Rubato*® and their integration *OHR*. The systems have their specific strengths and weaknesses, technical barriers and programming skills requirements. We show how to integrate them, so that one can use both the systems' interactive parameterization front ends and their functional back ends. Then we describe the general purpose resource comparison framework of the *Reisewissen* project and its ability to call into scriptable systems. Finally, we give an example of using the resulting system as an experimentation framework for comparing similarity algorithms.

2. OHR

The *OHR* system was an effort to integrate *OpenMusic* [1], *Humdrum*[2] and *Rubato*[3], and to make each system's algorithms reusable from each of the other systems. Why and how this was done is described in detail in [4, 5].

Essentially the approach uses the scripting facilities of each of the systems, which make available *expression evaluators* that are close or identical to the components implementation languages:

- The *Humdrum* tools can be invoked by (remotely) evaluating shell scripts. This allows users as well as the other systems to access and filter a large database of musical scores. Humdrum users can help other scientists to construct such scripts.
- *Rubato's* algorithms and components can be accessed via *FScript*, which provides wrappers for *Apple's Cocoa* libraries and user defined *Objective-C* classes. Besides simple evaluation of scripts, external modules can also change the state of the running *Rubato* application. Scientists who have a Macintosh computer can build their testing graphical user interfaces with Apple's free *XCode* IDE and *Interface Builder*.
- Functions that are implemented by the user in *OpenMusic* as graphical *Patches* or ordinary *Lisp* functions can be called via the underlying *Lisp* system.

In its extended version in *OHR OpenMusic* comes with routines for generating scripts and for sending them to *Humdrum* or *Rubato*. This allows users to graphically define functions that use arbitrary functionality of the three systems. So scientist are not required to have other programming environments but *OpenMusic*. From there they can use the other systems' components in ways that are often not anticipated by their developers.

3. The Reisewissen testing system

The *Reisewissen* testing system is described in detail in [6]. In short its graphical user interface allows users to compare

the performance of *parameterizable evaluators* and combinations of their results with respect to a list of resources (e.g. scores). The results are presented in an (*evaluator x resource*) table. Each evaluator instance comes with its own window in which the user enters the parameters and inspects details of the results. This setup is not only appropriate for testing within the hotel booking domain [7], where the tool was developed, but can also be used to do music similarity studies: An evaluator implements a melodic, harmonic or rhythmic distance between two scores. The query score is given as a parameter and compared with the database scores. The distances are presented in the table.

Writing an evaluator with a new form based front end is relatively simple and requires only basic knowledge of *Java*. *Swing* calls or *HTML* code are automatically generated from the programmer's form description, so scientists can concentrate on the algorithm. However, to use the system the scientist is not required to know *Java*. Instead there are general purpose scripting connectors for *SWI-Prolog* and *Lisp* based systems. The latter allows us to call as a function any *Patch* from *OpenMusic*, including *Patches* that call *Rubato*, *Humdrum* or one of the *IRCAM* synthesis applications that are callable from *OpenMusic*.

4. Example

To build and test a new complex similarity definition we might proceed as follows:

- To exclude ornaments from comparison, we apply a metrical filter. Therefore we use *Humdrum's metpos* command or *Rubato's MetroRubette* [8]. Notes on onsets with low metrical profile are omitted.
- *Humdrum's yank* and *extract* are used to create a reduction of the score segments to be compared.
- *Rubato's HarmoRubette* [9] is used to compute a functional harmonical analysis of the reduced scores. We play with parameters in the interactive user interface and store different harmonic theory configurations.
- *OpenMusic* is used as a front end to define distance functions and bind everything together: We play with several definitions for the distance between two harmonic sequences. A *distance Patch* takes two named score files and produces a real number. To test this patch quickly, some score files are named directly inside a *testing Patch* and the *distance Patch* is applied to them via *mapcar*, with one bound parameter.
- We run our hybrid algorithm from the *Reisewissen* testing system: To simplify the visual comparison, we may preorder the scores according to our similarity expectations. We use the system to compare our results with the results of existing similarity implementations, such as [10].

5. Outlook

The described systems will be used and improved within the *WITCHCRAFT* project at the *Utrecht University* and the *Meertens Instituut*. We will integrate them into the MUUGLE project (see poster by Martijn Bosma). We will evaluate the integrated system and its components with respect to scientific programming needs and its usability for end users and power users. On the basis of computational experiments on music we hope to stimulate the discussion between music theorists and the music information retrieval community. See http://www.cs.uu.nl/research/projects/witchcraft.

6. Acknowledgments

We want to thank those who have developed software that is used directly in this work: Guerino Mazzola, Oliver Zahorka, Carlos Agon, Philippe Mougin, David Huron and Magnus Niemann. Thanks also to my new colleagues in Utrecht for feedback.

References

[1] C. Agon, G. Assayag, M. Laurson, and C. Rueda, "Computer assisted composition at Ircam: Patchwork & OpenMusic," *Computer Music Journal*, 1998. [Online]. Available: http://www.ircam.fr/equipes/repmus/RMPapers/CMJ98/

[2] D. Huron, "Music information processing using the Humdrum toolkit: Concepts, examples, and lessons," *Computer Music Journal*, vol. 26, no. 2, pp. 11–26, 2002.

[3] G. Mazzola and O. Zahorka, "The RUBATO performance workstation on NeXTSTEP," *ICMA (ed.): Proceedings of the ICMC 94, S. Francisco*, 1994.

[4] J. Garbers, "Integration von Bedien- und Programmiersprachen am Beispiel von OpenMusic, Humdrum und Rubato," Ph.D. dissertation, Fakultät IV – Elektrotechnik und Informatik der Technischen Universität Berlin, 2004.

[5] ——, "User participation in software configuration and integration of OpenMusic, Humdrum and Rubato," *Lluis-Puebla, Emilio, Guerino Mazzola und Thomas Noll (eds.): Perspectives in Mathematical and Computer-Aided Music Theory, Verlag epOs-Music, Osnabrück*, 2003.

[6] J. Garbers and M. Niemann, "The Reisewissen testing system," FU-Berlin, Tech. Rep., 2006, to be published at http://reisewissen.ag-nbi.de/en/.

[7] J. Garbers, M. Niemann, and M. Mochol, "A personalized hotel selection engine," *Proceedings of the European Semantic Web Conference 2006*. [Online]. Available: http://www.eswc2006.org/poster-papers/FP15-Mochol.pdf

[8] A. Fleischer, "Die analytische Interpretation, Schritte zur Erschließung eines Forschungsfeldes am Beispiel der Metrik," Ph.D. dissertation, Philosophische Fakultät der Humbold-Universität zu Berlin, 2002.

[9] J. Garbers and T. Noll, "Harmonic path analysis," *Lluis-Puebla, Emilio, Guerino Mazzola und Thomas Noll (eds.): Perspectives in Mathematical and Computer-Aided Music Theory, Verlag epOs-Music, Osnabrück*, 2003.

[10] F. Wiering, R. Typke, and R. Veltkamp, "Transportation distances in music notation retrieval," *Computing in Musicology 13, 113-128*, 2004.