Providing Music Notation Services over Internet

Mike SOLOMON, Dominique FOBER, Yann ORLAREY and **Stéphane LETZ**
Grame
Centre national de création musicale
Lyon - France
mike@mikesolomon.org, {fober, orlarey, letz}@grame.fr

Abstract

The GUIDO project gathers a textual format for music representation, a rendering engine operating on this format, and a library providing a high level support for all the services related to the GUIDO format and it's graphic rendering. The project includes now an HTTP server that allows users to access the musical-score-related functions in the API of the GUIDOEngine library via uniform resource identifiers (URIs). This article resumes the core tenants of the REST architecture on which the GUIDO server is based, going on to explain how the server ports a C/C++ API to the web. It concludes with several examples as well as a discussion of how the REST architecture is well suited to a web-API that serves as a wrapper for another API.

1 Online musical editing

As client-server models for the processing, visualizing and analysis of data become more widespread in mobile computing (WordPress, YouTube, Instagram, SoundCloud), music engraving has entered the fray with various web-based score editing services. The GUIDO HTTP server merges the idea of a web-based music editor with a RESTful web service in order to expose the public API of the GUIDOEngine library[Hoos and Hamel, 1997]. This section explores several categories of online musical editing services, concluding with a discussion of general trends in current technologies and the main problems that the tool outlined in this paper – the GUIDO HTTP server – seeks to address.

1.1 Online music notation editors

As of the writing of this paper (2014), there are three main online musical score editors – Noteflight[1], Melodus[2] and Scorio[3]. Noteflight and Melodus seek to provide a full-featured music editing platform online, similar to Google Documents' role in the world of office suites. Scorio is a hybrid tool that mixes rudimentary layout via a mobile editing platform with publication-quality layout via JIT compilation through LilyPond when possible.

1.2 Online score sharing software

Several music tools, such as Sibelius[4], MuseScore [Bonte, 2009], Maestro[5], and Capriccio[6], offer online services where scores composed using this software can be uploaded, browsed, and downloaded online. Capriccio, can be run online in limited form as a Java applet. MuseScore, Sibelius, and Maestro allow for automatic score/MIDI synchronisation of embedded files.

1.3 Online music JIT compilation services

WebLily[7], LilyBin[8], and OMET[9] are all JIT compilation services that run the LilyPond executable to compile uploaded code and return embedded SVG, canvas or PDF visualizations depending on the tool. The GUIDO note server [K. and Hoos, 1998] uses the GUIDOEngine library to compile Guido Music Notation Format [Hoos et al., 1998] strings into images.

1.4 A RESTful alternative

All of the tools describe above facilitate the creation or visualization of scores via a variety of input methods (WYSIWYG, text, MusicXML etc.) but are not designed to facilitate

[1] http://www.noteflight.com
[2] http://www.melod.us
[3] https://scorio.com
[4] http://www.sibelius.com
[5] http://www.musicaleditor.com
[6] http://cdefgabc.com
[7] http://weblily.net
[8] http://www.lilybin.com
[9] http://www.omet.ca

low-latency server-client exchanges of score-related information. This is, in part, due to the fact that the majority of automated music engraving programs do not offer public APIs and are not designed to provide end-user information other than visual representations of scores and various non-human-readable file formats. The GUIDO Engine API [Daudin et al., 2009] [Grame, 2014b] seeks to remedy this issue by offering a public API that reports information about scores such as the number of pages, duration, and the placement of musical events both in time and on the page. The representational state transfer [Fielding, 2000], or REST, architectural style, is well suited for the porting of an API to the web because it is optimized for a system that is *stateless*, meaning that it does not require remembering intermediary states of a user. Contrast this to, for example, a server that needs to retain an undo history or the state of a logged-on user. As a result, the design of the server is clearer, quick and easy to scale [Richardson and Ruby, 2008]. This is further discussed in Section 3 and Section 4. The GUIDO HTTP server thus fills a gap in online score editing technology similar to the gap filled by Atom web feeds in news services.

2 Representational state transfer

Representational state transfer [Fielding, 2000] is an ubiquitous contemporary server architecture style [Richardson and Ruby, 2008]. The REST architecture is intended as a set of constraints to facilitate exchange in systems that deliver and report on hypermedia resources. The architectural style is based on a traditional *client-server* model with the design trade-off that the server is *stateless*, meaning that all of the information required to process a request is contained in the request itself and the server does not need to store intermediary states. In order to speed up interaction with the server, the REST architecture calls for *client-side caching* of data, which can potentially eliminate certain redundant server requests. It also calls for a *uniform interface*, harmonizing all applications' interactions with the server at the expense of application-specific interaction models that could speed up exchanges. *Layering* is possible in this model, with intermediary servers translating various forms of shorthand into longer or less human-readable server commands. With this layering comes the constraint that exchanging agents cannot "see" beyond the layer with which they are communicating. As the burden on the client to be server-compliant is high in REST, the architectural style provides an optional constraint of servers' offering downloadable *code-on-demand* (scripts, applets, etc.) to ease client-side software development.

Certain specific architectural elements are put into place in order to facilitate the above-described architecture. In addition to the transferring of *data*, REST calls for the transferring of *meta-data* about a server response. This allows for the client side to have information about how to de-encode the response without needing to send specific de-encoding instructions. REST also encourages resource requests that are constructed in a hierarchical and human-readable manner. For example, accessing today's weather in Lyon, France is preferably

```
http://website.fr/France/Lyon/weather/today
```

rather than

```
http://website.fr/?country=France&town=Lyon
&feature=weather&date=today
```

A server compliant with the REST architecture is said to be a RESTful server.

3 The GUIDO HTTP server : an overview

The GUIDO Hypertext Transfer Protocol Daemon (HTTP) server is a RESTful server that compiles strings written in the GUIDO Music Notation (GMN) Format into musical scores and reports to the client several representations of this data.[10] It accepts user requests via two main methods of the HTTP protocol: POST, used to place elements on the server, and GET, used to retrieve information about elements on the server.

3.1 The POST method

POST, as implemented by the GUIDO server, is RESTful insofar as it does not save any information about the user state and only saves information sent by the user.

Assuming that a GUIDO HTTP server is running on the subdomain `http://guido.grame.fr` on port 8000, a POST request containing GMN code `[a b c d]` is sent via `curl` as follows:

```
curl -d"data=[a b c d]" http://guido.grame.fr:8000
```

[10] In this paper, the terms "GMN" and "score" are used interchangeably when talking about music treated by or stored on the server.

Assuming that the GMN code is valid, response, in JSON, gives the user a unique identifier generated using an SHA-1 tag corresponding to the input file. This ensures that the server will not store the same information multiple times:

```
{
 "ID": "07a21ccbfe7fe453462fee9a86bc806c8950423f"
}
```

This identifier is generated via the SHA-1 cryptographic hashing algorithm [Gallagher, 2012] that encodes any digital document as a 160-bit hash or key. The algorithm has a low incidence of collision ($\frac{1}{2^{63}}$), making it almost impossible for two documents to share the same SHA-1 key.

This is the server's internal representation of the GMN code and used for all subsequent requests to the server. To access it, it is appended onto the URI. The following is a simple request using the SHA-1 tag (hereafter shortened to facilitate readability) that results in the image seen in Figure 1.

```
curl http://guido.grame.fr:8000/07a21...0423f
```

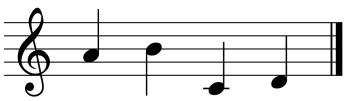

Figure 1: Score with SHA-1 tag `07a21...0423f`.

Technically speaking, the need to use an SHA-1 key in order to access scores and score-related information is not strictly RESTful. A strictly RESTful implementation would embed the score in every GET request. In accepting a GMN score via POST, the server must "remember" the score, which violates the principle of statelessness. The posting of a resource on the server is generally considered an acceptable compromise [Richardson and Ruby, 2008] so long as it is uniquely identifiable in an URI and the resource cannot be modified once uploaded on the server. This is the case with scores on the GUIDO server.

3.2 The GET method

Requests sent via GET query the server for information about scores. The main return type is JSON for all queries related to information about a score, MIDI for midi realizations of the score, and PNG for all queries asking for visual representations of the score itself. The latter is also possible in JPEG and SVG. All return types are specified in meta-data as per REST guidelines (see Section 2).

3.3 Uniform interface

The RESTful style specifies that a server's interface must be uniform, meaning that the operations that it executes must be the same for all clients interacting with the server. Furthermore, these operations should be conceptually different with no overlap and should ideally be widely used. The HTTP standard provides several atomic options that allow for the uniform interaction with a server [Richardson and Ruby, 2008]. The GUIDO web API uses the GET and POST methods from HTTP via libmicrohttpd [Grothoff, 2014], leaving out less widely-used methods such as PUT and DELETE in an effort to expose its full functionality to the largest group of client applications possible.

4 The GUIDO HTTP server as an API

The GUIDO HTTP server attempts to expose as much of the public API of the GUIDO Engine as possible, implementing one-to-one equivalencies with its functions when possible. Arguments are passed to these functions via optional key-value pairs in the URI's query part. Defaults are provided for all key-value pairs in case of omission. An exhaustive overview of the API can be found in the GUIDO HTTP server's documentation[Grame, 2014a].

This section aims to discuss some of the broad decisions made in exposing a C++ API via a web interface, giving three exhaustive examples at the end showing how the API is exposed.

4.1 SHA-1 key as musical score

Section 3.1 entertains the manner in which SHA-1 keys replace GMN scores in URIs sent to the server via in order to avoid having to send GMN scores in GET requests. This key corresponds to both an `ARHandler`, or *Abstract Representation*, and `GRHandler`, or *Graphic Representation* of a score in the GUIDO API. These two structures are used in order to generate information about the musical contents of a score (`ARHandler`) as well as its layout (`GRHandler`). The representation of both structures by one SHA-1 key allows the user to have a unique point of entry for each GMN score that conflates the data generated by several structures.

4.2 Function as URI segment

A function in the GUIDO public API is represented as a segment of the URI sent to the server.

C/C++ API	URI segment	scope
GuidoGetPageCount	pagescount	score
GuidoGetVoiceCount	voicescount	score
GuidoDuration	duration	score
GuidoFindPageAt	pageat	score
GuidoGetPageDate	pagedate	score
GuidoGetPageMap	pagemap	score
GuidoGetSystemMap	systemmap	score
GuidoGetStaffMap	staffmap	score
GuidoGetVoiceMap	voicemap	score
GuidoGetTimeMap	timemap	score
GuidoAR2MIDIFile	midi	score
GuidoGetVersionStr	version	engine
GuidoGetLineSpace	linespace	engine

Table 1: GUIDO API public functions and their representations as URI segments.

For example, the function `GuidoGetPageCount` in the GUIDO public API is represented as the URI segment `pagescount`.

The GUIDO public API provides two generic categories of functions:

- Functions addressed to the engine and reporting information about GUIDO.
- Functions addressed to a specific score processed by GUIDO.

With the C/C++ API, functions addressed to a score take *score handlers* as argument, which may be viewed as pointers to the internal score object. With HTTP, the SHA-1 tag plays the role of these score *score handlers* and the complete URI defines the scope of the request :

- Requests addressed to the engine are not prefixed.
- Requests addressed to a specific score are prefixed by the SHA-1 key.

For example,

```
http://guido.grame.fr:8000/version
```

reports the version of both GUIDO and the GUIDO server. On the other hand, the URI

```
http://guido.grame.fr:8000/<key>/voicescount
      where <key> is a SHA-1 key
```

exposes the API function `GuidoCountVoices` via the URI segment `voicescount`, giving the voice count of specific score.

Table 1 contains a succinct list of the servers' naming conventions showing the name of a function in the GUIDO public API, its representation as a server URI segment, and it's scope. Note that the only generic URI segment that does not correspond to a GUIDO public API function is `server`, which gives the version number of the server and thus is not related to the GUIDO API proper.

4.3 Arguments as key-value pairs

Several of the API functions listed in Table 1 require arguments in order to generate results. For example, the function `GuidoGetStaffMap` requires an argument `staff` specifying the staff for which the map should be generated. These arguments are specified in key-value pairs in the URI.

```
http://guido.grame.fr:8000/<key>/staffmap?staff=1
```

Default arguments are provided for all argument-taking functions in case the user fails to specify an argument. These arguments are values that would work in the majority of scores (for example, `page=1`) and often come from defaults provided in the API.

4.4 Layout and formatting options as key-value pairs

The GUDIO server allows for the specification of several parameters relating to the layout and formatting of scores as key-value pairs. These parameters are used in several different ways in the GUIDO public API. Some, such as `topmargin`, are become values of structures such as `GuidoPageFormat`. Others, such as `resize`, represent calls to functions that effect layout (in this case `GuidoResizePageToMusic`). Yet others, such as `width`, are used at several points in the layout process depending on the chosen backend. Rather than devising separate URI construction conventions to represent different layout and formatting information in GUIDO, all layout and formatting options are implemented as key-value pairs to make interacting with the server uniform in keeping with RESTful style.

4.5 Return values

In order to handle the diversity of return types provided by the GUIDO API, the server attempts to find MIME types that best approximate the values returned by API functions. Sometimes, there is a direct correspondance. For example, the formats of images returned by the GUIDOEngine library when compiled with Qt (JPEG, PNG and SVG) are all MIME types.

In many cases, the GUIDO API returns custom structures that have no MIME type equivalent. In these cases, JSON [Crockford, 2013]

is used to represent hierarchical relationships contained within these structures.

For example, the `Time2GraphicMap` struct is a composite structure consisting of pairs of `TimeSegment` and `FloatRect` structures. `TimeSegment` corresponds to beginning and end of a musical event whereas `FloatRect` corresponds to its placement on the page. To represent these structures in server responses, the GUIDO server uses JSON where key-value pairs correspond to a structure's element's name and its value. An example of this is given in Section 4.6.3, `time` corresponds to a `TimeSegment` and `graph` corresponds to a `FloatRect`.

4.6 Examples

4.6.1 voicescount

The command `voicescount` returns the number of voices in a score. It exposes the GUIDO Engine API method `GuidoCountVoices`. For example, the request:

```
http://guido.grame.fr:8000/<key>/voicescount
```

yields the following result:

```
{
  "<key>": {
    "voicescount": 1
  }
}
```

where "<key>" is the SHA-1 key given by the URI.

4.6.2 pageat

The command pageat returns the page given a specific date, expressed as a rational number. It exposes the GUIDO Engine API method `GuidoFindPageAt`. For example, the request:

```
http://guido.grame.fr:8000/<key>/pageat?date=1/4
```

yields the following result:

```
{
  "<key>": {
    "page": 1,
    "date": "1/4"
  }
}
```

4.6.3 staffmap

The command staffmap returns a map of the space each element of a given staff takes up in 2D space (represented by a box) and time space (represented as an interval of rational numbers). It exposes the GUIDO Engine API method `GuidoGetStaffMap`. For example, the request:

```
http://guido.grame.fr:8000/<key>/staffmap?staff=1
```

yields the following result, abbreviated below to minimize its space on the page:

```
{
  "<key>": {
    "staffmap": [
      {
        "graph": {
          "left": 916.18,
          "top": 497.803,
          "right": 1323.23,
          "bottom": 838.64
        },
        "time": {
          "start": "0/1",
          "end": "1/4"
        }
      },
      .
      .
      .
      {
        "graph": {
          "left": 2137.33,
          "top": 497.803,
          "right": 2595.51,
          "bottom": 838.64
        },
        "time": {
          "start": "3/4",
          "end": "1/1"
        }
      }
    ]
  }
}
```

5 Conclusion

The GUIDO HTTP server uses RESTful architectural principles such as statelessness, a uniform interface and a separation of client-server functionality in order to provide low-latency information retrieval. Information corresponds to uploaded GMN scores, encoded as various MIME types and transmitted via the HTTP protocol. The server exposes the robust GUIDO Engine public API via an interface based on standardized URI construction. It is intended for use by various applications needing to visualize musical scores and process score-related data. It is especially well-suited as an alternative to embarking libraries or external applications in score processing software. As cloud computing and mobile human-computer interaction becomes more common, this form of data transmission and processing is increasingly necessary. The GUIDO HTTP server in-

tends to fill this by following RESTful architectural recommendations that have proven successful in other server-based services.

The GUIDO project is an open source project hosted by sourceforge[11]. The GUIDO HTTP server is running at

`http://guidoservice.grame.fr/`.

6 Acknowledgements

This work has been realized in the framework of the INEDIT project that is supported by the French National Research Agency [ANR-12-CORD-0009].

References

T. Bonte. 2009. MuseScore: Open source music notation and composition software. Technical report, Free and Open source Software Developers' European Meeting. `http://www.slideshare.net/thomasbonte/musescore-at-fosdem-2009`.

D. Crockford. 2013. The json data interchange format. Technical report, ECMA International, October.

C. Daudin, D. Fober, S. Letz, and Y. Orlarey. 2009. The Guido Engine - a toolbox for music scores rendering. In *Proceedings of the Linux Audio Conference 2009*, pages 105–111.

R. Fielding. 2000. *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. thesis, University of California, Irvine.

P. Gallagher. 2012. Secure hash standard (shs). Technical report, National Institute of Standards and Technology, March.

Grame, 2014a. *Guido Engine Web API Documentation v.0.50*.

Grame, 2014b. *GuidoLib v.1.52*.

C. Grothoff. 2014. GNU libmicrohttpd: a library for creating an embedded http server. `http://www.gnu.org/software/libmicrohttpd/index.html`.

H. H. Hoos and K. A. Hamel. 1997. The GUIDO Music Notation Format Specification - version 1.0, part 1: Basic GUIDO. Technical report TI 20/97, Technische Universitat Darmstadt.

H. Hoos, K. Hamel, K. Renz, and J. Kilian. 1998. The GUIDO Music Notation Format - a Novel Approach for Adequately Representing Score-level Music. In *Proceedings of the International Computer Music Conference*, pages 451–454. ICMA.

Renz K. and H. Hoos. 1998. A Web-based Approach to Music Notation Using GUIDO. In *Proceedings of the International Computer Music Conference*, pages 455–458. ICMA.

L. Richardson and S. Ruby. 2008. *RESTful Web Services*. O'Reilly Media.

[11] `http://guidolib.sf.net`