PROVIDING A RESTFUL MUSIC NOTATION WEB API

M. Solomon, D. Fober, Y. Orlarey, S. Letz
Grame
Centre national de création musicale
mike@mikesolomon.org - {fober, orlarey, letz}@grame.fr

RÉSUMÉ

The Guido HTTPD Server is a RESTful server that exposes many elements of the public GUIDO Engine API to clients. This article resumes the core tenants of the REST architecture and goes on to explain the functioning of the GUIDO server with several examples, concluding with potential applications of the server.

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. 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 HTTPD 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, Melodus and Scorio. 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 publishing-quality layout via JIT compilation through LilyPond when possible.

1.2. Online score sharing software

Several music tools, such as Sibelius, MuseScore [1], Maestro, and Capriccio, 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, LilyBin, and OMET 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 [5] uses libGUIDOEngine to compile Guido Music Notation Format [4] 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 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 [2] 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 [6], or REST architectural style, is well suited for exposing this public API via the web because of several features such as its use of both data and metadata so that data can be decoded by clients, its insistence on a uniform interface, and its stateless functioning. This is further discussed in Section 4 and Section 5. The GUIDO HTTPD 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 [6] is an ubiquitous contemporary server architecture style [7]. The REST architecture is elaborated as a response to existing hypermedia systems and is intended as a set of constraints to facilitate exchange in these systems. 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. Like other aspects of REST, this is intended to encapsulate all information in a single request made to a single agent. 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 encourages resource requests that favor the retrieval of conceptually relevant entities rather than specific entities at a point in time. For example, a request to the server for "best movies of 1964" or "weather in Lyon" should change with time rather than pointing to an arbitrary entity responding to this request at a given time.

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

3. THE GUIDO HTTPD SERVER : AN OVERVIEW

The GUIDO Hypertext Transfer Protocol Daemon (HTTPD) server is an austerely RESTful server that compiles strings written in the GUIDO Music Notation (GMN) Format and reports to the client several representations of this data. 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 HTTPD 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
```

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 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:

```
curl http://guido.grame.fr:8000/
07a21ccbfe7fe453462fee9a86bc806c8950423f
```

It results in the image seen in Figure 1.

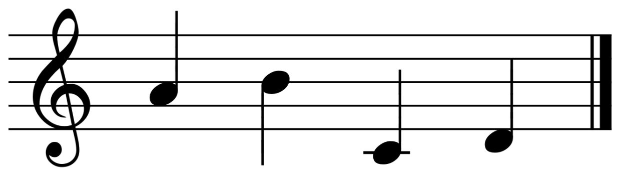

Figure 1. Score with SHA-1 tag `07a21ccbfe7fe453462fee9a86bc806c8950423f`.

3.2. The GET method

GET requests 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).

4. THE GUIDO HTTPD SERVER AS A RESTFUL SERVER

To summarize the main tenants of a restful posited by Fielding [6], a restful architecture implements the following design features :

1. Client-Server Interaction
2. Stateless
3. Client-Cache
4. Uniform Interface
5. Layered System
6. Code-on-Demand (optional)

The GUIDO server implements all architectural elements of a RESTful server. Its design also encourages client-side RESTful practices. The following subsections treat each criteria above individually, concluding with a discussion how the GUIDO server uses HTTP as well as a comparison to previously-discussed web-based musical engraving services.

4.1. Client-Server

The GUIDO server performs only server-related tasks insofar as it processes incoming requests and sends responses. It performs no client-side tasks. The server does not needed provide clients with tools for inputting requests or decoding responses. This is because the server is a HTTP server, meaning that client-side HTTP connectors [6], such as libcurl or a web-browser, can send and retrieve data.

4.2. Stateless

The GUIDO server retains no information about users' queries related to scores, nor does it rely on previous states in order to generate its current state. It clearly separates the posting of scores via the POST method (see Section 3.1) versus the retrieval of data about those scores via the GET method (see Section 3.2).

4.3. Client-Cache

The RESTful recommendation calls for the ability of clients to cache common requests, which is beyond the purview of the GUIDO server. However, Fielding [6] also describes the possibility of a server cache in order to speed up server interaction. However, the GUIDO server does not have a cache for GET requests save an Apache-like log file. This is because the processing time of GET requests is not necessarily slower than reading from a cache. The GUIDO server does, however, maintain a small internal cache of recently consulted abstract and graphical representations of [2] scores in order to avoid persistent GMN complication.

4.4. Uniform Interface

This is, according to Fielding [6], the most important aspect of RESTful architecture. The principle is also at the core of the GUIDO server. Rather than providing several ways of doing the same thing, the GUIDO server only ever provides one way of interpreting requests and representing information. Furthermore, the way data is retrieved is uniform for similar requests and derives from the mapping of URI segments to API functions (see Section 5). The inconvenience of this is that URIs sent to the GUIDO server are long and only partially readable (the SHA-1 tag generated after POST requests is not - see Section 3.1). This difficulty can be mitigated by layered systems that provide score-access shortcuts.

4.5. Layered System

For all but the most technologically savvy of users, a layered system is required to use the GUIDO server. For example, users wishing to map a GUIDO-generated PNG to an HTML canvas would need a library that translates GUIDO dates and boxes into programmatic JavaScript. This is because the server is a lightweight wrapper around the GUIDO public API, which is itself created to provide generic structures that allow for the easy manipulation of musical information.

4.6. Code-on-Demand

The GUIDO server does not currently provide Code-on-Demand as its functionalities are intended to have almost one-to-one correspondence with the GUIDO public API. As a future project, a group of best-practice calls to the server in JavaScript, curl, and wget can be elaborated as well as language-specific response parsers. The decision to send all GET information via JSON (save graphical representations of scores) facilitates parsing that, to an extent, eases the need to provide code to developers and can be seen as an architectural decision in the spirit of code-on-demand.

4.7. HTTP

According to Fielding [6], a RESTful server must be able to send both data and meta-data in responses. The GUIDO server uses HTTP in order to do this via libmicrohttpd [3], sending both a MIME type and data in its responses.

4.8. REST and online music engraving

As stated in Section 1.4, the GUIDO HTTPD server seeks to provide a full implementation of the RESTful architecture recommendations for server constructions in order to facilitate the transfer of visual score representations and score-related data. It offers stateless, server-only functionality via a uniform interface. While this is done at the expense of application-specific shortcuts and human-interpretability, it favors parsing by intermediary tools and a robust palate of extractable data. In this way, it is ideal for web-based and mobile applications that wish to outsource score-data calculation to a server instead of linking to a library or running an executable. The next section describes how this data is extracted via the exposure of GUIDO Engine's public API via a standardized URI-construction system.

5. THE GUIDO HTTPD SERVER AS AN API

The GUIDO HTTPD 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 HTTPD server's documentation[8].

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.

5.0.1. Function as URI segment

A function in the GUIDO public API is represented as an segment of the URI sent to the server. 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 reporting information about GUIDO.
- Functions reporting information about a specific score processed by GUIDO.

API Function	URI segment	score?
GuidoGetPageCount	pagescount	Yes
GuidoGetVoiceCount	voicescount	Yes
GuidoDuration	duration	Yes
GuidoFindPageAt	pageat	Yes
GuidoGetPageDate	pagedate	Yes
GuidoGetPageMap	pagemap	Yes
GuidoGetSystemMap	systemmap	Yes
GuidoGetStaffMap	staffmap	Yes
GuidoGetVoiceMap	voicemap	Yes
GuidoGetTimeMap	timemap	Yes
GuidoAR2MIDIFile	midi	Yes
GuidoGetVersionStr	version	No
GuidoGetLineSpace	linespace	No

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

To represent this distinction via that GUIDO server, the function is elaborated as a URI segment in the path part of the URI either after the server's URI (for information about GUIDO) or after the SHA-1 tag of a score (for information about a specific score processed by GUIDO). For example,

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

reports the version of both GUIDO and the GUIDO server and thus does not need a SHA-1 tag. On the other hand, the URI

```
curl http://guido.grame.fr:8000/
07a21ccbfe7fe453462fee9a86bc806c8950423f/voicescount
```

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 if it is score-specific or generic to all of GUIDO. 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.

5.0.2. 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.

```
curl http://guido.grame.fr:8000/
07a21ccbfe7fe453462fee9a86bc806c8950423f/staffmap?staff=1
```

Default arguments are provided for all argument-taking functions in case the user fails to specify an argument.

5.0.3. 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.

In order to deal with malformed URIs, the GUIDO server provides defaults for out-of-range or nonsensical values and ignores nonsensical keys.

```
curl http://guido.grame.fr:8000/
07a21ccbfe7fe453462fee9a86bc806c8950423f/?
topmargin=0.0&format=jpg&resize=useless&foo=bar
```

In the above example, the `topmargin` and `format` values in the query part of the URI are used by the server whereas `resize` is given its default value because `useless` is useless and `foo` is ignored because it is not a valid key.

5.0.4. JSON as structs

GUIDO provides several structures in its public API for the transmission of information that falls outside standard C types. For example, the `Date` struct is a numerator-denominator pair used to specify a point in time in a score. The `Time2GraphicMap` struct is a composite structure consisting of pairs of `TimeSegment` and `FloatRect` structures. `TimeSegment` corresponds to beginning and end `Date` instances whereas `FloatRect` shows four points in graphical space.

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. For example, in the JSON returned in Section 5.0.7, `time` corresponds to a `TimeSegment` and `graph` corresponds to a `FloatRect`.

5.0.5. Example: voicescount

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

```
curl http://guido.grame.fr:8000/
07a21ccbfe7fe453462fee9a86bc806c8950423f/voicescount
```

yields the following result:

```
{
  "07a21ccbfe7fe453462fee9a86bc806c8950423f": {
    "voicescount": 1
  }
}
```

5.0.6. *Example: 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:

```
curl http://guido.grame.fr:8000/
07a21ccbfe7fe453462fee9a86bc806c8950423f/pageat?date=1/4
```

yields the following result:

```
{
  "07a21ccbfe7fe453462fee9a86bc806c8950423f": {
    "page": 1,
    "date": "1/4"
  }
}
```

5.0.7. *Example: 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:

```
curl http://guido.grame.fr:8000/
07a21ccbfe7fe453462fee9a86bc806c8950423f/staffmap?staff=1
```

yields the following result:

```
{
  "07a21ccbfe7fe453462fee9a86bc806c8950423f": {
    "staffmap": [
      {
        "graph": {
          "left": 916.18,
          "top": 497.803,
          "right": 1323.23,
          "bottom": 838.64
        },
        "time": {
          "start": "\"0/1\"",
          "end": "\"1/4\""
        }
      },
      {
        "graph": {
          "left": 1323.23,
          "top": 497.803,
          "right": 1730.28,
          "bottom": 838.64
        },
        "time": {
          "start": "\"1/4\"",
          "end": "\"1/2\""
        }
      },
      {
        "graph": {
          "left": 1730.28,
          "top": 497.803,
          "right": 2137.33,
          "bottom": 838.64
        },
        "time": {
          "start": "\"1/2\"",
          "end": "\"3/4\""
        }
      },
      {
        "graph": {
          "left": 2137.33,
          "top": 497.803,
          "right": 2595.51,
          "bottom": 838.64
        },
        "time": {
          "start": "\"3/4\"",
          "end": "\"1/1\""
        }
      }
    ]
  }
}
```

6. CONCLUSION

The GUIDO HTTPD 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 HTTPD server intends to fill this by following RESTful architectural recommendations that have proven successful in other server-based services.

References

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

[2] C. Daudin, Dominique Fober, Stephane Letz, and Yann Orlarey. La librairie guido – une boite à outils pour le rendu de partitions musicales. In ACROE, editor, *Actes des Journées d'Informatique Musicale JIM'09 – Grenoble*, pages 59–64, 2009.

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

[4] H. Hoos, K. Hamel, K. Renz, and J. Kilian. 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, 1998.

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

[6] Fielding R. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[7] L. Richardson and S. Ruby. *RESTful Web Services*. O'Reilly Media, 2008.

[8] M. Solomon. Guido server documentation. `http://www.mikesolomon.org/guido/server/index.html`, January 2014.