A Document Repository for Social Media and Speech Conversations

Adam Funk, Robert Gaizauskas, Benoit Favre
University of Sheffield, University of Sheffield, Aix Marseille Université
Sheffield (UK), Sheffield (UK), Marseille (France)
a.funk@sheffield.ac.uk, r.gaizauskas@sheffield.ac.uk, benoit.favre@lif.univ-mrs.fr

Abstract

We present a successfully implemented document repository REST service for flexible SCRUD (search, create, read, update, delete) storage of social media and speech conversations, using a GATE/TIPSTER-like document object model and providing a query language for document features. This software is currently being used in the SENSEI research project and will be published as open-source software before the project ends. It is, to the best of our knowledge, the first freely available, general purpose data repository to support large-scale multimodal (i.e., speech or text) conversation analytics.

Keywords: Document repository; social media; REST service

1. Introduction

Conversational interaction, including social media streaming and spoken conversation, is a natural paradigm for interactions with customers and other users. In the SENSEI project we are going beyond keyword search and sentence-based analysis of such conversations to provide automatic descriptions and summaries of who said what and of opinions held on relevant subjects. To this end, we need to store structured data for millions of constituents of the conversations, including the original texts and associated metadata as well as annotations generated in the course of syntactic, semantic, and pragmatic analysis, summarization, clustering, etc. We have therefore designed a custom repository for this purpose, which we will publish as open-source software as part of the project's dissemination and exploitation activity. It is, to the best of our knowledge, the first freely available, general purpose data repository to support large-scale multimodal (i.e., speech or text) conversation analytics.

Our conversational repository offers SCRUD (search, create, read, update, delete) functionality (Martin, 1983) over richly annotated documents. It runs as a standard REST service with JSON for data interchange. This offers a number of advantages for easy interaction, testing, and debugging, and JSON avoids the verbosity and high-level parsing hassles associated with XML (e.g., walking a DOM tree or constructing one with SAX); it is also trivially easy to process with Python's standard library and fairly easy to process in Java with the Jackson library (MongoLab, 2011; Saloranta, 2013; Ecma International, 2013).

2. Document model

The main unit of data storage in the repository is a *document* object similar to the GATE or TIPSTER model (Wilks et al., 2000; Cunningham et al., 1997), which allows arbitrary data to be stored as stand-off annotations (grouped in named sets), annotation features, and document-wide features. The document is represented in the repository input and output as a JSON object[1] containing an unique integer `id` assigned by the repository, an optional `name` string, a `content` string representing the plain text content of the document (e.g., of a web page with the HTML tags stripped), a `features` object, and an `annotations` object.

The `features` map contains any data relating to the whole document, such as the source URL, the external document ID, the parent document ID, a link or pointer to an audio file, etc.; the keys are strings and the values can have any JSON type, including nested arrays and objects.

The `annotations` map's keys are strings representing annotation set names (e.g., "`nlp`" for tokenization and POS-tagging), and its values are arrays of annotations; each annotation is itself an object containing a string `type`, `start` and `end` offsets (in characters with respect to the document content), and a `features` map similar to that of the document. Annotations can overlap within the same set and across sets, so the formal representation of this system is an annotation graph (Bird and Liberman, 2001).

Audio documents are a special case and will contain links (in document features) to binary data stored elsewhere, although the document content can contain a transcription. (Annotations' start and end offsets can refer to characters in the transcription or to milliseconds of audio.)

Meta-documents, such as summaries generated by off-line processing, clustering output, etc., can be stored in the repository as additional documents, using content, annotations, and features as appropriate for each case. For example, a multi-document summary or cluster could be stored with the summary text itself in the meta-document's content and an array of document IDs in a document feature to refer to the documents that have been summarized.

An example document is shown in the `result` value of the output in Figure 1 below.

3. REST API

Every HTTP response body consists of a JSON wrapper indicating an error or containing a `result` value, which is a document object, an array of documents, or an array of document IDs. Figures 1 illustrates an HTTP GET (which in this example returns one complete document) the JSON output, and the contained document data structure.

[1] The JSON *object* corresponds to other languages' map, dictionary, or associative array. (Ecma International, 2013)

GET

`http://localhost:8080/repository/document/261169709824069530035677325 97`

⇓


```
{"success":true,
 "code":200,
 "result":{"content":"The cat sat on the mat.",
           "id":1415926,
           "features":{"CRAWLER_externalID":"http://example.com/cat.html",
                       "USFD_GATE_language":"eng",
                       "CRAWLER_langDetected":"eng",
                       "XYZ_MISC_keywords":["cat", "mat"],
                       "CRAWLER_crawled_date":"2014-09-20T13:39:16+0100"},
           "annotations":{"USFD_GATE_NLP":[{"type":"Sentence", "start":0, "end":23,
                                            "features":{}, "id":100},
                                           {"type":"Token", "start":0, "end":3, "id":101,
                                            "features":{"category":"DT", "kind":"word",
                                                        "orth":"upperInitial"}},
                                           {"type":"Token", "start":4, "end":7, "id":102,
                                            "features":{"category":"NN", "kind":"word",
                                                        "orth":"lowercase"}},
                                           {"type":"Token", "start":8, "end":11, "id":103,
                                            "features":{"category":"VBD", "kind":"word",
                                                        "orth":"lowercase"}},
                                           {"type":"Token", "start":12, "end":14, "id":104,
                                            "features":{"category":"IN", "kind":"word",
                                                        "orth":"lowercase"}},
                                           {"type":"Token", "start":15, "end":18, "id":105,
                                            "features":{"category":"DT", "kind":"word",
                                                        "orth":"lowercase"}},
                                           {"type":"Token", "start":19, "end":22, "id":106,
                                            "features":{"category":"NN", "kind":"word",
                                                        "orth":"lowercase"}},
                                           {"type":"Token", "start":22, "end":23, "id":107,
                                            "features":{"category":".",
                                                        "kind":"punctuation"}}],
                            "USFD_GATE_Events":[{"type":"Event", "start":0, "end":23, "id":101,
                                                 "features":{"kind":"sitting",
                                                             "agent":"cat"}}]
                           }
          }
}
```


Figure 1: Fetching a document by ID


```
{"success":true, "code":200, "result":{..document..}}

{"success":true, "code":200, "result":[{..doc0..}, {..doc1..}, ...]}

{"success":true, "code":200, "result":"document content"}

{"success":true, "code":200, "result":[10001, 13500, ...]}

{"success":false, "code":302, "result":"error message"}
```


Figure 2: Examples of output format containing one or more documents, one document's content, a list of document IDs, and an error message

Most operations consume `application/json` (other than GETs, which consume nothing but have parameters in their URLs); all operations produce HTTP responses in `application/json` in the format shown in Figure 2. Endpoints are provided for operations such as the following.

- POST methods are provided for storing new documents from various formats (Websays XML, JSON documents already in the repository format, and plain text), including some simple ones for testing and debugging.
- PUT methods are provided for adding features, annotation sets, and annotations to a specified document. These methods create new document features and annotation sets as necessary; for example, a call to add annotations to a named set that does not exist yet will create that set on the document.
- GET methods are provided to return a complete document, its features alone, or its content alone (as a string), by specifying its document ID.
- A range of GET methods for simple and complex queries on document features are provided, so that document features can be used as flags so that a client can set them as part of its output and then ask for more documents that it has not already processed. The complex queries allow the client to specify conjunctions of features to test, disjunctions of possible feature values, features that must be present, missing, or missing or false (in a sense similar to Python's), the maximum number of documents to return, and whether to return an array of full documents or an array of document IDs. (For large results, it is usually more efficient to get a list of document IDs and then request them individually for processing. This also allows the client to request only the features or content of the documents, as required.)

  The query system uses feature-value pairs and specification-feature pairs of the form `<string>=<string>` joined with the `&` symbol. All the values for one feature are joined with a logical *or* into a subquery, then all the subqueries are joined with a logical *and*. The order of the pairs in the query is unimportant. A pair can also consists of a specification name as follows. Figure 3 shows two examples.

  `_MAX_=<integer>` This sets the limit for the number of documents to be returned. If it is zero or omitted, there is no limit.

  `_PRESENT_=<fname>` This specifies that the named feature must be present for documents to match, although the feature's value can be `null`, `false`, `0`, or an empty string. This can be used to find documents on which prerequisite processing has been done.

  `_MISSING_=<fname>` This specifies that the named feature must be absent for documents to match. This is intended mainly for detecting documents on which a flag has not yet been set.

  `_FALSE_=<fname>` This specifies that the named feature must be missing, `null`, `false`, `0`, or an empty string. Note that `FOO=false` is more restrictive than `_FALSE_=FOO`. This is intended for detecting documents on which a flag either has not been set or has been set so as to indicate that all required processing has not been carried out yet.

  The query methods in effect translate the REST URL query language to the MongoDB database query language. We have not provided queries on document annotations because the SENSEI project does not require it, but they could be implemented similarly (although a more complicated input syntax would need to be developed).

- DELETE methods are provided for deleting a whole document by ID and for deleting specific things—named features document, a named annotation sets, or a specified annotation—from a document specified by ID. (The method that deletes a whole document returns a copy of it, which the client could use to "undo" the deletion.)

4. Implementation and deployment

The repository software has been developed in Java to be used as an Apache Tomcat web service. The document model components are instances of Java classes, mapped to and from JSON using the Jackson (Saloranta, 2013) library, and the documents are stored in a MongoDB (MongoDB Community, 2014; Pollack et al., 2014) server running on a host specified in a configuration file (`localhost` by default). The REST service uses the Apache CXF and Spring libraries (Johnson et al., 2014; Apache Software Foundation, 2014; Balani and Hathi, 2009). We also provide a set of command-line tools in Python for querying, uploading text files, and deleting documents, features, and annotation sets.

MongoDB is itself a "document-oriented database" with its own extensive (and complicated) query language, but our repository software enforces the structure of the document object model and provides easy methods for adding data to existing documents and querying the database by document features.

Because the repository is intended to be hosted as a central service, in a potential multi-party project, all partners' components have to be able to contribute to it. The central repository is run on a Tomcat server which is not accessible from the internet. For access control, ssh accounts and keypairs have been created for each project member using the repository. These accounts cannot run commands but can only create ssh tunnels to the Tomcat port on the prototype machine, so that the clients use URLs such as `http://localhost:8080/repository/` (port-forwarded).

We are in the process of benchmarking the repository's performance (speed and memory) and improving its resilience under heavy loads. The benchmarking evaluation will be

`doc-feature-query/full?&_MISSING_=USFD_NER&_MAX_=5` Return up to 5 full documents whose feature maps do not contain a `USFD_NER` feature at all. For example, these documents need processing by the client tool that will then set the `USFD_NER` flag.
`doc-feature-query/ids?_FALSE_=USFD_Events&NLP_LEVEL=2&NLP_LEVEL=3` Return the IDs of all documents whose feature maps contain either `"NLP_LEVEL":2` or `"NLP_LEVEL":3` but either no `USFD_Events` feature or one that is `null`,`false`, `0`, or an empty string.

Figure 3: Examples of complex document feature queries

published by May 2016 in a project deliverable [2] as well as in the presentation of this paper.

5. Modules

We are developing the following modules that interact successfully with the repository. Other modules in the project will be adapted and wrapped for integration. These components will not be included in the repository software distribution itself but the project's academic partners intend to release them all as open source software by the end of the project.

5.1. Back-end modules

The main source of documents in the project is the output of the Websays[3] crawler, which produces large XML documents, which one of our repository's endpoints is specifically designed to process. Each XML document consists of a large collection of "clippings", e.g., a newspaper article and the readers' comments under it. Each clipping is transformed into one repository document with various document features, including links to the other clippings in the same set.

The following modules add further information to the documents in the repository.

We adapted existing GATE (Cunningham et al., 2011) tools from the ARCOMEM project (Maynard et al., 2014) into a component for carrying out the following tasks for English: standard NLP functions, named-entity recognition, event detection, and sentiment detection. The GATE pipeline is wrapped in a Java component which interacts smoothly with the conversational repository; it polls the repository for batches of unseen documents, processes them, and then sends back annotation sets and document features. The component is highly configurable so it can be used to run other GATE pipelines over repository documents and add to the repository any specified document features and annotations. (The configuration includes options for mapping feature and annotation set names.)

A repository population module has been developed for the MACAON semantic analysis tool chain (Nasr et al., 2011). Thanks to Python bindings, it is possible to interface each stage of the analysis with the repository and update document annotations with those provided by MACAON. Models also had to be trained for Italian while they were already available for French and English.

Integration is currently in progress to allow other tools relevant to the project (including the BART coreferencing system (Broscheit et al., 2010)) to process repository documents and add information back to them. Any NLP or related tools can be integrated with the repository by wrapping them in REST clients.

5.2. Prototype user interface

Our social media prototype demonstrates a front-end application of the repository. An article, its comments, and our summarization outputs are presented to the user as a split web page with the *The Guardian*'s article and comments in the left pane and in the right pane SENSEI's added value, which allows the user to view clusters, cluster labels as a summary, and comments grouped appropriately and in thread context.

The page is dynamically generated on the web server starting from a master document (an example of a meta-document) for a particular summarization of an article's set of comments; the master document's features include the information used to put the rest of the page together, in particular identifiers for other documents retrieved to generate the pie chart with its labels, the summary paragraphs, and the groups of comments. This UI prototype is described in more detail in our separate paper here on the task-based evaluation (Barker et al., 2016).

6. Conclusion

We have presented a novel approach to storing, processing, and using conversational data of various kinds, which has been proven in use with a successful software deployment, with which several working input and output models are already interacting. We are currently adding asynchronous response handling through Spring's ThreadPoolTaskExecutor (Johnson et al., 2014, §33) to the REST service, but otherwise the repository software is complete and we expect it to require no further essential work except for debugging as necessary and handling any additional features that might be needed in the project. Future work may include developing a separate tool for ensuring that inserted documents are consistent with a specified JSON schema; adding endpoints for full JSON path querying using the MongoDB query language (this would also allow querying by annotations); and providing a UI for browsing the repository.

[2] Deliverable D5.3—see `http://www.sensei-conversation.eu/deliverables/`

[3] One of the SENSEI project partners, Websays provides services that "focus on online reputation monitoring and social media marketing".
`https://websays.com/about/`

7. Acknowledgements

This research is supported by the European Union's Seventh Framework Program project SENSEI (FP7-610916).
http://www.sensei-conversation.eu/

8. Bibliographical References

Apache Software Foundation, (2014). *Apache CXF: an Open-Source Services Framework*.

Balani, N. and Hathi, R. (2009). *Apache CXF Web Service Development*. From Technologies to Solutions. Packt Publishing, December.

Barker, E., Funk, A., Paramita, M., Kurtic, E., Aker, A., Foster, J., Hepple, M., and Gaizauskas, R. (2016). What's the issue here?: Task-based evaluation of reader comment summarization systems. In *Language Resources and Evaluation Conference (LREC)*, Portorož, Slovenia, May.

Bird, S. and Liberman, M. (2001). A formal framework for linguistic annotation. *Speech Communication*, 33(1):23–60.

Broscheit, S., Poesio, M., Ponzetto, S. P., Rodriguez, K. J., Romano, L., Uryupina, O., Versley, Y., and Zanoli, R. (2010). BART: A multilingual anaphora resolution system. In *Proceedings of the 5th International Workshop on Semantic Evaluation, ACL 2010*, pages 104–107, Uppsala, Sweden, July.

Cunningham, H., Humphreys, K., and Gaizauskas, R. (1997). GATE—a TIPSTER-based general architecture for text engineering. In *Proceedings of the TIPSTER Text Program (Phase III) 6 Month Workshop*. Morgan Kaufmann.

Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V., Aswani, N., Roberts, I., Gorrell, G., Funk, A., Roberts, A., Damljanovic, D., Heitz, T., Greenwood, M. A., Saggion, H., Petrak, J., Li, Y., and Peters, W. (2011). *Text Processing with GATE (Version 6)*. University of Sheffield.

Ecma International. (2013). The JSON data interchange format. Technical Report ECMA-404, Ecma International, October.

Johnson, R., Hoeller, J., Donald, K., Sampaleanu, C., Harrop, R., Risberg, T., Arendsen, A., Davison, D., Kopylenko, D., Pollack, M., Templier, T., Vervaet, E., Tung, P., Hale, B., Colyer, A., Lewis, J., Leau, C., Fisher, M., Brannen, S., Laddad, R., Poutsma, A., Beams, C., Abedrabbo, T., Clement, A., Syer, D., Gierke, O., Stoyanchev, R., Webb, P., Winch, R., Clozel, B., Nicoll, S., and Deleuze, S., (2014). *Spring Framework Reference Documentation*.

Martin, J. (1983). *Managing the Data-base Environment*. Prentice-Hall.

Maynard, D., Gossen, G., Fisichella, M., and Funk, A. (2014). Should I care about your opinion? detection of opinion interestingness and dynamics in social media. *Journal of Future Internet*.

MongoDB Community, (2014). *The MongoDB 2.6 Manual*. MongoDB, Inc.

MongoLab. (2011). Why is JSON so popular? Developers want out of the syntax business. Technical report, MongoLab.

Nasr, A., Béchet, F., Rey, J. F., Favre, B., and Roux, J. L. (2011). MACAON: an NLP tool suite for processing word lattices. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Systems Demonstrations*.

Pollack, M., Risberg, T., Gierke, O., Leau, C., Brisbin, J., Darimont, T., and Strobl, C., (2014). *Spring Data MongoDB Reference Documentation*. Pivotal Software.

Saloranta, T., (2013). *Jackson JSON Processor Wiki*. FasterXML, LLC.

Wilks, Y., Gaizauskas, R., Humphreys, K., and Cunningham, H. (2000). LaSIE jumps the GATE. Technical report, University of Sheffield.