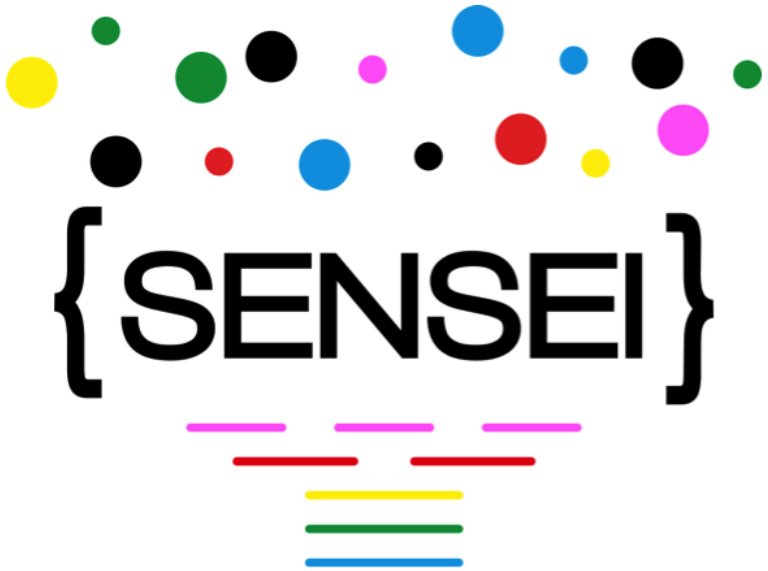


D6.2 – Report on the summarization views of the SENSEI prototype

Document Number	D6.2
Document Title	Report on the summarization views of the SENSEI prototype
Version	1.6
Status	Final
Workpackage	WP6
Deliverable Type	Report
Contractual Date of Delivery	31.10.2015
Actual Date of Delivery	30.10.2015
Responsible Unit	USFD
Keyword List	Summarization views, user interface, prototype, deployment, software licensing
Dissemination level	PU

Editor

Adam Funk (University of Sheffield, USFD)

Contributors

A R Balamurali	(Aix Marseille Université, AMU)
Fabio Celli	(University of Trento, UNITN)
Benoit Favre	(Aix Marseille Université, AMU)
Carmelo Ferrante	(University of Trento, UNITN)
Adam Funk	(University of Sheffield, USFD)
Rob Gaizauskas	(University of Sheffield, USFD)
Vincenzo Lanzolla	(Teleperformance, TP)

SENSEI Coordinator

Prof. Giuseppe Riccardi
Department of Information Engineering and Computer Science
University of Trento, Italy
giuseppe.riccardi@unitn.it

Document change record

Version	Date	Status	Author (Unit)	Description
0.1	2015-07-31	Draft	Rob Gaizauskas (USFD)	Initial Outline
0.2	2015-08-03	Draft	Benoit Favre (AMU)	Rework outline
0.3	2015-08-04	Draft	Rob Gaizauskas (USFD)	Minor mods to outline
0.4	2015-08-24	Draft	A R Balamurali (AMU)	Added section: Conversation as a graph view
0.5	2015-08-25	Draft	Benoit Favre (AMU)	Add prototype table
0.6	2015-09-01	Draft	Adam Funk (USFD)	Add repository updates
0.7	2015-09-02	Draft	Benoit Favre (AMU)	Repository deployment and software distribution
			Fabio Celli (UNITN)	Add Mood view
0.8	2015-09-02	Draft	Fabio Celli (UNITN)	Updated Mood view
0.9	2015-09-28	Draft	Adam Funk (USFD)	Extrinsic evaluation
1.0	2015-10-04	Draft	Vincenzo Lanzolla (TP)	Speech use case views
			Adam Funk (USFD)	Social media use case views; introduction; conclusion; software distribution
			Fabio Celli (UNITN)	Update mood view
1.1	2015-10-04	Draft	Benoit Favre (AMU)	Prototype description
1.2	2015-10-05	Draft	Adam Funk (USFD)	Conclusion
1.3	2015-10-10	Draft	Elisa Chiarani (UNITN)	Quality check completed
			Mijail Kabadjov (UESSEX)	Scientific review completed
1.4	2015-10-19	Draft	Adam Funk (USFD)	Corrections, introduction, executive summary
			Fabio Celli (UNITN)	Introduction
			Vincenzo Lanzolla (TP)	Introduction
1.5	2015-10-20	Final	Adam Funk (USFD)	Corrections
1.6	2015-10-27	Final	Giuseppe Riccardi (UNITN)	Final review

Executive Summary

In this deliverable, we present our progress on summarization views in Period 2. We continued the lines of work pursued during Period 1 of the project on discourse parsing of spoken conversations, on extracting event structure and temporal expressions and on inter- and intra-document coreference in social media, and in addition to these we pursued a new line of work on argumentation structure of conversations planned for the second year.

In this report, Section 2 explains the objectives and broad design of the prototype; Section 3 explains the implementation of the summarization views as well as changes to the repository software and details of the software deployment; Section 4 presents our plans for public distribution of the project's software.

Contents

List of Acronyms and Abbreviations

Acronym	Meaning
ACOF	Agent Conversation Observation Form
BART	Beautiful Anaphora Resolution Toolkit
CLI	command-line interface
CRF	Conditional Random Field (a type of machine learning)
CSS	Cascading Style Sheets
DOM	Document Object Model
FBK	Fondazione Bruno Kessler
GATE	General Architecture for Text Engineering
GPL	GNU Public License
GUI	graphical user interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ISF	Interactive Summarization Framework
JAPE	Java Annotation Patterns Engine
JSON	JavaScript object notation
ML	machine learning
NER	named entity recogntion
NLP	natural language processing
PDTB	Penn Discourse Treebank
PHP	PHP Hypertext Preprocessor
PNG	Portable Network Graphics
POS	part of speech
QA	quality assurance
REST	Representational State Transfer
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
SQL	Structured Query Language
TBD	to be determined
UI	user interface
URL	Uniform Resource Locator
WADL	Web Application Description Language
WEKA	Waikato Environment for Knowledge Analysis
XML	Extensible Markup Language

1 Introduction

This report describes the current state of summarization views—i.e., user interfaces created for evaluation and demonstration purposes—of the SENSEI prototype and the supporting software, summarizes activities in this workpackage during Period 2, and outlines our plans for the workpackage in Period 3.

1.1 Follow-up to Period 1 activities

In Period 2 in this workpackage, USFD principally implemented a social media summarization view user interface, which (as Section 3.3.1 explains) is integrated with the repository, makes use of our summarization work in WP5, and has been successfully used in extrinsic evaluation (D1.3). USFD also made refininements to the conversational repository (Section 3.1) and implemented a basic event- and sentiment-detection tool wrapped in a versatile, configurable Java component for processing repository documents with GATE applications (Section 3.5.1).

In Period 2 in this workpackage, AMU as focused on consolidating the convergence between social media and speech use cases. This entailed the development of view which shows conversations as a graph of interactions complemented by a plugin framework to facilitate integration of other partners modules. AMU also focused on making new algorithms developed in other WPs available in the prototype.

In Period 2 for the speech use case, some improvements to the SENSEI annotation tool and summarization views were considered necessary. Teleperformance, as detailed in Section 3.2, developed a new version of the SENSEI ACOF tool and implemented an instance of Elastic Search and Kibana to support data analysis activities and the prototype Evaluation phases.

In Period 2 for this workpackage, UNITN focused on improving the visualization experience for both social media data and spoken conversational data, and consolidated the the convergence between the tools and views for both conversation types. To this end, UNITN added mood view and template-based summary modules to the visualization of social media conversations (which already included agreement/disagreement and conversational visualizations). Input processing was moved to a REST service on a separate machine, to allow the algorithms to evolve without altering the current UI, which will allow other services and visualization to be integrated easily at any time.

1.2 Follow-up to recommendations from the first review

In accordance with the recommendation made at the review, we intend to publish all the software produced by academic partners with source code. Section 4 and Appendix C provide the full details of these plans.

2 Prototype description

The SENSEI prototype results from the emanation of core work package technology to facilitate the extrinsic evaluation and showcase the results of the project. The objective of the prototype workpackage is to foster and organize all software development in the project in order to make it fit design, quality and distribution guidelines. Another objective is to make software developed for the two use cases of the project (social media and speech) converge towards a common interpretation of what a conversation is and how it should be presented to a user.

The philosophy of the prototype development is to use good development practises without hampering on research. Deliverable D6.1, released in period one, lists the specific guidelines towards this objective. The prototype modules are partitioned in three domains: core offline annotation which is not constrained by time and computation resources and can run on each site's choice of infrastructure, backend modules which run on a server hosted at AMU, and frontend modules which run on the client's web browser. A few mediation modules are set up so that cross-domain communication is possible: a data crawling module is provided by Websays in order to scrape social-media data from the web and store it in xml files; a conversation repository stores conversations enriched by core annotation modules; a web server hosts the prototype UI and backend modules, which dig from the repository.

In phase two, the evaluation workpackage has organized a pilot evaluation scenario for which software was developed, meeting the needs of extrinsic evaluation. The description of the prototype starts with an overview of the modules developed for intrinsic and extrinsic evaluation. This overview is detailed in Table 1 and Table 2, showing that both use cases encompass intrinsic and extrinsic evaluation scenarios. For each evaluation, we have defined a number of tasks, associated with internal ids, metrics and a time span which drives the prototype development. As seen in these tables, a large development effort was spend to meet the summer deadlines.

From this evaluation schedule, we have prioritized the development of prototype modules which were necessary for running the different scenarios. Table 3 and Table 4 give details of the modules according to their internal ids and the different responsibilities in the work

Table 1: Evaluation of prototype for social media

Evaluation	Task	ID	Metric	Start	End
Intrinsic	Article-comment linking	1,2	Link Precision/Pseudo Recall	Oct'14	Apr'15
	Clustering	1,2	soft clustering metrics	May'15	Jun'15
	Cluster labelling	1,2	Amazon Mech.Turk	May'15	Jun'15
	Summarization	1,2	ROUGE	Jun'15	Jun'15
Extrinsic	Information Seeking	1,2	Accuracy on factual questions, Likert scales	Jun'15	Aug'15

Table 2: Prototype evaluation scenarios for speech

Evaluation	Task	ID	Metric	Start	End
Intrinsic	ACOF on 3 questions	1,2,3	ACOF 2,9, 11 are classification task; 9 also includes emotion detection; P/R/F1	May'15	Jul'15
	Complex queries based on synposes	4	Rouge-based metrics	Jul'15	Aug'15
Extrinsic	ACOF on 3 questions	1,2,3	time-to-task comparison; accuracy; agreement among evaluators	Jul'15	Aug'15
Extrinsic	Complex queries based on synposes	4	time-to-task comparison; accuracy; agreement among evaluators	Aug'15	Sep'15

package. The prototype versions are documented according to the languages targeted, the kind of features involved (semantic, parasemantic, discourse), the owner and the contributors.

Table 3: Prototype versions for social media. The second version will be developed during phase 3.

ID	Language	Semantic	Parasemantic	Discourse	Owner	Contributor Features
1	EN	no	no	no	USFD	UI, Summary generation, clustering, labelling
						AMU: UI, topic clustering
2	TBD	yes	yes	yes	USFD	UI, summary generation, clustering, labelling
						UNITN: Agreement/disagreement, moods
						AMU: UI, topic clustering, framenet annotations, sentiment analysis

2.1 Extrinsic evaluation scenarios

Extrinsic evaluation scenarios are fully detailed in D1.3. In the social media use case, the scenario consists in asking journalists to write a town-hall meeting summary from the comments to a news article with the approach developed at USFD. This approach works in three steps: the user first clusters the comments according to their stance, then the clusters need to be labeled with a representative title, and finally, a summary is written from the cluster labels. At each stage of the methodology, the user can be helped by automatic or semi-automatic processes which might replace one or multiple stages.

For the speech use case, two scenarios are evaluated. First quality assurance supervisors from a call center must assess a subset of the agent conversation observation form questions, using predicted values and conversation analysis and summarization tools. Second, QA supervisors are asked to answer complex queries about the content of a conversation collection.

Table 4: Prototype versions for speech.

ID	Language	Semantic	Parasemantic	Discourse	Owner	Contributor Features
1	IT	no	yes	yes	UNITN	Dialog Acts, overlaps, emotion Recognition, lexical, sentiment
2	FR, EN	yes	no	yes	AMU	Baseline template-based synopses, abstractive summarization, framenet parsing
						UNITN: abstractive summarization
3	FR, IT	no	yes	no	AMU	Sentiment analysis, prosody-based classifiers, ACOF classifier
4	FR, IT	no	no	no	TP	Complex queries UI with Elastic Search / Kibana

They achieve this goal through a user interface which provides them with statistics, analyses and search amenities.

Both use case scenarios cover the semantic, parasemantic and discourse core features as well as summarization technology. In addition, care has been take to provide users with well designed UI so that ergonomy does not impact evaluation results.

2.2 Demonstration purposes

In addition to extrinsic evaluation, an important aspect of the prototype is demonstration. More than a communication device towards the outside of the project, this need was evidenced by the difficulty of journalism professionals to describe how natural language processing technology, and in particular summarization technology, could help them in their daily work. The prototype, and in particular its modules related to the social media use case, has the extra requirement that it should be a starting point for discussing with non-scientist professionals which work with conversations as to help them expression their need, and show them what is possible with today technology and what will be at reach with tomorrow technologies.

2.3 Evolution of design elements

Compared to the design elements proposed in D6.1, we had to evolve a few concepts in order to keep up with the project objectives. In particular, we decided to improve the robustness of the repository towards large data streams, and had as well to create lightweight schemas for putting annotations in the repository. An other problem was the user interface flexibility required for extrinsic evaluation with non computer scientist users such as quality assurance supervisors or journalism professionals. The last aspect was one of collaboration between developers where a plugin system was defined so that software developed by one partner could be run by other partners in the context of their UI elements without complex software deployment.

All those aspects are covered in details in subsequent sections.

3 Implementation details

3.1 Updates to repository

The repository specified in D5.1 has been successfully used, but some improvements were considered necessary and carried out as follows.

- The original implementation was unable to process very large (e.g., 700 MB) Websays XML documents without running out of memory. (One Websays XML document contains many crawled items and translates into the same number of SENSEI repository documents.) We changed the internal implementation of the corresponding HTTP POST endpoint to process the XML as a stream rather than a DOM (a document object model, all of which has to be in memory at the same time).
- We added a GET endpoint that supports more flexible queries based on document features, with options to limit the number of results, to return either full documents or document IDs, and to look for the absence or presence of features as well as specific values. This is described in detail in Appendix A. This query system allows the various SENSEI components to interact with each other through the repository with finer control.
- We removed the REST endpoint that allowed a document (specified by ID) to be completely overwritten, because it was considered dangerous—if two components called it on the same document at the same time, the results would be unpredictable. Instead, components must use the correct operations to add and remove annotation sets and document features.

The WADL (Web Application Description Language) for the current version of the repository is included for reference as Appendix B.

3.2 Summarization views: speech use case

3.2.1 Elastic search / Kibana

Elastic Search is a search server based on Lucene, with full-text search engine, REST web interface and schema-free JSON documents, for its native features can be easily integrated with the JSON technology used in SENSEI project. Elastic Search is developed in Java and is released as open source under the terms of the Apache License.

Elastic Search has many plugins that increase capabilities and functions, for our purposes, we installed:

- the Marvel plugin, used for building a GUI to show statistics, information about the server, etc.;
- the Sense query tool, with its capabilities to create, update, delete indexes and documents and searching them.

All annotated data in Periods 1 and 2, previously stored in a MySQL database, have been exported, converted to JSON, then imported into Elastic Search (ES). Before importing data through the Sense plugin, we created an index in ES with a mapping for all necessary data to. For the bulk export from the MySQL database and subsequent import into ES, we developed a PHP module that uses the Elastic Search library for PHP. The machine-generated synopsis and ACOF which we received from other partners have been converted to JSON and imported in Elastic Search, where human- and machine-annotated data can coexist.

Kibana is an open source analytics and visualization platform designed to work with Elasticsearch and adopted for the SENSEI project to analyze data indexed in Elastic Search. Kibana has a web UI user friendly that shows data in a simple way but do not have native function that allow administrator to customize the fields of a report result, so to realize fields customization, we have installed the Kibana development version and developed some custom field formatters. Some custom fields implemented in our Kibana installation are the audio player, that allow to play the audio file of the conversation, and the conversation transcription that open a popup with all conversation transcription. For audio player we put an HTML5 audio player linked to audio file.

Kibana offers many charts and reports that can be created, customized and organized in a dashboard for a better user experience.

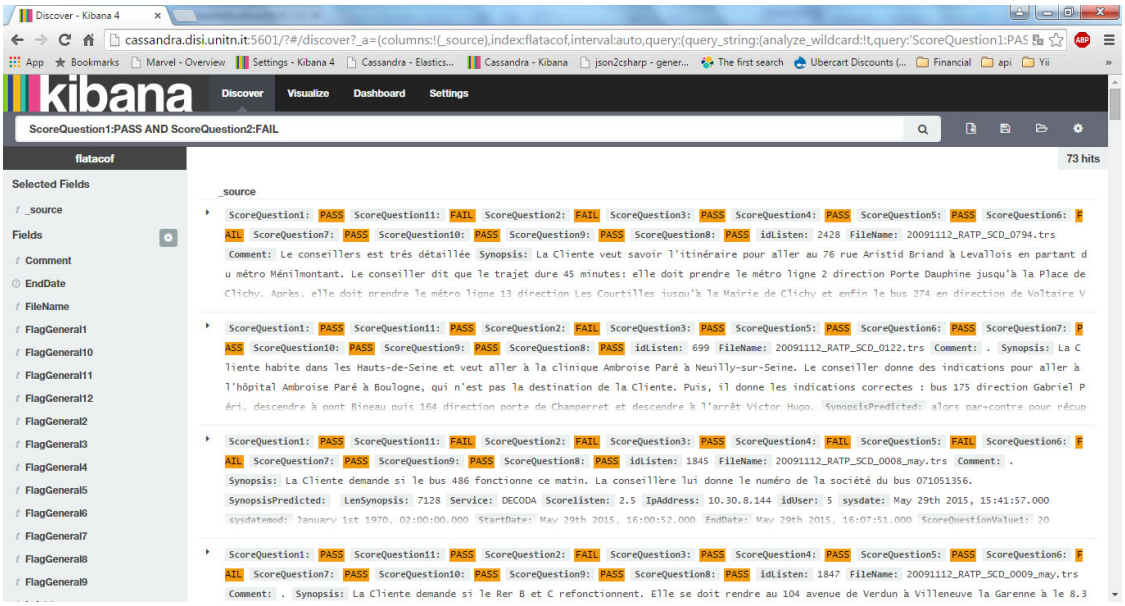
Figures 1 and 2 show examples of the user interface.

3.2.2 Updates to ACOF tool

In Period 1 the SENSEI ACOF Annotation tool, specified in D2.2, has been developed to support Quality Assurance supervisors in the annotation work, providing a user friendly web interface to fill the SENSEI ACOF and the synopsis for each conversation. The annotated data were stored in a MySQL Database. In Y2 the tool has been reviewed as listed below:

- The data layer changed from the initial MySQL Database, where are stored all human annotated data, to a combination of MySQL database and Elastic Search: in the new version when users save the ACOF, data are stored both in MySQL and Elastic Search. We decide to maintain the MySQL database as a backup solution, as soon as we verify that our Elastic Search implementation is robust and reliable, we can decide to quit the MySQL database and avoid redundancy.

Figure 1: Kibana search result



- To read the machine generated synopsis and answers to the questions, a new form “Monitoring prefilled” has been developed. The new view shows at the top of the form, the synopsis predicted by partner systems (pre-loaded into ES), info that can help the QA Supervisor to understand the conversation before starting the listening and evaluation. This form is shown in Figure 3.
- Report are empowered with a better search criteria and filters to refine search, there are more options to what to filter, what to search, and which elements compare (like questions)
- A customizable table of results, that lets the user choose which columns were visualized, very useful to compare questions, scores and other data.
- Excel export of search results

The new version of the SENSEI ACOF tools carry out searches directly in Elastic Search instead of using the MySQL database—this gives greater efficientcy in full-text searches.

Figure 2: Kibana Dashboard

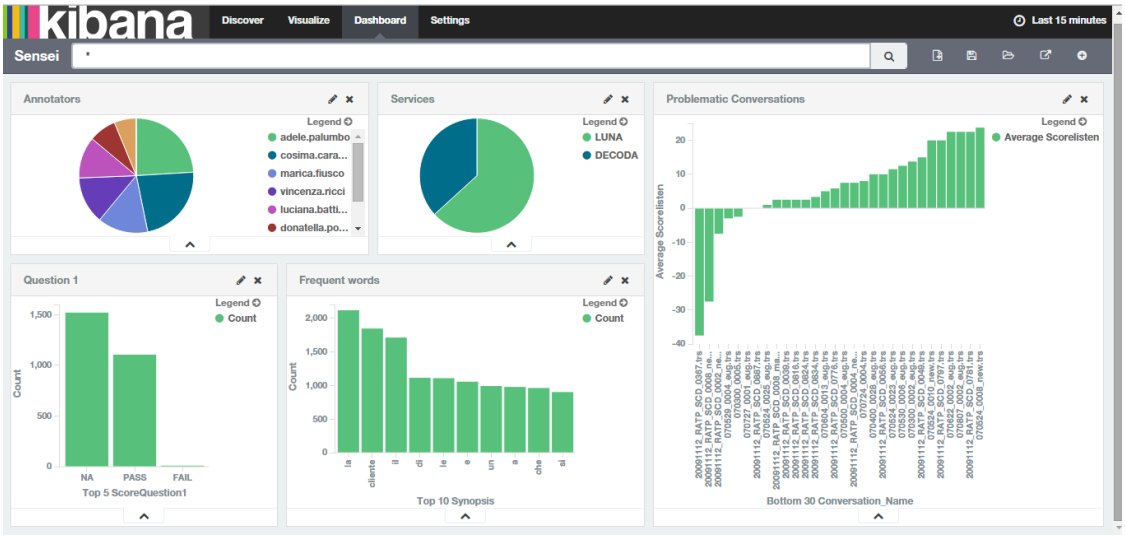
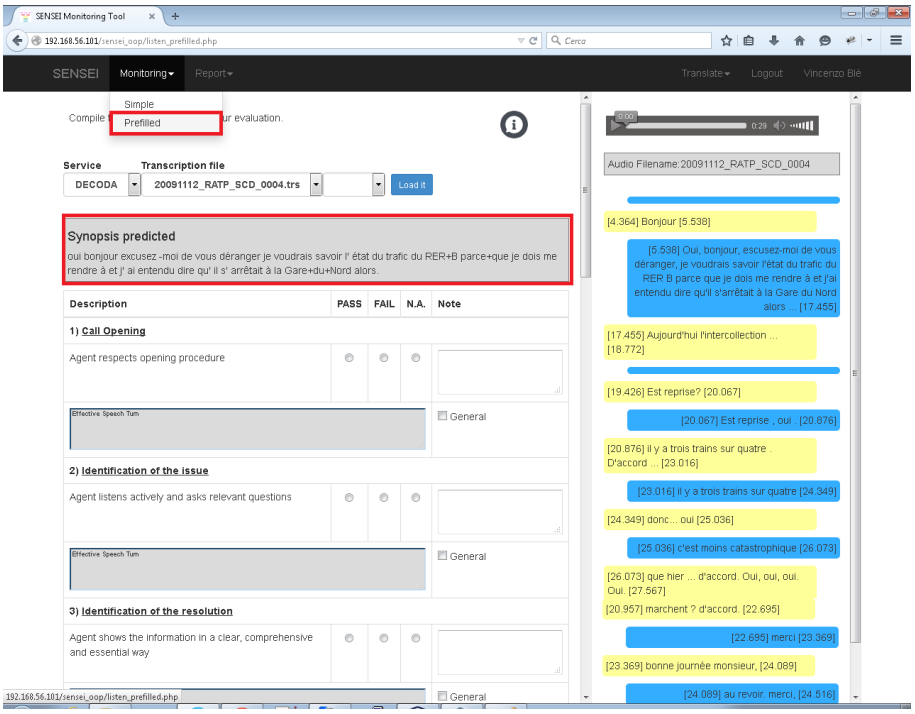


Figure 3: New view monitoring form prefilled

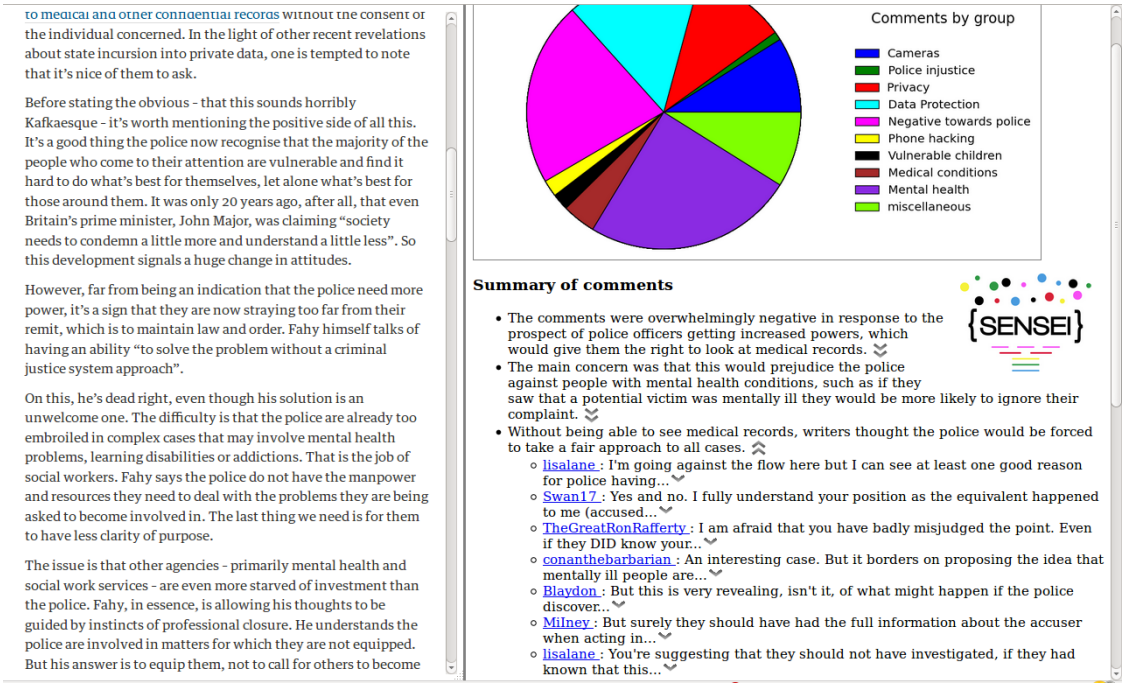


3.3 Summarization views: social media use case

3.3.1 Social Media Prototype 1

The first social media prototype, as used in the extrinsic evaluation described in D1.3, is presented to the user as a web page. As Figure 4 shows, it consists of a split page with *The Guardian*'s article and comments in the left pane and SENSEI's "added value" in the right pane.

Figure 4: Screenshot of the social media prototype.



The page is dynamically generated with PHP[1] from documents in a local SENSEI conversational repository running at USFD (so it can be integrated with the central repository later). The PHP page first reads a master document for a particular summarization (manual or automatic) of an article and a set of comments; the master document's features[2] include the following information, used to put the rest of the HTML page together.

- The URL of the original article is used to embed the article in the left-hand pane.

[1] `https://secure.php.net/manual/en/index.php`
[2] Please refer to Section 3 of D5.1 for a full explanation of the document model used.

- An identifier for the pie chart document is used to embed it in an `<img>` tag using a `data:image/png` URL. The pie chart in the current version is generated in advance from the clusters as a base64-encoded PNG[3] image. In the next version of the UI, the pie chart will be dynamically generated using PHP and JavaScript and will have active features that appear when users hover and click on the wedges.

- A list of identifiers for summary sentence documents is used to retrieve those sentences in order from the repository, and to generate a bullet point for each one (although the comments are hidden when the page is first displayed).

  Each summary sentence document has a feature listing the identifiers of the user comment documents associated with it; this list is used to retrieve the comments and generate a bullet point for one, indented under the summary sentence. Each comment document contains the following pieces of information used by the PHP page:

  – user name;
  – the full comment text;
  – a snippet of the comment text (generated in advance by removing any quoted material in `<blockquote>` tags, and then truncating the rest at the last whitespace before 90 characters; the effect of this is that a snippet can be up to 90 characters long but will never be broken in the middle of a word).

  If the full comment text and the snippet are identical, the whole comment is shown without a button to expand the snippet. If they differ, the snippet is shown along with a downward single chevron that can be clicked to show the full comment instead.

When the page is first displayed, all the comments under the summary sentences are hidden. Clicking the downward double chevron icon displays the list of comments (as snippets, where applicable) and shows an upward double chevron icon, which can be used to hide the list again. All the comments are generated in the page's HTML initially, however—the displaying and hiding of comments and changing of icons are all carried out with JavaScript and CSS, so that once the page is loaded, interaction with it takes place in the user's browser for speed.

D1.3 explains how this UI has been used in extrinsic evaluation; D5.2 explains how the automatic summary are generated.

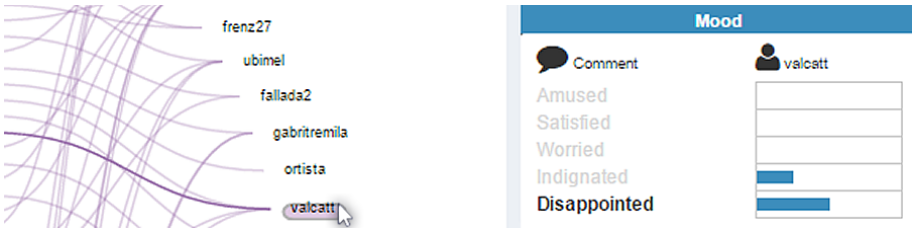
3.3.2 Mood view

The tool for the visual summarization of moods is based on the mood classes defined in the CorEA corpus [1], namely "disappointed", "worried", "indignated", "amused" and "satisfied".

[3] Portable Network Graphics

The classes are defined by the *corriere.it* and are referred to the mood state of the reader after reading an article. In SENSEI we predict two types of mood scores: mood scores associated to article/comments and mood scores associated to readers' profiles (bloggers, author of comments). We visualize both the mood types in a single box associated to the visual structure of the conversation. When the user of the SENSEI technology activate a onMouseOver action on a comment or a blogger's name in the visual conversation summary, the mood of comments and bloggers are displayed in the mood view box, as shown in Figure 5. In order to obtain the

Figure 5: Visualization of moods in the UNITN demo. We display the predicted mood scores in the selected comment and the moods expressed by the author of the selected comment.



visual conversation summary, we modified ConVis [2] *hoque.carenini14*, a visual text analytic system for exploring topics, threads and bloggers in asynchronous blog conversations (a detail is reported on the left in Figure 5). The mood view box consists of 2 columns, on the left we display the mood of the selected comment, on the right the mood expressed by the author of the comment in the whole conversation, including other comments not selected.

3.4 Convergence of summarization views: conversation as a graph view

Summarization systems exist to make the human-human interaction comphrenshible, often acting as agents for information assimilation [3]. However, such systems have underlying assumptions like *limited topic associations*, *coherent dialogue structure* to name a few. These assumptions do not hold true for conversations that happen on current Web2.0 platforms.

Instead of conventional summarization, this problem of conversation comprehension can be seen as an Interactive Summarization problem. Visual representation aided with commonly used NLP/Data mining technologies can help the end user in a better understanding of these conversations. With evolving NLP and visual libraries, it is imperative such a system should be able to associate itself with any future technologies.

Towards this objective, we present the Interactive Summarization Framework (ISF). This frame work is useful in analyzing the newspaper comments. They follow a structure in which there

is an article to which comments are posted by the readers. These comments in-turn can be commented up on. The framework can be applied to any conversation of threaded structure[4] that is predominantly seen on existing social media platforms like Twitter/Facebook.

3.4.1 Framework Description:

The framework consists of two parts:

1. Persistent Modules - These modules forms the back end of the system where the interaction with data happens. The processing of the data also happens at this part.
2. Non-Persistent Modules - These modules take care of representation of the processed data. The developer is provided with freedom to create their implementation of presenting the data.

Persistent modules are so called because we would like to keep some of the data interfaces closed. This would standardize the data ingestion and query process. The users can keep their implementation data on local repostitories or can access a global repository through the public interface given to the non persistant modules.

Framework is developed in such way that system developed can assist an end user in understanding the conversation by using various NLP and data mining algorithm categories. The algorithms are categorized into:

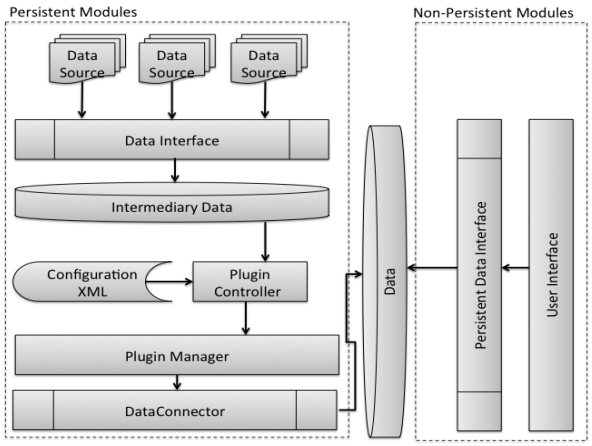
1. **Sentiment Analysis Algorithms**: Detects the sentiment of each conversation [4].
2. **Summarization Algorithms**: Abstractive and extractive summaries generated from main articles or using just the comments [5].
3. **Troll Detection Algorithms**: Removes unwanted conversations which deviate from the topic of conversation [6].
4. **Topic Detection Algorithms**: Topics of conversation are uncovered [7].
5. **Clustering Algorithms**: Clusters the conversation based on the topics and user annotations.
6. **Argument Structure Detection Algorithms**: Detects the nature of conversation with respect to their parent conversation [1].

Developers can implement their algorithms in each of the above categories.They are referred as *algorithm instance*. Different algorithm instances of each category can co-exist. The developer can which algorithm instance to be used for the non-persistance modules.

Figure 6 shows the high level architecture of the framework. A brief description of each module is given in the following sub sections.

[4] reader comments of Guardian/Lemonde.fr etc

Figure 6: A High Level Architecture



3.4.2 Data Interface:

A standard data format for the news comments is followed. The *data interface module* interacts with data sources to collect and transform the data in the required format. At present two data structures are used: one for representing each conversation as a node with its associated properties and another for respresenting the edges which links these conversations (nodes).

3.4.3 Configuration XML:

If the system is being deployed locally, developers can add their algorithm instances. To enable this, a configuration xml is provided. The layout of the xml is given in figure 7. Based on the type and subtype the framework decides which algorithm instances should be applied to the data.

3.4.4 Plugin Controller & Plugin Interface Manager:

Framework enables the developer to add algorithm instances as plugins. Basic framework is developed in python. However, it is platform independed. The developer needs to take care of extending two interfaces in their pythonic plugin modules. The first interface is *executeProcess()* which interacts with their algorithm implementations. Thereafter, appropriate *dbconnect()* interface, depending on the algorithm type, is called. Processed data is communicated between modules as native dictionaries.

Figure 7: A sample configuration xml for summarization algorithm instance

```
<SENSEI>
<ALGORITHM>
<TYPE>summarization</TYPE>
<SUBTYPE>1</SUBTYPE>
<NAME>MMR Summarization</NAME>
<IDENTIFIER>python.package.name</IDENTIFIER>
<DESCRIPTION> This is an implementation of MMR summarization module</
DESCRIPTION>
<MODEL></MODEL>
<STOPWORD></STOPWORD>
<PARAM1></PARAM1>
<PARAM2></PARAM2>
<PARAM3></PARAM3>
<OUTPUT></OUTPUT>
 <REFERENCE></REFERENCE>
</ALGORITHM>
</SENSEI>
```

3.4.5 Persistent Data Interface:

The processed data from algorithm instances are stored in the database. The database can be local or remote. However, interfaces to this database are of fixed format and not available for extension.

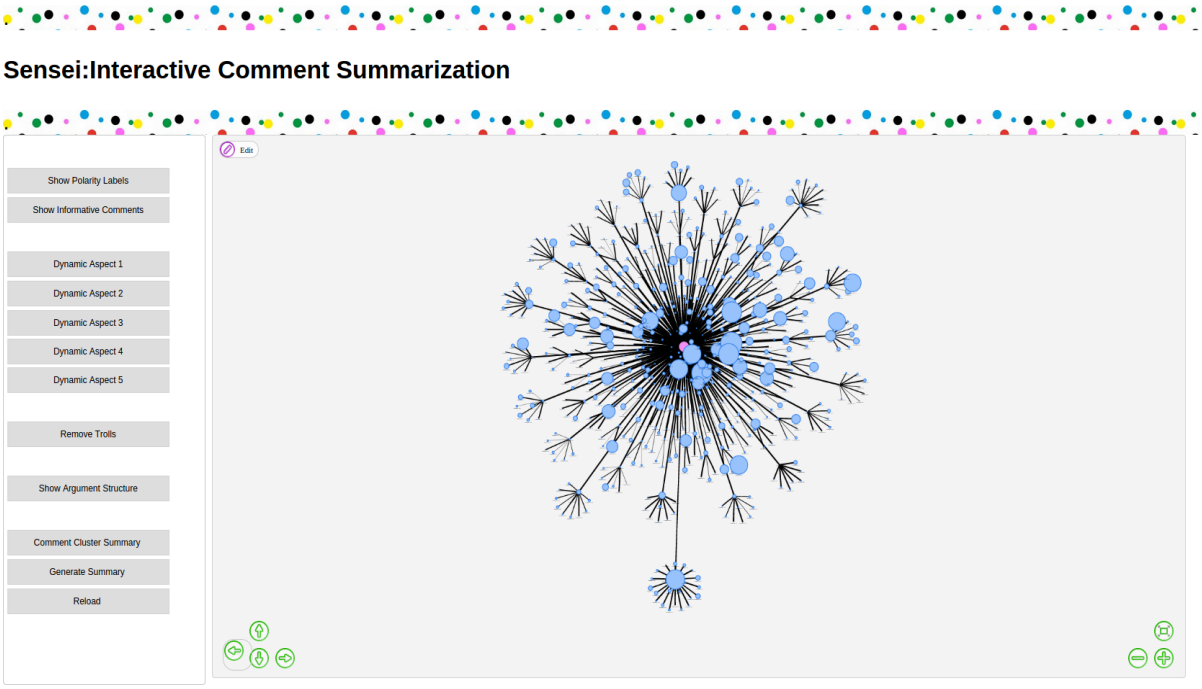
3.4.6 Non-Persistent Interface Modules:

The interfaces to database are provided to nonpersistant modules to read the processed data. However, for including user annotations, a local database is recommended. The developer can extend the interface module to include additional information they would like to capture from the end user.

3.4.7 User Interface:

A web interface/desktop interface can be provided as a non-persistant user interface. Figure 8 shows the current user interface for conversations reprsentation in the graph view. The article is shown in the centre (marked in pink) and the edges represents the comments. This inturn can go in a recursive manner. All the output of the non-persistent module is projected on to this graph.

Figure 8: Graph view of the conversation. The article is in the center with comments around it. The user can interact with each entity to analyze and understand them.



3.4.8 Platform Details:

The framework is build using python 2.7. The backend (database) is supported by MySQL version 14.14.[5] For managing plugins, we use Yapsy[6]. New plugins can be integrated into the system by adding the implementation into plugin folder along with the description of the plugin in the associated *yapsy.plugin* file. The user interface is developed using PHP (version 5.4) with the vis.js (version 3.11)[7] framework for visual representation. The server used for hosting the user interface is Apache 2.4.[8] A demo of the framework implementation can be found at

[5] http://www.mysql.com/
[6] http://yapsy.sourceforge.net/
[7] http://visjs.org/
[8] http://httpd.apache.org/

http://youtu.be/sGyDnDHZDmk.

3.5 Summary of backend module development activity

3.5.1 Event and sentiment detection

We adapted existing GATE [8, 9] tools from the ARCOMEM project[10] into a component for carrying out the following tasks for English:

- standard NLP (e.g., tokenization, POS-tagging, lemmatization) to a high standard;
- named-entity recognition to a high standard;
- event detection at a baseline level on our data (to be tuned for better performance on our data in the future);
- sentiment detection at a baseline level on our data (to be tuned for better performance on our data in the future).

The GATE pipeline is wrapped in a Java component which works well with the conversational repository; this component polls the repository (using the advanced query system described in Section 3.1 and Appendix A) for batches of documents that it has not yet processed, processes them, and then sends annotation sets and document features back to the repository, including a flag document feature used in subsequent queries to distinguish the processed documents.

The Java component is highly configurable so it can be used to run other GATE pipelines over repository documents and send back to the repository any specified document features and annotations.

3.5.2 MACAON semantic analysis

A repository population module was developed for the MACAON semantic analysis toolchain. Thanks to python bindings, it is possible to interface each stages of the analysis with the repository and update document annotations with that provided by MACAON. Models also had to be trained for Italian while they were already available for French and English.

3.6 Deployment

Since Period 1, the prototype has been hosted on a server at AMU. Since the repository is a central service, all partners have to be able to contribute to it. Multiple options have been discussed:

Option 1 Every partner releases software modules which are run on AMU servers in order to feed the repository

Option 2 The repository is made accessible to all partners and they run their modules to feed it remotely

Since option 1 requires a lot of effort and synchronization, it was decided to set option 2 up with appropriate security. Option 1 might be implemented towards the end of the project when the repository-feeding modules are stable. Releasing modules and running them in a centralized way could ensure that they can be released to the public, and therefore benefit the dissemination of the project's results.

The repository is run on the `sensei-proto.lif.univ-mrs.fr` machine as a tomcat service, exposed on port 8080. This port is not directly accessible through the internet due to the repository not implementing access control. Instead, we are using the LIF/AMU ssh gateway as a means to access that host/port. Users have been created on that gateway for each member of the project using the repository. These users, identified through public/private keys, cannot run commands, but have a tunnel automatically created to the `sensei-proto:8080` machine, which can be exploited on the local end of the ssh connection to access the repository securely and rightfully through urls such as `http://localhost:8080/repository/documents`.

Details for obtaining credentials and accessing the repository are available on the sensei-proto wiki[9].

Under that mode of access (ssh tunnel), the repository still has two shortcomings:

- Consistency is not imposed (an other partner could delete the document you are writing);
- Robustness is not provided (system administrators can reboot the machine anytime to perform updates without accounting for in progress transactions).

Users of the repository must pay attention to those shortcomings and implement proper safeguards to ensure the integrity of the data.

[9] `https://gitlab.lif.univ-mrs.fr/benoit.favre/sensei-proto/wikis/RemoteRepositoryAccess`, access restricted to registered users.

4 Software distribution

In response to a recommendation made at the first review, we are exploring the idea of distributing the software created in the course of the SENSEI project as a return to the community. The long term objective is to enable an outsider of the project to gather is own data, annotate it with SENSEI tools, put it in the conversation repository and have it presented in the various user interfaces of the prototype. To attain that objective, we have devised a strategy:

1. Document the available software modules candidate for release
2. Have partners develop, document and release those modules
3. Make a testing scenario in order to validate the set as a whole
4. Find a long-term hosting plan
5. Release the software

In this phase, we have focused on the first step in order to document the state of the software modules pre-existing or developed for SENSEI. For each piece of software, a number of fields have been defined to characterize different aspects relevant to software distribution. The following items are documented:

- Name: name of the component (or placeholder if no name yet)
- Owner: Partner in charge of delivering this component
- Provides: features / UI provided
- Used in: what other components / activities require that component
- URL: url where the software can be downloaded, where the doc is available
- Opensource: whether or not it is opensource
- License: license name (proprietary if it is a specific license)
- Programming languages: main programming languages of the component
- Dependencies: what other software it depends on and their license
- Models can be distributed?: whether or not models can be distributed
- Available models: languages / domains for which models can be distributed

- Repository integration: whether the software has repository integration (or the glue code can be distributed)

Appendix C lists those values for the candidate modules for release.

5 Conclusions and recommendations

The conversational repository has been completed and deployed successfully, and no further work is required on it except for debugging as necessary and handling any additional features that might be needed.

We have provided several different summarization views for the speech and social media use cases. These are currently integrated with the repository and other components to varying degrees, and need to be fully integrated in Period 3, along with other improvements based on the results of the extrinsic evaluations as reported in other current deliverables.

We will also refine the backend modules; in particular, the event and sentiment detection tool needs to be tuned for our data and we will explore temporal information extraction as well.

References

[1] Celli Fabio, Riccardi Giuseppe, and Ghosh Arindam. Corea: Italian news corpus with emotions and agreement. In *Proceedings of CLC-2014*, 2014.

[2] Enamul Hoque, Giuseppe Carenini, and Shafiq Joty. Interactive exploration of asynchronous conversations: Applying a user-centered approach to design a visual text analytic system. In *Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces*, pages 45–52. Association for Computational Linguistics, 2014.

[3] Gabriel Murray and Giuseppe Carenini. Summarizing spoken and written conversations. In *Proceedings of EMNLP-2008*, pages 773–782, 2008.

[4] Aditya Joshi, A. R. Balamurali, Pushpak Bhattacharyya, and Rajat Mohanty. C-feel-it: A sentiment analyzer for micro-blogs. In *Proceedings of ACL-2011*, pages 127–132, 2011.

[5] Trione Jeremy. Méthodes par extraction pour le résumé automatique de conversations parles provenant de centres d'appel. In *Proceedings of RECITAL -2014*, pages 104–109, 2014.

[6] Erik Cambria, Praphul Ch, Avinash Sharma, and Amir Hussain. Do not feel the trolls. In *Proceedings of ISWC-2010*, 2010.

[7] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[8] H. Cunningham, V. Tablan, A. Roberts, and K. Bontcheva. Getting more out of biomedical documents with gate's full lifecycle open source text analytics. *PLoS Comput Biol*, 9(2), 2013.

[9] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Niraj Aswani, Ian Roberts, Genevieve Gorrell, Adam Funk, Angus Roberts, Danica Damljanovic, Thomas Heitz, Mark A. Greenwood, Horacio Saggion, Johann Petrak, Yaoyong Li, and Wim Peters. *Text Processing with GATE (Version 6)*. University of Sheffield, 2011.

[10] Diana Maynard, Gerhard Gossen, Marco Fisichella, and Adam Funk. Should I care about your opinion? detection of opinion interestingness and dynamics in social media. *Journal of Future Internet*, 2014.

A Complex queries by document features

Queries with a limit on the number of documents will return all the matching documents if the number of matches is less than the limit, and calling the same query repeatedly with the same limit (or a smaller one) will generally return the same documents (or a subset) unless some documents have been modified between calls.

- *GET* `doc-feature-query/ids?<query>`
  This returns a list of IDs for documents that match the query, using the system described below.
- *GET* `doc-feature-query/full?<query>`
  This returns a list of full documents that match the query, using the system described below.

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.

Each specification-feature pair, which generates a single subquery, consists of one of the following pseudo-features followed by a feature name or integer.

- `_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.
- `_MISSING_=<fname>` This specifies that the named feature must be absent for documents to match.
- `_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 pseudo-feature is intended for detecting documents on which a flag either has not been set yet or has been set to indicate that the specified processing has not yet been carried out.

Here are some examples:

- `?provenance=plaintext&_MISSING_=USFD_NER&_MAX_=5` Return up to 5 documents whose feature maps contain `"provenance":"plaintext"` but do not contain a `USFD_NER` feature at all.

- `?_FALSE_=USFD_Events&NLP_LEVEL=2&NLP_LEVEL=3` Return 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.

B Repository WADL

```
<?xml version="1.0"?>
<application xmlns="http://wadl.dev.java.net/2009/02"
             xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <grammars/>
  <resources base="http://localhost:8080/repository/">
    <resource path="/">
      <resource path="doc-feature-query/full">
        <method name="GET">
          <doc>Return documents that match the complex feature query</doc>
          <request/>
          <response>
            <representation mediaType="application/json">
              <doc>list of full documents</doc>
            </representation>
          </response>
        </method>
      </resource>
      <resource path="doc-feature-query/ids">
        <method name="GET">
          <doc>Return IDs of documents that match the complex feature query</doc>
          <request/>
          <response>
            <representation mediaType="application/json">
              <doc>list of document IDs</doc>
            </representation>
          </response>
        </method>
      </resource>
      <resource path="document">
        <method name="POST">
          <doc>Store the document (JSON) and return the ID</doc>
          <request>
            <representation mediaType="application/json"/>
          </request>
          <response>
            <representation mediaType="application/json">
              <doc>newly allocated ID</doc>
            </representation>
          </response>
        </method>
      </resource>
      <resource path="document/plaintext">
        <method name="POST">
          <doc>Store the document (plain text) and return the ID</doc>
          <request>
            <representation mediaType="text/plain">
```

```
        <param name="request" style="plain" type="xs:string"/>
      </representation>
    </request>
    <response>
      <representation mediaType="application/json">
        <doc>newly allocated ID</doc>
      </representation>
    </response>
  </method>
</resource>
<resource path="document/websays">
  <method name="POST">
    <doc>Store the document(s) (XML) and return the list of IDs</doc>
    <request>
      <representation mediaType="application/xml"/>
      <representation mediaType="text/xml"/>
    </request>
    <response>
      <representation mediaType="application/json">
        <doc>newly allocated ID</doc>
      </representation>
    </response>
  </method>
</resource>
<resource path="document/{docId}/annotation-set/{name}/{annId}">
  <param name="docId" style="template">
    <doc>document ID</doc>
  </param>
  <param name="name" style="template" type="xs:string">
    <doc>annotation set name</doc>
  </param>
  <param name="annId" style="template">
    <doc>annotation ID</doc>
  </param>
  <method name="DELETE">
    <doc>Delete an annotation from the document</doc>
    <request/>
    <response>
      <representation mediaType="application/json"/>
    </response>
  </method>
</resource>
<resource path="document/{id}">
  <param name="id" style="template"/>
  <method name="DELETE">
    <doc>Delete the specified document</doc>
    <request/>
    <response>
      <representation mediaType="application/json"/>
    </response>
```

```
  </method>
  <method name="GET">
    <doc>Retrieve the specified document</doc>
    <request/>
    <response>
      <representation mediaType="application/json"/>
    </response>
  </method>
</resource>
<resource path="document/{id}/annotation-set/{name}">
  <param name="id" style="template">
    <doc>document ID</doc>
  </param>
  <param name="name" style="template" type="xs:string"/>
  <method name="DELETE">
    <doc>Delete an annotation set from the document</doc>
    <request/>
    <response>
      <representation mediaType="application/json"/>
    </response>
  </method>
  <method name="PUT">
    <doc>Add annotations (create the named set if needed)
         to the document (specified by ID)</doc>
    <request>
      <representation mediaType="application/json"/>
    </request>
    <response>
      <representation mediaType="application/json"/>
    </response>
  </method>
</resource>
<resource path="document/{id}/annotations">
  <param name="id" style="template"/>
  <method name="PUT">
    <doc>Add annotations (creating sets as needed) from JSON
         to the document (specified by ID)</doc>
    <request>
      <representation mediaType="application/json"/>
    </request>
    <response>
      <representation mediaType="application/json"/>
    </response>
  </method>
</resource>
<resource path="document/{id}/feature/{key}">
  <param name="id" style="template">
    <doc>document ID</doc>
  </param>
  <param name="key" style="template" type="xs:string">
```

```
      <doc>feature name</doc>
    </param>
    <method name="DELETE">
      <doc>Delete a feature from the document</doc>
      <request/>
      <response>
        <representation mediaType="application/json"/>
      </response>
    </method>
  </resource>
  <resource path="document/{id}/features">
    <param name="id" style="template">
      <doc>document ID</doc>
    </param>
    <method name="DELETE">
      <doc>Delete features from the document</doc>
      <request>
        <representation mediaType="application/json">
          <doc>list of feature names</doc>
        </representation>
      </request>
      <response>
        <representation mediaType="application/json"/>
      </response>
    </method>
    <method name="GET">
      <doc>Return the complete feature map of the specified (by ID) document</doc>
      <request/>
      <response>
        <representation mediaType="application/json"/>
      </response>
    </method>
    <method name="POST">
      <doc>Return a feature map containing the selected features
           of the specified (by ID) document</doc>
      <request>
        <representation mediaType="application/json"/>
      </request>
      <response>
        <representation mediaType="application/json"/>
      </response>
    </method>
    <method name="PUT">
      <doc>Add features (from JSON) to the document (specified by ID)</doc>
      <request>
        <representation mediaType="application/json"/>
      </request>
      <response>
        <representation mediaType="application/json"/>
      </response>
```

```
  </method>
</resource>
<resource path="documents">
  <method name="GET">
    <doc>Return all document IDs in the repository</doc>
    <response>
      <representation mediaType="application/json"/>
    </response>
  </method>
</resource>
<resource path="documents/false/feature/{key}">
  <param name="key" style="template" type="xs:string"/>
  <method name="GET">
    <doc>Return documents with the specified feature missing,
         null, empty (list, string) or zero</doc>
    <request/>
    <response>
      <representation mediaType="application/json">
        <doc>list of documents</doc>
      </representation>
    </response>
  </method>
</resource>
<resource path="documents/false/feature/{key}/{limit}">
  <param name="key" style="template" type="xs:string"/>
  <param name="limit" style="template" type="xs:int"/>
  <method name="GET">
    <doc>Return documents (up to the limit) with the specified feature
         missing, null, empty (list, string) or zero</doc>
    <request/>
    <response>
      <representation mediaType="application/json">
        <doc>list of documents</doc>
      </representation>
    </response>
  </method>
</resource>
<resource path="documents/features">
  <method name="GET">
    <doc>Return documents with features matching the query (flat features only)</doc>
    <request/>
    <response>
      <representation mediaType="application/json">
        <doc>list of documents</doc>
      </representation>
    </response>
  </method>
</resource>
<resource path="documents/missing/feature/{key}">
  <param name="key" style="template" type="xs:string"/>
```

```
  <method name="GET">
    <doc>Return documents without the specified feature</doc>
    <request/>
    <response>
      <representation mediaType="application/json">
        <doc>list of documents</doc>
      </representation>
    </response>
  </method>
</resource>
<resource path="documents/missing/feature/{key}/{limit}">
  <param name="key" style="template" type="xs:string"/>
  <param name="limit" style="template" type="xs:int"/>
  <method name="GET">
    <doc>Return documents (up to the limit) without the specified feature</doc>
    <request/>
    <response>
      <representation mediaType="application/json">
        <doc>list of documents</doc>
      </representation>
    </response>
  </method>
</resource>
<resource path="documents/present/feature/{key}">
  <param name="key" style="template" type="xs:string"/>
  <method name="GET">
    <doc>Return documents with the specified feature (which might be
         false, zero, or empty)</doc>
    <request/>
    <response>
      <representation mediaType="application/json">
        <doc>list of documents</doc>
      </representation>
    </response>
  </method>
</resource>
<resource path="documents/present/feature/{key}/{limit}">
  <param name="key" style="template" type="xs:string"/>
  <param name="limit" style="template" type="xs:int"/>
  <method name="GET">
    <doc>Return documents (up to the limit) with the specified feature
         (which might be false, zero, or empty)</doc>
    <request/>
    <response>
      <representation mediaType="application/json">
        <doc>list of documents</doc>
      </representation>
    </response>
  </method>
</resource>
```

```
      <resource path="test-doc">
        <method name="GET">
          <doc>Create and store a standard test document</doc>
          <response>
            <representation mediaType="application/json">
              <doc>newly allocated ID</doc>
            </representation>
          </response>
        </method>
        <method name="POST">
          <doc>Create and store a test document</doc>
          <request>
            <representation mediaType="text/plain">
              <param name="request" style="plain" type="xs:string"/>
            </representation>
          </request>
          <response>
            <representation mediaType="application/json">
              <doc>newly allocated ID</doc>
            </representation>
          </response>
        </method>
      </resource>
      <resource path="test-doc/{text}">
        <param name="text" style="template" type="xs:string">
          <doc>text</doc>
        </param>
        <method name="GET">
          <doc>Create and store a test document</doc>
          <request/>
          <response>
            <representation mediaType="application/json">
              <doc>newly allocated ID</doc>
            </representation>
          </response>
        </method>
      </resource>
    </resource>
  </resources>
</application>
```

C Software distribution details

Name	Macaon
Owner	AMU
Provides	tokenization, pos tags, dependency parsing
Used in	Framenet parsing; synopsis generation
URL	`https://gitlab.lif.univ-mrs.fr/benoit.favre/macaon`
Opensource	YES
License	GPL v3
Programming languages	C
Dependencies	glib (GPL), libxml (GPL)
Models can be distributed?	YES
Available models	FR, EN, IT
Repository integration	YES

Name	Framenet parser
Owner	AMU
Provides	semantic frames
Used in	Synopsis generation
URL	not yet published
Opensource	to be determined
License	to be determined
Programming languages	Python
Dependencies	svmlight (GPL), cplex (proprietary)
Models can be distributed?	YES
Available models	FR
Repository integration	YES

Name	Decoda synopsis generation
Owner	AMU
Provides	synopses
Used in	Evaluation
URL	not yet published
Opensource	to be determined
License	to be determined
Programming languages	Python
Dependencies	?
Models can be distributed?	YES
Available models	FR
Repository integration	YES

Name	Tweet sentiment analyser
Owner	AMU
Provides	sentiment polarity
Used in	?
URL	not yet published
Opensource	to be determined
License	to be determined
Programming languages	Ruby
Dependencies	kaldi DNNs (BSD)
Models can be distributed?	to be determined
Available models	FR
Repository integration	NO

Name	BART
Owner	UESSEX
Provides	Multilingual Coreference
Used in	Higher-level semantic processing, summarisation, argument parsing, etc.
URL	`http://www.bart-coref.eu/`
Opensource	YES
License	Apache license (v2.0)
Programming languages	Java
Dependencies	Stanford NER (GPL), WEKA (GPL), TexPro (FBK licence)
Models can be distributed?	YES
Available models	EN,IT,FR
Repository integration	Not yet

Name	Website parsers
Owner	Websays
Provides	parsed data text
Used in	data collection compilation
URL	not yet published
Opensource	to be determined
License	to be determined
Programming languages	Java
Dependencies	?
Models can be distributed?	no models
Available models	no models
Repository integration	?

Name	PDTB Discourse Parser
Owner	UNITN
Provides	Discourse Relations
Used in	?
URL	not yet published
Opensource	to be determined
License	to be determined
Programming languages	PHP
Dependencies	?
Models can be distributed?	?
Available models	EN
Repository integration	json schema only

Name	?
Owner	UNITN
Provides	agree/disagree
Used in	social media prototype v2
URL	not yet published
Opensource	to be determined
License	to be determined
Programming languages	Perl
Dependencies	icsiboost, CRF++
Models can be distributed?	no models
Available models	no models
Repository integration	json schema only

Name	?
Owner	UNITN
Provides	mood
Used in	social media prototype v2
URL	not yet published
Opensource	to be determined
License	to be determined
Programming languages	Java
Dependencies	None
Models can be distributed?	?
Available models	?
Repository integration	json schema only

Name	Repository
Owner	USFD
Provides	Repository
Used in	everything in the long term
URL	not yet published
Opensource	YES
License	to be determined
Programming languages	Java
Dependencies	Jackson; Apache Commons-Lang; Apache CXF; Spring Framework; Spring MongoDB interface; a MongoDB instance running on the server
Models can be distributed?	no models
Available models	no models
Repository integration	YES

Name	Repository tools
Owner	USFD
Provides	CLI tools
Used in	testing and debugging
URL	not yet published
Opensource	YES
License	to be determined
Programming languages	Python
Dependencies	Python Requests `http://docs.python-requests.org/en/latest/`
Models can be distributed?	no models
Available models	no models
Repository integration	YES

Name	Comment linking
Owner	USFD
Provides	Links between comments
Used in	Social media prototype
URL	not yet published
Opensource	YES
License	to be determined
Programming languages	Java
Dependencies	
Models can be distributed?	YES
Available models	EN, FR, IT
Repository integration	partial

Name	Comment clustering
Owner	USFD
Provides	
Used in	Social media prototype
URL	not yet published
Opensource	YES
License	to be determined
Programming languages	Java
Dependencies	
Models can be distributed?	YES
Available models	EN, FR, IT
Repository integration	partial

Name	Comment cluster labelling
Owner	USFD
Provides	Clusters of user comments
Used in	Social media prototype
URL	not yet published
Opensource	YES
License	to be determined
Programming languages	Java
Dependencies	
Models can be distributed?	YES
Available models	EN, FR, IT
Repository integration	partial

Name	Comment summarization
Owner	USFD
Provides	Summaries of groups of user comments
Used in	Social media prototype
URL	not yet published
Opensource	YES
License	to be determined
Programming languages	Java
Dependencies	
Models can be distributed?	YES
Available models	EN, FR, IT
Repository integration	partial

Name	Event/sentiment detection
Owner	USFD
Provides	Standard NLP, named-entity recognition, event detection, sentiment detection
Used in	to be used in summarization and clustering in the future
URL	not yet published
Opensource	YES
License	to be determined
Programming languages	Java, Groovy, JAPE
Dependencies	GATE
Models can be distributed?	YES
Available models	EN
Repository integration	YES

Name	Social media prototype UI for *The Guardian*
Owner	USFD
Provides	GUI
Used in	extrinsic evaluation
URL	not yet published
Opensource	YES
License	to be determined
Programming languages	PHP, JavaScript, CSS
Dependencies	
Models can be distributed?	no models
Available models	no models
Repository integration	YES