European Commission | Horizon 2020 European Union funding for Research & Innovation

opencare

opencare

Deliverable 5.3: Implementation and integration of the semi-automated aid to ethnographic coding prototype into the OpenCare platform

Project Acronym	OPENCARE
Title	Open Participatory Engagement in Collective Awareness for REdesign of Care services
Project Number	688670
Work package	WP5 – Data processing for aggregating collective intelligence processes
Lead Beneficiary	UBx — University of Bordeaux
Editor(s)	Guy Melançon *University of Bordeaux* Jason Vallet *University of Bordeaux*
Reviewer(s)	Guy Melançon *University of Bordeaux* Jason Vallet *University of Bordeaux* Luce Chiodelli *University of Bordeaux*
Dissemination Level	Public
Contractual Delivery Date	01/03/2017
Actual Delivery Date	15/07/2017
Version	1.0
Status	Final

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688670.

Table of content

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688670.

Preliminary remarks

We wish first to comment on the *official* versus *actual* delivery dates of this document. It is the present document that indeed was uploaded on the EC portal while the *real* deliverable is a software prototype that was put into use and accessible on the web, with its code stored in a publicly accessible git archive.

The document was uploaded on the EC portal by mid-July. The software itself has been available *on due time*. Previous versions were actually put in use much earlier than the due date.

This document is an accompanying document to the software. It can be seen more as a milestone style deliverable than a technical document *per se*. All technical documentation, as well as the code, is available on a public git repository; see:

- github.com/opencarecc/graph-ryder-dashboard
- github.com/opencarecc/graph-ryder-api

Reviewers (mid-term review) had denoted previous WP5 deliverables 5.1 and 5.2 as having a "milestone nature". There are two reasons why these documents take such a form, just as does this one.

Firstly, this document is a formal document meant to comply with the promised deliverables. Indeed, it would make no sense to upload the deliverable itself (the entire compressed archive of the code and software resources) on the EC portal.

Also, this document is not used internally to communicate the output of WP5 to other partners. The edgeryders.eu online forum and git repo are the places where all (dev and users) team communication takes place. Technical documentation and code are accessible on the web on the git repository. The git wiki additionally is a natural place where users can post issues and/or features requests.

As a consequence, the delay for uploading this accompanying document does not impact our collective work or action. What counts is that the software and documentation was indeed put into the hands of users soon enough so they could use it for their own work and feed back the technical team with remarks and requests.

Secondly, the impact is minimized since UBx (author of this document) is the only ICT team on the project. So this accompanying document is not a technical document used by any other partner. Its sole use is, in a sense, to report on the project progress to the commission and reviewers.

Finally, since this accompanying document was to be uploaded later than initially planned, we took advantage of our June 2017 consortium meeting so we could reflect on recent interactions with users.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688670.

Introduction

This document is twofold. A first part goes over a series of functionalities supporting user tasks that were identified in Deliverable 5.1: "User tasks and requirements; data abstractions and operations requirements", and more particularly those tasks that pertain to the work of ethnographers.

The second part of the document details the techniques, technologies and tools used to implement and deploy the GraphRyder platform as well as the functionalities offered with the developed solutions. It is worth noting that the portal gathers different views and actually serves both the ethnographers' tasks as well as those of other users (community managers, and wider audience).

Technical documentation is also available on github [1].

Our users have adopted the GraphRyder portal; members of EdgeRyders use it with fresh harvested data on a daily basis.

A series of weekly walkthrough tutorial have been organized and publicized to anyone interested (on edgryders.eu) in order to amplify the use and adoption of GraphRyder.

GraphRyder is now accessible from the URL graphryder.opencare.cc

Supporting tasks pertaining to the work of ethnographers

Ethnographers scan all content posted by users on the edgeryders.eu website order to help community managers, and eventually decision makers, to make sense of the large scale conversation taking place online.

The role of GraphRyder in this setting is to support ethnographers in this task. The edgeryders.eu website already implements an annotating mechanism: ethnographers are granted rights to select portion of text in posts/comments and edit a tag field. Annotations are then stored in the Drupal database; requests to annotate content are performed using the javascript annotator library[2].

GraphRyder thus comes as an aid to help ethnographer gain an overview on their work, and help them reflect on their progress and use of tags to annotate the

[1] See :

- github.com/opencarecc/graph-ryder-dashboard
- github.com/opencarecc/graph-ryder-api

[2] See http://annotatorjs.org

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688670.

conversation.

A series of tasks GraphRyder needed to support had been identified in deliverable D5.1, namely:

Task set #1

- See how tags cover the conversations being examined.
- From a set of tags, find the most important conversations associated with them.
- Find "rich" conversations; find the most "insightful" post, the one engaging the most people.
- Picture the tags most often associated with a person (through the posts/comments they author).
- Provide feedback on the "level of expertise" of users involved in a discussion thread.
- Tags associated to a given user could be pictured using a tag cloud.

Task set #2

- Distinguish "rich" posts or comments, those having a larger number of associated tags, and presumably being longer posts.
- The notion of a "popular" tag (associated with more content) also came up as being of interest.
- The number of persons involved in a post is an interesting statistics.
- Make keyword categories and work on these categories to find conversations and users.

This section goes over views offered by GraphRyder and details how each view support different tasks (from the above task sets).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688670.

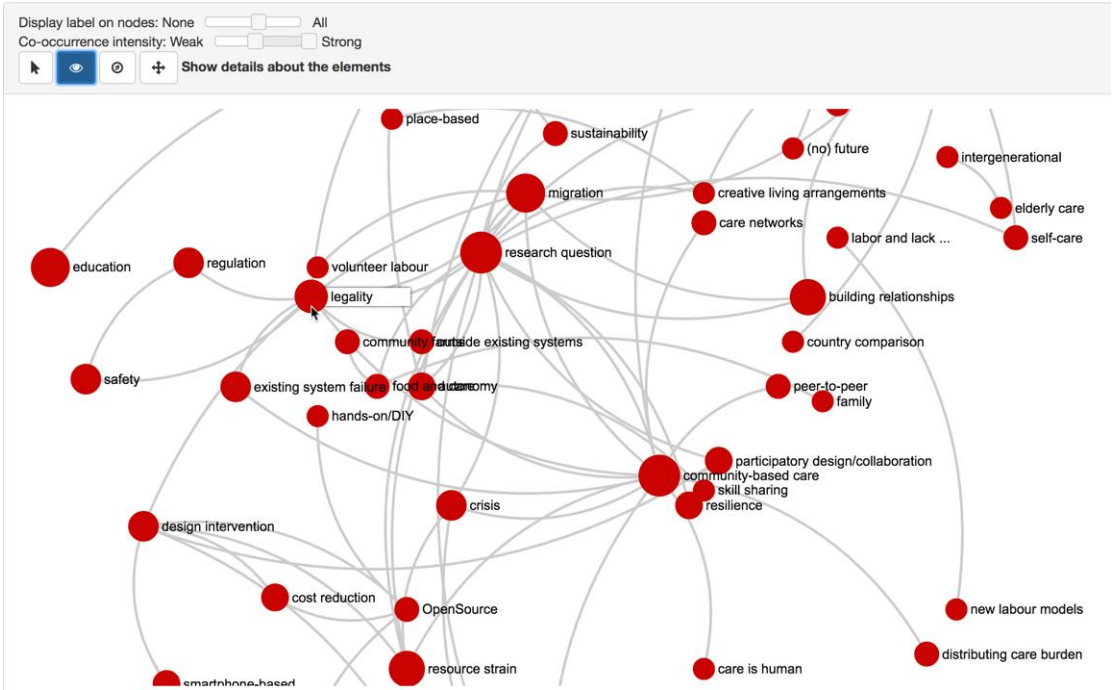

Figure 1. The *tag co-occurrence view*. Tags are connected when co-occurring in a same piece of content. Edges are weighted according to the number of co-occurrences in all ethno-tagged conversations.

The tag co-occurrence network view

Following participatory workshops that were organized early during the project, the *tag co-occurrence view* decidedly became a central object of investigation for all our users. It also turned out to be just as crucial for ethnographers.

The tag co-occurrence network turns out to be quite relevant when looking for the *most relevant conversations or insightful posts/comments* (task set #1). Indeed, the view connects tags appearing in a same content (post/comment) and computes the number of co-occurrences there are in all conversations. Higher co-occurrences values in a sense map to convergent ideas or perspectives; lower co-occurrences usually maps to ideas of a more anecdotal nature.

Users can filter the whole network (605 nodes / 2391 edges as of July 10, 2017) to focus on a smaller part of the network as shown in Figure 1. Tag popularity can readily be accessed since it is mapped to node size (node degree); the larger the node, the more "popular" the tag (see Figure 1); that is, ideas pointed to by these tags have been discussed more thoroughly and/or by more users.

Information on the posts themselves, as well as on users is accessible on demand and appears in a pop-up window (see Figure 2). When clicking on a post, a pop-up window shows the original content and gives access to all other relevant elements: comments/users (authors) to this content, as well as tags annotating the post/comment content. *The number of persons involved in a post* is readily accessible from the pop-up window.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688670.

On such content is more relevant for ethnographers and provides feedback on their own work. Annotations – that is, the text selected by ethnographers when annotating content, can also be visualized.

This detailed view allows ethnographers to review how a tag has been used to annotate a post/comment.

The tag co-occurrence view has been used on several occasions to showcase the opencare methodology to a variety of audiences.

Figure 2. Detailed content shown on demand (post/comment content; tags; users; annotations).

The social semantic network view

A second view builds on top of the co-occurrence network, to which a social network is added. A social network is inferred from the conversation history: users A and B are connected whenever A authors a comment to a content authored by B (see Deliverable 5.1, Figure 3 reproduced here).

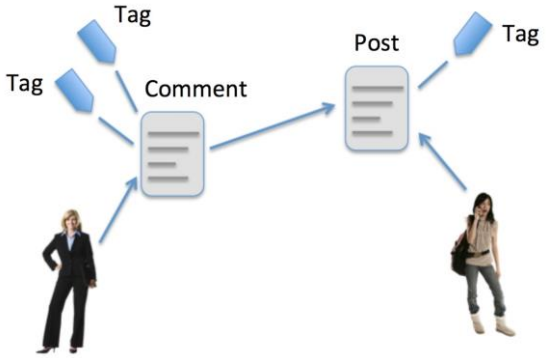

Figure 3. Inferring a social network from authored content. Users A and B are connected whenever A authors a comment to a content authored by B.

The social semantic network view offers a coupled and synchronized view on both the

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688670.

tag co-occurrences and on the social network (Figure 4).

This view can typically be used to detect tags *engaging the most people* (task set #1). Tags can be filtered just as in the tag co-occurrence network, before the lasso is used to select a subset of tags. Brushing around the lasso in the tag view triggers a selection of users who authored content tagged with the selected tags.

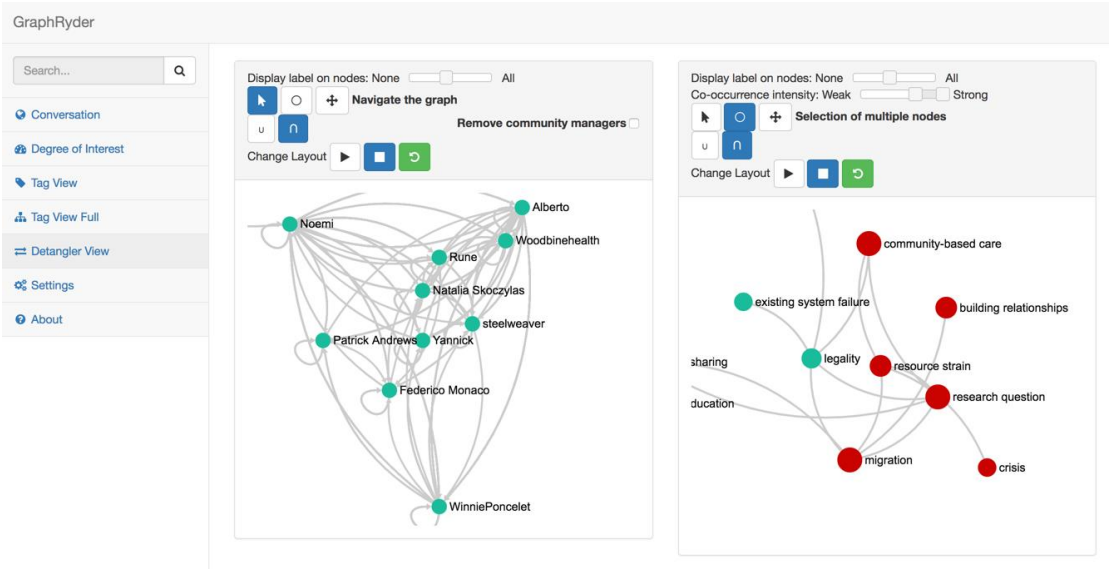

Figure 4. The social semantic network view. People (left) participate to discussions around topics involving codes (right).

It is worth emphasizing that the goal of the social semantic network is not to examine individual users, but to be informed about how conversations develop. Ethnographers may explore the network and realize that non-connected groups of users develop conversations that are similarly tagged. Or conversely, that an isolated subset of tags indeed match an isolated set of users.

Incidentally, this social semantic view is useful to both ethnographers and community managers, for instance.

The technical side of GraphRyder

This section provides an overview of the technical aspects of GraphRyder. It is worth noting that this section applies to GraphRyder as a whole, whether it supports tasks for ethnographers or a wider audience.

This deliverable is related to its predecessor deliverable D5.2 (Toolbox for developing network based software for collective intelligence), which introduced the basics of graph visualization and recalled some task taxonomies for analytical reasoning. Indeed, the technical design of GraphRyder is meant to suit the implementation of

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688670.

the various views and interactions best supporting these analytical tasks.

The platform's source code and technical documentation is available on the github website at:

- github.com/opencarecc/graph-ryder-dashboard
- github.com/opencarecc/graph-ryder-api

Collaborators from Université de Bordeaux have extensive experience in the field of information visualization (and more particularly network visualization) and mre recent experience developing web-based network visualization analysis tool (see Renoust et al. 2015, for instance). Despite this, the construction of an online platform offering a social network analysis tool was a challenge in terms of *accessibility*, *availability* and *openness*.

Expectations

The platform and its deployment have been thought through to address and respect the following characteristics.

Accessibility: Despite the different profiles of the targeted users for the tool, we early established that the platform should be accessible to everyone. We showcased the tool at different gatherings and upon several occasions, to a mixed audience of both experts, enthusiasts, or first-timers (Rome DSI Fair, Milan and Geneva Consortium meetings, Bordeaux hackathon), to favor its adoption. During these live sessions, we listened to the comments and requests of the users to improve the tool and make it accessible to anybody, regardless of their scientific background or lack thereof.

The tool will survive the EC funded opencare project. We plan to keep improving it until the end of the project and after. A public and major release is planned during the #openvillage event to occur next September (in the context of a dedicated workshop session). The GraphRyder platform is seen as one tool contributing to the opencare community dynamics, to help spread the word of existing solutions and shared experiences and make sense of a large scale conversation on these topics.

Availability:

Early in the project, we agreed that the tool should be easily available, anywhere and around-the-clock to agree with the geographical dispersion of the edgeryders. The design of a web application then came as an evidence, to avoid users the burden of a desktop application. The widespread use of web browsers, as well as the latest development on web technologies and more particularly those enabling the display of graphical information were strong arguments favoring such a design choice.

Openness: Based on the same principles as opencare, our tool must be completely opened. The transparency of the technologies used in the platform are meant to

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688670.

allow third parties to observe and react to our work, sometimes sharing their knowledge with us or learning on their own from our work. To this end, all the third parties framework we propose to use in our tool should be open source to avoid black-boxes and opaque solutions and the solution is free, available to anyone, and open for request and feedback from the user through the free code management hosting service GitHub[3].

Description of the solution

To implement our solution, we established a workflow aiming at clearly separating data from operational treatment and presentation/user interface. This strict differentiation allows for quicker and cleaner developments as each "block" is completely independent from the others, and, if need be, can easily be switched with an alternative solution. This possibility makes our solution very modular, allowing such "blocks" to be exchanged with others if a new technology offering better performances were to appear.

We present this workflow in Figure 1. **Users** connect to the framework using a web browser. The server sends content to be displayed, itself assembling content from different web services, some of which rely on data stored in a dedicated database. The returned data is then treated by the web service accordingly to the initial request and the result is transmitted back to the web server for display. This architecture is quite common when proposing web applications and although the database, web service server and web server appear as distinct entities, they can actually be regrouped on the same physical machine hosting the different services at the same time.

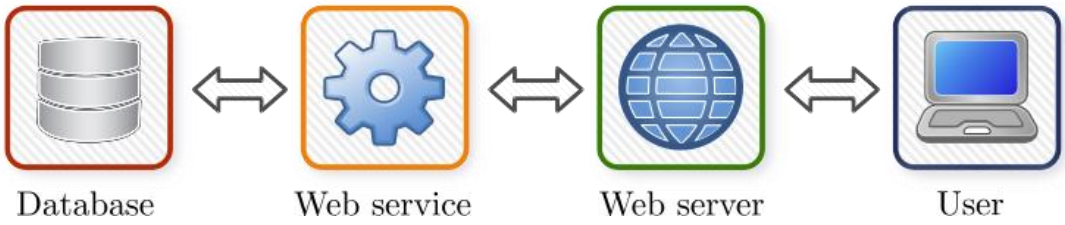

Figure 1. The solution architecture.

Basically, this workflow allows users to have access to the information stored in a database – in our case, public conversations between users, as well as *annotations* and tags entered by ethnographers. The web service can be used to perform additional computations on the data (filtering it, laying it out, specifying a graphical representation of the data, etc.) before returning it to the user.

The client web browser also relies on locally computed procedures to properly render

[3] https://github.com/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688670.

the data sent by the server. A schematic view of the solution architecture is given in Figure 5 below.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688670.

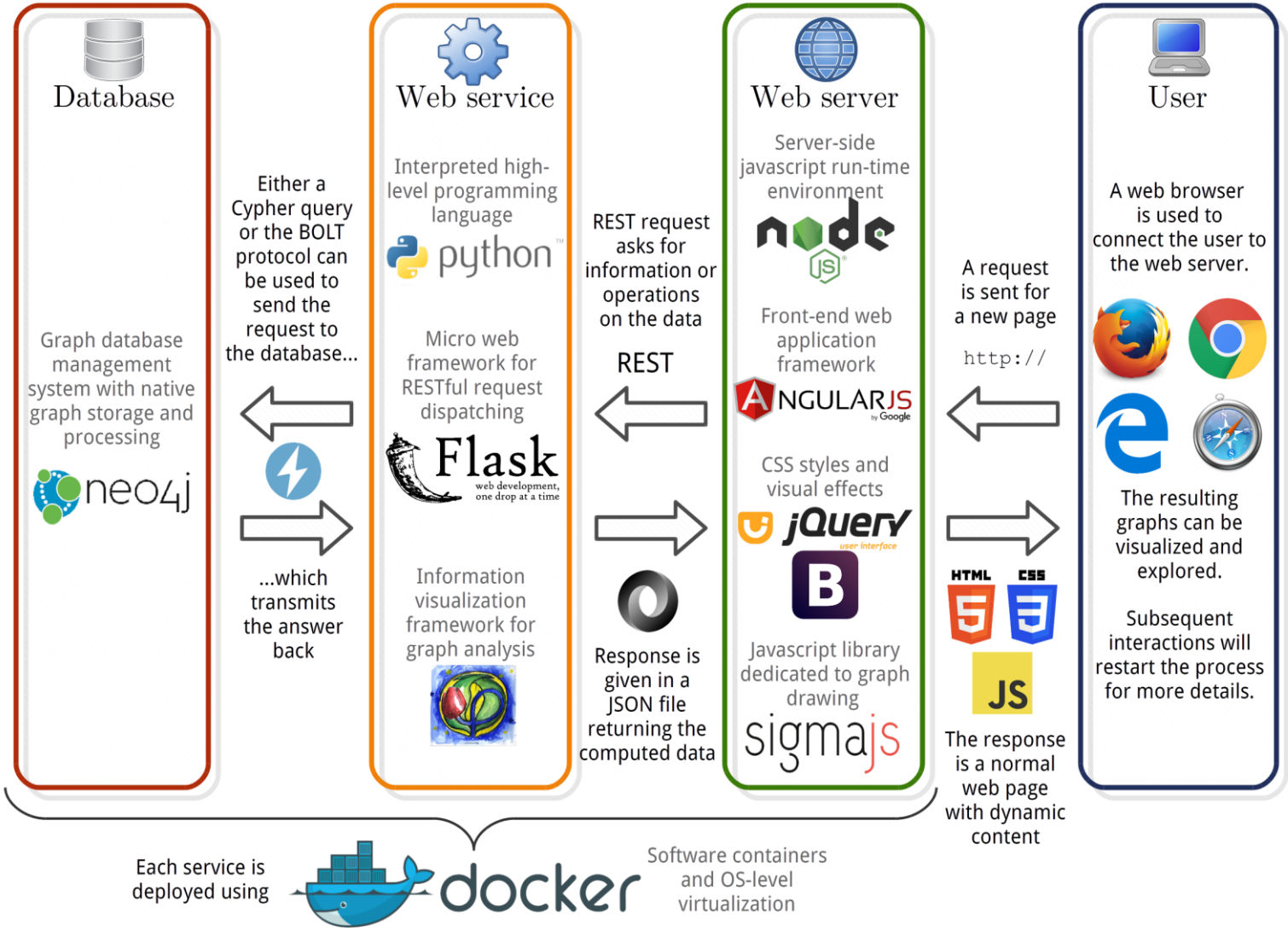

Figure 5. Schematic view of the GraphRyder architecture.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688670.

Database

Several database management systems (DBMS) exist to store and access information. Depending on the kind of data one considers, different database models can be considered. The relational model is currently the most common one and the majority of implementations use the SQL (for Structured Query Language[4]) data definition. However in recent years, another model called NoSQL[5] has started to gain popularity.[6] While some of the most used DBMS are still Oracle DB, MySQL and Microsoft SQL server (all relational DBMS), newcomers like MongoDB or Elasticsearch, which follow the document model [7], have seen their ranking increase significantly throughout the years. Those NoSQL solutions store the information more efficiently and allow for more flexibility due to the underlying data structure used to store information in the database (Sadalage & M. Fowler 2013).

For our particular application case, we have decided to opt for a NoSQL database. Furthermore, while the differences in the system performing the queries can drastically change the efficiency of the DBMS, so can the underlying data structure. Thus, because the information we will work with can be very easily described using graphs, we decided to select a graph database (Ian Robinson *et al.* 2013). Indeed, by using a DBMS preserving the graph structure of the data as we intend to use it, we avoid changing the representation model and thus we do not risk to encounter an object-relational impedance mismatch [8].

Several different graph databases such as Neo4J [9], ArrangoDB [10] or OrientDB [11] propose a native graph structure. In our case, we opted for Neo4j for several reasons. Firstly, while existing as a paying and complete DBMS for enterprises, Neo4j also exists in a community version. The community version has reduced performance and other limitations when compared to its professional version, but it is completely open source, free when used in non-profit applications, and possesses a very large and lively community of users. Secondly, the query language offered by Neo4j, called Cypher, is very well thought and can be used to naturally express the information being queried. Its popularity is such that the project openCypher[12] has been created

[4] https://en.wikipedia.org/wiki/SQL
[5] https://en.wikipedia.org/wiki/NoSQL
[6] https://db-engines.com/en/ranking_trend
[7] https://en.wikipedia.org/wiki/Document-oriented_database
[8] https://en.wikipedia.org/wiki/Object-relational_impedance_mismatch
[9] https://neo4j.com/
[10] https://www.arangodb.com/
[11] http://orientdb.com/
[12] http://www.opencypher.org/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688670.

to bring the language to other graph database. Lastly, drivers allowing to communicate with the database exists in different languages such as JavaScript, C++, Java, Python, etc., offering some additional freedom when deciding upon which technology to use within web service modules.

While each DBMS has strengths and weaknesses, an alternate version of our platform could be deployed using a relational solution. A more conservative PostGreSql[13] database for instance could have offered the same functionalities but with lower performances due to the SQL technology and its inner structure not being well suited to store and query the intrinsic graph structure of the data.

Web service

Web services are used to perform operations on distant machines and get the final result without performing the computation locally. All communications are achieve over a network, such as a local network or Internet, with a peculiar command being first sent to specify which operation to perform, and a response giving the final result asked for.

Although different languages, protocols, frameworks and specifications exist to characterize web services, we have oriented our choice toward a RESTful API [14], receiving requests through specific URLs and returning the computed values structured in a JSON file [15]. For instance, a REST web service could receive a request addressed at [*http://serveraddress/comment/count/since/20161211/0900*] and understand it as an instruction to return the number of comments which have been written since December the 11th at nine o'clock last year. In the same way, replacing *count* with *list* could return the whole list of comment if the web service is correctly configured. The commands sent to the web service can be defined to be very expressive, thus making the calls easy to write and remember. All the treatments become completely hidden from the user and the program or solution interpreting the commands can be changed easily without breaking the complete workflow, provided that each command is also defined on the new service.

Several solutions propose to manage RESTful requests but we opted for the micro web framework Flask [16], programmed in Python 3, to handle the task. Despite its relative simplicity, Flask is very effective. Furthermore, because the tool is based on Python, all the data obtained in return from requests sent to the database can be directly treated using any of the graph analysis solutions available for that language.

[13] https://www.postgresql.org/
[14] https://en.wikipedia.org/wiki/Representational_state_transfer
[15] https://en.wikipedia.org/wiki/JSON
[16] http://flask.pocoo.org/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688670.

We use the Tulip visualization framework [17] as a graph computing engine; Tulip is developed by the visualization research team at the University of Bordeaux. Other graph analysis library such as igraph [18], graph-tool [19], or networkx [20] are good alternative candidates.

Once data processing is achieved, that is the necessary queries for the operation have been sent to the database and all the corresponding responses have been received and integrated to the procedure, then a result is sent to the web server. To keep neutrality between the technologies used in the different modules of our solution, the information returned is structured in a JSON file, a structured text file easy to both generate and parse.

Web server

The web server is the true interface proposing the solution to the user. To make the solutions easily available, we propose a web application compatible with the most common and recent web browsers. Since the late 90's, content of web pages (written in Hypertext Markup Language or HTML) is separated from the visual appearance or styling (using Cascading Style Sheets or CSS). Modern web pages also propose dynamic content nowadays by using JavaScript (or JS). However, these features are handled on the client side and can sometimes greatly strain smaller devices like phones, tablets or notebooks. A few years ago, in 2009, a new JavaScript runtime environment called Node.js [21] was launched which allowed executing JavaScript code on the server side. While this was not the first initiative offering this possibility, Node.js quickly obtained great support from the web development community and was adopted as an efficient system to alleviate computation load on lighter clients.

While Node.js uses JavaScript, alternative solutions for event-driven server frameworks exist like Twisted [22] in Python or Vert.X [23] for Java, JavaScript and Python. However, as we plan to propose a web server, we logically sticked to a web-oriented language (that is, JavaScript) and choose Node.js for its widespread adoption (with online help and documentation being at hand from a quite lively user/developer community.

Above Node.js, we have deployed the front-end web application framework

[17] http://tulip.labri.fr
[18] http://igraph.org/
[19] https://graph-tool.skewed.de/
[20] https://networkx.github.io/
[21] https://nodejs.org/en/
[22] https://twistedmatrix.com/trac/
[23] http://vertx.io/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688670.

AngularJS[24] to structure our website and handle dynamic content and responsive views. Alternatives like Ember.js [25], React.js [26] and Express.js [27] could have offered similar performance but lacked the widespread adoption of AngularJS (and limited available online resources).

Concerning the styling and visual appeal of our solution, we used the Bootstrap framework [28] with the jQuery UI library [29]. Both components are supported by the major web browsers, with Bootstrap allowing us to obtain an identical and consistent website no matter the device used, and jQuery UI proposing a set of graphical user interface widgets to make the navigation smoother.

Finally, we used the library SigmaJS [30] for our visualizations. SigmaJS is a JavaScript library dedicated to graph drawing using node-link representations. This library is one of the central pieces of our solution as it is used to display and communicate in the form of graphs the information to the user and receive feedback through interaction for requesting additional data or details. Very few alternatives exist to this tool currently, the most well known being certainly D3js [31] offering both node-link representations and more general visualizations. A similar in-house solution is actually in development within the visualization research team of the University of Bordeaux. The next version of Tulip [32] is planned to offer a fully functional JavaScript graph visualization library. However, while steadily coming in form, using Tulip in its current state of completion proved to be unwise as several functionalities are lacking still when compared to SigmaJS or even D3js.

Deployment and scalability

The affordability of our solution is as important as its efficiency and its openness. When we had to deploy our solutions, several possibilities were ahead of us. With three different modules, the simple solution would have been to use three different servers. The term server here does not necessary designate a single machine but more likely a node, defined as a virtual machine in a bigger server [33]. This option, while allowing for more bandwidth, thus limiting time delays and allowing more users, and

[24] https://angularjs.org/
[25] https://emberjs.com/
[26] https://facebook.github.io/react/
[27] https://expressjs.com/
[28] https://getbootstrap.com/
[29] https://jqueryui.com/
[30] http://sigmajs.org/
[31] https://d3js.org/
[32] https://github.com/tulip5/tulip
[33] https://en.wikipedia.org/wiki/Virtualization

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688670.

properly separating each module, thus increasing the dedicated computation power for each, would have been quite expensive. Instead, keeping scalability in mind, we decided to try out our solution on a single server and decided to handle the virtualization ourselves using Docker [34], an OS-level virtualization solution offering software containers. This single server solution is cost and energy effective as we only use a single virtual server instead of three and, although the computation resources are shared across the software containers, the web server, web service and graph database are still very responsive.

Should the need arise, as if a lot of users started using the platform at the same time or the database content were to start growing very rapidly, the solution could be moved to a more powerful server with increased processing power, memory and storage. Then, if need be, the separation of each module to different servers would still be possible without completely breaking the actual solution and implementation. Moving each module on their own server would give them a lot of room for processing. Ultimately, if the graph database were to become incredibly large (several billions of nodes and edges), Neo4j proposes a distributed version of its graph database to be deployed on several servers at the same time. Furthermore, the product is available freely for completely open-sourced project under the Affero GPL license.

Conclusion

This document went over the different tasks our web-based application needed to support. It also provided details on the different technologies and tools that were assembled to realize the GraphRyder SSNA dashboard (Figure 5). Inspired from the Detangler application (Renoust et al. 2015), we have built a modular platform application to evolve and to withstand different amount of user load depending of the deployment details.

The GraphRyder architecture has been designed to be as modular as possible, with the possibility to aggregate modules into a single application. Modules correspond to the different views the application offers on the conversation data, and to the modal pop-up window providing details on-demand.

Other modules have been considered but were left as open issues. For instance, at some point, we considered designing views on ethnographic codes letting ethnographers tag content by a simple drag and drop gesture (from the list of codes onto a piece of content). This simple and quite natural design however raised a number of technical difficulties. Indeed, while HTML5 allows designing such interactions, the drag and drop operation turns out to be a challenge, partly because

[34] https://www.docker.com/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688670.

of the frameworks we use (AngularJS / SigmaJS). The design and realization of such functionalities had to be postponed in order to deliver a functional framework early enough in the project. The management of a hierarchy of tags also raised a number of issues, partly because GraphRyder has no control on the storage of annotations (AnnotatorJS together with Drupal).

GraphRyder is published as an open-source application framework from which other applications can be built by assembling modules (those already realized and others from contributors).

GraphRyder is still being improved and will be until the end of the project. Our team is preparing a workshop session part of the #openvillage event (part of the opencare agenda). The workshop goal is to encourage users and disseminate GraphRyder as a companion tool to the edgeryders.eu portal, turn it into a daily tool for users who wish t follow the evolution of the overall opencare large scale conversation.

References

B. Renoust and G. Melançon and T. Munzner (2015). "Detangler: Visual Analytics for Multiplex Networks", Computer Graphics Forum, Volume 34 (2015), Number 3.

P. J. Sadalage and M. Fowler (2013). "NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence", Addison-Wesley.

Ian Robinson, Jim Webber, and Emil Eifrem (2013). "Graph Databases", O'Reilly Media, 2nd edition.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688670.