Fact Checking and Analyzing the Web

François Goasdoué[1] Konstantinos Karanasos[2]* Yannis Katsis[3]
Julien Leblay[1] Ioana Manolescu[1] Stamatis Zampetakis[1]

[1]OAK team, Inria Saclay & LRI
Université Paris-Sud
Orsay, France

[2]IBM Almaden
Research Center
San Jose, CA

[3]UCSD Database group &
WebDam project, Inria Saclay
San Diego, CA

firstname.lastname@inria.fr

ABSTRACT

Fact checking and *data journalism* are currently strong trends. The sheer amount of data at hand makes it difficult even for trained professionals to spot biased, outdated or simply incorrect information. We propose to demonstrate FactMinder, a fact checking and analysis assistance application. SIGMOD attendees will be able to analyze documents using FactMinder and experience how background knowledge and open data repositories help build insightful overviews of current topics.

Categories and Subject Descriptors

H.4.3 [**Communications Applications**]: Information browsers

Keywords

Linked Data, Online Fact Checking, Semantic Annotations

1. INTRODUCTION

Tools for authoring electronic content and sharing it through the Internet are very widely adopted by now. First blogs, and then social networks, grew more or less in parallel with the major media providers' move towards allowing users to record their opinions next to the articles. These technical means to hold back-and-forth conversations, as well as the Linked Open Data movement[1], in which public and private institutions publish their data for transparency and accountability reasons, have sparked new usages of the Web. As another example, public figures rely heavily on social networks to communicate their positions to the public, collect feedback, or survey opinion trends. At the same time, individuals interested in a particular topic or action (e.g., the impact of tax policies on exports, or the impact of traffic on asthma cases) can comb the Web for bits of information, connect, interpret, annotate and re-share them. Such data gathering and fact checking have come at the core of "data journalism", pioneered, e.g., in Europe by The Guardian[2] and growing through efforts such as FactCheck[3], Politifact[4], and similar French sites[5].

Fact checking and analysis (*FCA*, for short), viewed as the process of analyzing a piece of information, crossing it with existing knowledge, verifying its accuracy and possibly enriching it with nuances, comments and connections to reputable sources, has an inherent part of human effort, thus it is unlikely to ever be completely automatized. This is because FCA requires not only *modus ponens*-style manipulation of facts (i.e., automated reasoning, for which mature software tools exist by now), but also involves information extraction tasks at which humans are still better than software, as well as judging (through the context of one's experience) the cultural environment of the target audience of the analysis result.

Nevertheless, many computerized tools for natural language processing, information extraction, and data storage, indexing, querying and visualization can be exploited to facilitate FCA. Such tools are today mostly used by IT specialists, yet the avidity of the public at large for reading, analyzing, commenting, and crossing information raises the need for integrated, generic and open tools. Wide-audience FCA tools must be *integrated* to spare the user the effort of gluing several software components. They need to be *generic* to handle many types of input. For instance, if one wants to analyze some tweet stream, one very likely needs to archive and analyze Web pages, RSS feeds or domain ontologies. Finally, FCA platforms need to be *open* w.r.t. the supported data formats, so that they are not only information sinks but also information sources, and w.r.t. the architecture, so that they can be easily customized and extended.

We propose to demonstrate FactMinder, a customizable FCA assistant based on (*i*) existing technologies for information extraction, (*ii*) the W3C standards XML for representing structured Web documents, RDF for encoding facts and more generally Semantic Web data, and RDF Schema for encoding knowledge (i.e., ontologies), and (*iii*) off-the-shelf XML and RDF content management tools. At the core of

*This work started while the author was at Inria Saclay.
[1]http://linkeddata.org

[2]http://guardian.co.uk/data
[3]http://www.factcheck.org
[4]http://www.politifact.org
[5]http://www.liberation.fr/desintox, http://decodeurs.blog.lemonde.fr

FactMinder lies XR [1], a data model combining XML and RDF under the single paradigm of *annotated documents*, and XRQ, its associated query language. XRQ is used to define *XRQ views* that are the basic building blocks of our FCA application.

The demonstration will enable users to build and enrich a topic-centric FCA repository, through the following steps:

- load documents they want to analyze, and visualize the annotations/connections that FactMinder may have stored about these documents in the past;
- perform simple information extraction (e.g., named entity recognition, identifying people and places) on these documents using automated tools;
- manually enhance the information extraction based on a structured knowledge base (domain ontology) stored within FactMinder, defining new concepts and/or using free-text comments;
- semi-automatically check the consistency of facts from the documents, against FactMinder's repository of documents and facts;
- record in FactMinder's repository the program- and user-generated annotations about the analyzed documents, while keeping them intact;
- visualize and/or share with others (by publishing through a FactMinder-provided URI) the results of the analysis, consisting of documents and annotations.

Usability in FCA translates in the ability and ease to find and inspect relevant information, to make informed judgments. To support this, we rely on expressive XRQ *views* over joint document and semantic corpora, as we explain later on.

The rest of the paper is organized as follows. In Section 2, we briefly present the FactMinder data model (XR) and query language (XRQ) for representing and handling annotated documents. In Section 3, we give an overview of our system by describing its architecture and GUI. In Section 4, we describe the FactMinder demonstration. In particular, we discuss the tasks users will be able to experiment in a typical data journalism-inspired FCA scenario. We then present some related works and conclude.

2. BACKGROUND: THE XR DATA MODEL AND QUERY LANGUAGE

A crucial feature of FactMinder is its ability to annotate semi-structured documents with semantic information. Hence, it requires a data model allowing the representation of such data, and a query language to extract information from it, based on both document structure and annotation semantics. To this end, FactMinder leverages the XR data model for representing annotated documents and the corresponding XRQ query language. We briefly describe XR and XRQ before explaining how they function within FactMinder. For more details, the reader is referred to [1].

XR allows the representation of semantically annotated semi-structured documents by combining in a single model two standard W3C data models: the XML model for semi-structured documents and the RDF model for semantic information. In a nutshell, an *XR instance* is a pair (X, R) of two sub-instances X and R, where X is an XML instance and R is an RDF instance (or graph), respectively, as per the corresponding W3C specifications. An XML document can be viewed as a named, labeled, ordered, unranked *tree*, while an RDF graph is a set of (s, p, o) *triples*, where each triple states that the *property* p of *resource* (or *subject*) s has *value* o. Furthermore, the RDF specification enables declaring *classes*, *properties*, and semantic relationships (i.e., constraints) between them, e.g., Senator is a subclass of Politician.

The main feature of XR is that it allows the RDF annotations to refer to XML nodes. This is accomplished by assigning to *each XML node* a *Unique Resource Identifier (URI)*. This URI of an XML node (called XURI), can then be referenced by an RDF triple, thus enabling the interconnection between the XML and the RDF sub-instance. XURIs can appear in the RDF sub-instance R in any place where a URI is allowed (which is, in the s, p, or o position of any triple). For instance, an XML node can be treated as the *subject (s)* in order to state that a document fragment is relevant to some concept or entity.

To query documents, FactMinder employs the *XRQ query language*, combining (subsets of) the standard W3C query languages for XML and RDF, namely, XQuery and SPARQL, respectively. To query XML, it supports a conjunctive subset including (possibly nested) FLWR expressions. To query RDF, it relies on conjunctive *B*asic *G*raph *P*atterns (or BGPs, in short), a useful core fragment of SPARQL. Importantly, XR queries allow expressing not only disjoint queries over each of the sub-instances X and R of an XR instance, but also queries that *join across the sub-instances*, as illustrated by the sample query below:

q_X	for $p in doc($d_1$)/blog//post[date=doc($d_2$)//tweet/date], $an in $p/author/name, $pu in XURI($p) return $p/title, $an, $pu
q_R	($x, rdf:Type, Senator), ($x, hasName, $an), ($x, fullName, $f), ($x, inParty, $z)
r_X	⟨quote author={$f}⟩{$p/date}, {$p/title}⟨/quote⟩
r_R	($pu, postedBy, $f), ($pu, postFromParty, $z)

In this query, q_X, respectively, q_R are the XML and RDF sub-queries, whereas r_X and r_R build the query result as a new XR instance, consisting of an XML element and two RDF triples. The query searches for an XML blog post whose author is declared to be a Senator, affiliated to party $z, in the RDF database. The query returns: an XML snippet with the Senator's full name (coming from the RDF database) and her blog quote, and two RDF triples connecting the post to the senator's full name and political affiliation. The query illustrates XRQ's ability to join data from the XML and RDF sub-instances (on $an), and output in XML data coming from RDF (the value of $f) and vice-versa (the URI $pu of the blog post in the result triples).

In accordance with the RDF semantics [2], an XR query takes into account all the triples that can be *entailed* from the data stored in the RDF instance. For example, if the resource BillClinton is described as a Politician and Politician is a subclass of Person, BillClinton should be in the result of any query looking for resources of type Person. As we explain in Section 3.2, this can be used to tailor the data shown to the user *in connection with the document under analysis*, to the precise concepts found within that document.

3. SYSTEM OVERVIEW

We present the FactMinder architecture (Section 3.1) before describing the user-facing modules (Section 3.2).

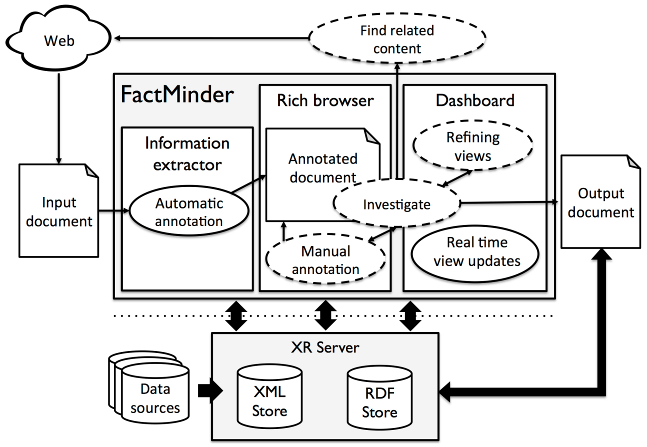


Figure 1: FactMinder architecture.

3.1 Architecture

FactMinder follows a client-server architecture, detailed in Figure 1. In this Figure, boxes depict the FactMinder modules. Solid ovals represent the tasks performed by the application automatically, whereas dashed ovals are the tasks performed by users inside and outside the system.

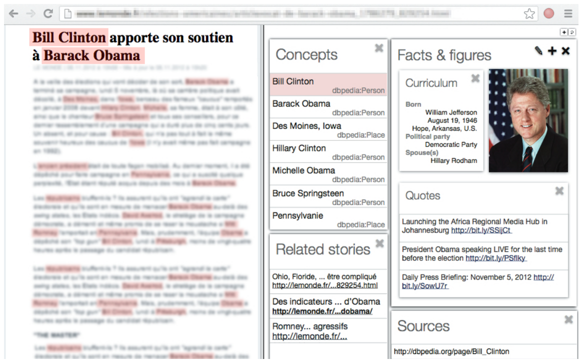
The FactMinder client is made of three components, an *information extractor*, a *rich browser* and a *dashboard*, illustrated by the screenshot in Figure 2, each playing a different role in the fact checking process. When a document is opened within the client, the information extraction module (which relies on OpenCalais [3]) automatically finds the topics, entities and relationships it contains. Documents are opened in the rich browser, where the user can manually add or edit annotations. The dashboard is made of views over the XR database with the background information, each rendering a specific aspect of the information at hand. These views are easily customizable; the user iteratively refines them through the GUI when performing an analysis task. To pursue her investigation, the user switches back and forth between the rich browser and the dashboard. Any insight she gets from the interface may lead her to add some detail to the document as a new annotation, or to refine a view and get a better understanding of the data. FactMinder uses an XR server to integrate XML and RDF from the Web, and store the annotations created both automatically and manually. The server runs on top of BaseX 7.3 [4] for managing XML data, and Virtuoso 6.1.6 [5] for the RDF data. Bold arrows in Figure 1 denote data flows between each component of the system. Content created during the investigation is stored in the database for future use. The analysis may lead the user to find related contents, run additional documents through the same analytical process etc.

Next, we detail how users interact with the application.

3.2 User Interface

The screenshot in Figure 2 exemplifies the main UI components assuming a scenario where a journalist checks some facts from a French online newspaper, about Bill Clinton's support for Barack Obama, during the 2012 US election campaign.

The rich browser. In this area, users can open documents by entering their location, either on the local machine or on the Web. Unlike conventional browsers, it provides a rich



Bill Clinton apporte son soutien à Barack Obama

Concepts

Bill Clinton dbpedia:Person
Barack Obama dbpedia:Person
Des Moines, Iowa dbpedia:Place
Hillary Clinton dbpedia:Person
Michelle Obama dbpedia:Person
Bruce Springsteen dbpedia:Person
Pennsylvanie dbpedia:Place

Related stories

Ohio, Floride, ... être compliqué http://lemonde.fr/...829254.html
Des indicateurs ... d'Obama http://lemonde.fr/...dobama/
Romney... agressifs http://lemonde.fr/...

Facts & figures

Curriculum

Born William Jefferson August 19, 1946 Hope, Arkansas, U.S.
Political party Democratic Party
Spouse(s) Hillary Rodham

Quotes

Launching the Africa Regional Media Hub in Johannesburg http://bit.ly/SSijCt
President Obama speaking LIVE for the last time before the election http://bit.ly/PSflky
Daily Press Briefing: November 5, 2012 http://bit.ly/SowU7r

Sources

http://dbpedia.org/page/Bill_Clinton

Figure 2: Main FactMinder window.

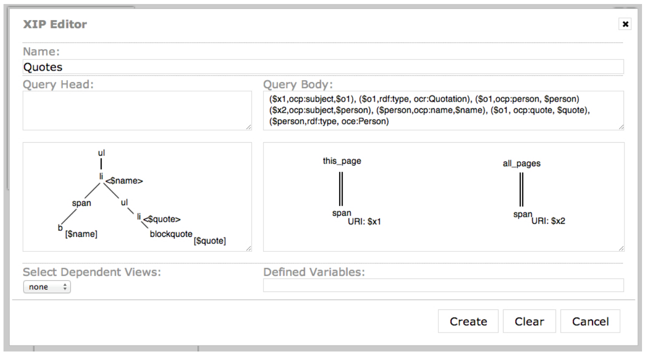
set of annotation tools. First, the annotations produced by the information extractor upon opening the document are accessible to the user by hovering over the text. Second, the user can add her own annotations in a faceted editor by selecting some content, specifying comments and new knowledge, and enriching or correcting existing annotations. Comments contain, e.g., a text body, creation date, author and category, such as "FalseClaim", "uri_1 confirms this", etc.

The dashboard. The dashboard area is composed of XR info panels (XIPs, for short). These panels assist the user in understanding the content she is working on; each panel is dedicated to one aspect of the information under scrutiny. The information content of an XIP is gathered through an *XRQ parameterized view* over the data in the browsing panel and/or the background information (the pre-existing XR database). Semantic connections may exist among XIPs used simultaneously (much in the way the content in part of a Web page changes according to the user interaction with the rest of the page). For instance, the default XIP, shown in Figure 2, comprises a "Concepts" XIP, at the center top of the Figure, featuring all the unique concepts appearing in the currently selected document. Selecting an item in the list, here "Bill Clinton", causes all the occurrences of the item to be highlighted in the document. The "Related stories" XIP, located directly below, gathers all documents that were opened by the user or member of her group, containing the same concepts as those of the current document.

XIPs are highly customizable, and users can add, delete, and re-arrange them at any time. Users define new XIPs by opening a "new panel" editor (detailed below), providing the XIP name and the associated XR query.

Dependent XIPs. By default, an XIP is refreshed automatically when the user opens or selects a new document. Moreover, the recomputation of an XIP can be governed by the user interaction with another XIP; in this case, we call the former *dependent*, and the latter *the parent*. For instance, in Figure 2, the "Facts & figures" XIP depends on the "Concepts" XIP. Whenever an item is selected by the user in the parent "Concept" panel, facts and figures about the chosen concept are displayed in the dependent panel.

Editing XIPs. To create or edit an XIP, the user relies on a graphical editor, such as the one illustrated in Figure 3. An XIP editor opens up, for building XR queries in a graphical form (by editing tree and triple queries). Auto-completion, based on the RDF vocabularies and the datasets, helps the user type-in long URIs and common sentences. To create dependent XIPs, the user simply needs to designate an XIP among the list of existing ones at the bottom of the editor.


XIP Editor

Name: Quotes

Query Head:

Query Body: ($x1,ocp:subject,$o1), ($o1,rdf:type, ocr:Quotation), ($o1,ocp:person, $person) ($x2,ocp:subject,$person), ($person,ocp:name,$name), ($o1, ocp:quote, $quote), ($person,rdf:type, oce:Person)

Select Dependent Views: none

Defined Variables:

Figure 3: View editor.

The user is then presented with the list of the returned variables of the parent XIP which can be re-used to establish connections among dependent views.

Semantic template XIPs. One can easily write concept-specific XIPs. For instance, the default facts for a Company may include the net result in 2012, whereas for a Person, the relevant facts may be the age, nationality and profession. XR's support for RDFS semantics (e.g., subclass relationships) allows one to adapt queries to the context. Suppose that XIP "Concept" of Figure 2 has the following query definition:

q_R^1	($x, rdf:type, owl:Thing), ($x, rdf:label, $label)

q_X^1	for $p in $html//p, $x in XURI($p)
	 return $p, $x

r_X^1	⟨div class="list-item"⟩$label⟨/div⟩

Next, one can define an XIP called "Bio" depending on "Concept", displaying focused information about the item currently selected in "Concept" and referring to a Person:

q_R^2	($x, rdf:type, Person), ($x, hasName, $name), ($x, dateOfBirth, $date)
r_X^2	⟨div⟩ ⟨div class="info field"⟩Name⟨/div⟩ ⟨div class="info value"⟩$name⟨/div⟩ ⟨div class="info field"⟩Date of birth⟨/div⟩ ⟨div class="info value"⟩$date⟨/div⟩ ⟨/div⟩

Notice that variable $x appears in the queries of both XIPs. By selecting an item from "Concept", the value bound to $x in that item is passed to the dependent query before executing it. The dependent query only yields a result if the resource bound to $x has type Person (notice the first triple pattern in q_R^2). If the selected item has another type, e.g., Company, the dependent will not show up.

Further, assume that the user wants to show more specific information if the selected item is a Politician. He can customize the template by creating a second XIP depending on "Concept", for instance to include information about politicians. The new XIP's query could look like this:

q_R^3	($x, rdf:type, Politician), ($x, inParty, $y), ($y, name, $party)
r_X^3	⟨div⟩ ⟨div class="info field"⟩Affiliation⟨/div⟩ ⟨div class="info value"⟩$party⟨/div⟩ ⟨/div⟩

When the selected item in the parent XIP refers to a Politician, both dependent XIPs will be displayed.

4. SCENARIOS

Our demo is centered around political fact checking scenarios. Our background datasets will include DBpedia, YAGO, selected datasets from http://data.gov, Twitter feeds from prominent international politicians collected in the past months, and pre-selected, recent, political news articles from reputable online newspapers. We will show how to use FactMinder to explore the background knowledge, and engage the user in an interactive investigation process.

Users will be able to adapt scenarios to their own interest using documents from the Web. As they browse, users will be able to experiment with the annotation tools, and see how automatic and user-generated annotations from other users can be enhanced through the annotation editor. Upon starting the demo, a default set of XIPs will be displayed. Then, users will be guided through the process of modifying them and creating new ones to better reach the information they are looking for. Finally, we will demonstrate how all content created throughout the process is stored in the knowledge base, and can be reused in other scenarios.

5. CONCLUDING REMARKS

Making sense of Web resources, analyzing and consolidating their claims is an activity from which humans will most probably never be completely eliminated. However, the complexity of knowledge found in open data sources requires powerful, flexible ways for users to retrieve, analyze and understand the data, as well as produce and share complex annotations on the analyzed documents. Existing tools let users annotate Web pages directly from their browser, e.g., Dispute Finder [6] helps distributed users annotate pages to uncover disputed content or use it as supporting evidence for other claims. Closer to our work, DocumentCloud [7] provides an online service for newsrooms, where users create collections of documents, analyze and annotate them, share annotations, etc.

FactMinder is positioned as a software tool for facilitating fact checking and analysis. At its core is the interplay between visualization and knowledge production. FactMinder is powered by XR, a data model for documents with annotations [1], and is based on open standards, enabling it to integrate existing sophisticated tools for dedicated tasks. Unlike existing tools, an innovative characteristic of FactMinder is its support for XIPs, which can be seen as customizable "data lenses".

6. REFERENCES

[1] F. Goasdoué, K. Karanasos, Y. Katsis, J. Leblay, I. Manolescu, and S. Zampetakis. Growing Triples on Trees: an XML-RDF Hybrid Model for Annotated Documents. In *Very Large Data Search Workshop*, 2011.
[2] RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-mt/, 2004.
[3] OpenCalais. http://opencalais.com/.
[4] BaseX. http://basex.org.
[5] Virtuoso Open Source Edition 6.1.6. http://virtuoso.openlinksw.com.
[6] R. Ennals, B. Trushkowsky, and J. M. Agosta. Highlighting disputed claims on the web. In *WWW Conference*, 2010.
[7] The document cloud. http://documentcloud.org/.