Application and System Requirements

Deliverable D1.3
Version 0.6

Authors: Ian Hopkinson[1], Francis Irving[1], Aidan McGuire[1]
Affiliation: (1) ScraperWiki Ltd

BUILDING STRUCTURED EVENT INDEXES OF LARGE VOLUMES OF FINANCIAL AND ECONOMIC DATA FOR DECISION MAKING

ICT 316404

Grant Agreement No.	316404
Project Acronym	NEWSREADER
Project full title	Building structured event indexes of large volumes of financial and economic data for decision making.
Funding Scheme	FP7-ICT-2011-8
Project Website	http://www.newsreader-project.eu
Project Coordinator	Prof. dr. Piek T. J. M. Vossen VU University Amsterdam Tel. +31 (0) 20 5986466 Fax. +31 (0) 20 5986500 Email: piek.vossen@vu.nl
Document Number	Deliverable D1.3
Status & Version	DRAFT
Contractual Date of Delivery	July 2013
Actual Date of Delivery	
Type	Report
Security (distribution level)	Public
Number of Pages	12
WP Contributing to the deliverable	WP01
WP Responsible	SCW
EC Project Officer	Sophie Reig

Authors: Ian Hopkinson[1], Francis Irving[1], Aidan McGuire[1]

Keywords: API, user applications

Abstract: This document presents the application and system requirements for NewsReader. These requirements are derived from information on use case studies which are then presented as a set of example user applications. In addition we provide a brief description of the information provided by the Newsreader Annotation Format. We present examples of simple as well as advanced applications including a “Sober search” application, “dashboard” application and “Serendipity” application. The “Sober search” presents a typical advanced search interface. The “dashboard” provide alerts on pre-saved searches. The “Serendipity” interface provides a visualization of the NewsReader data allowing for serendipitous discovery of interesting material. The functionality requirements arising from these applications are listed, and an outline of how they might be supplied in the form of a Web API is provided. Finally, we identify the timeliness requirements of the NewsReader system in terms of a response time for API queries (of order 10 seconds for a user response application), and the timescale on which new news articles should be incorporated (of order 6 hours) to fit with

current news cycles.

Table of Revisions

Version	Date	Description and reason	By	Affected Section
0.1	4th June 2013	Initial document	Ian Hopkinson	
0.2	10th July 2013	Review	Marieke van Erp	
0.3	16th July 2013	Update	Ian Hopkinson	
0.4	18th July 2013	Review	Antske Fokkens	
0.5	19th July 2013	Update	Francis Irving	
0.6	22nd July	Update	Francis Irving	

Executive Summary

This document presents the application and system requirements for NewsReader. These requirements are derived from information on use case studies which are then presented as a set of example user applications. In addition we provide a brief description of the information provided by the Newsreader Annotation Format. We present examples of simple as well as advanced applications including a “Sober search” application, “dashboard” application and “Serendipity” application. The “Sober search” presents a typical advanced search interface. The “dashboard” provide alerts on pre-saved searches. The “Serendipity” interface provides a visualization of the NewsReader data allowing for serendipitous discovery of interesting material. The functionality requirements arising from these applications are listed, and an outline of how they might be supplied in the form of a Web API is provided. Finally, we identify the timeliness requirements of the NewsReader system in terms of a response time for API queries (of order 10 seconds for a user response application), and the timescale on which new news articles should be incorporated (of order 6 hours) to fit with current news cycles.

Contents

List of Tables

1 Introduction

In this document, we detail the system requirements for user applications and external APIs for the Newsreader project. To do this, we have established the types of user applications currently used in the space we believe NewsReader will occupy.

Existing applications involve users, typically in the finance industry, searching archives of news. These are described in full in "D1.2 User study on early demonstrators". As an indication of the value of this type of product, typically organizations charge of order $500 per calendar month for commercial news search and analysis subscriptions. This cost covers both the software application and the underlying data.

We have also considered what new applications may be made possible using the new technology developed in the NewsReader project. These include new ways of visualising news events.

In addition, we have specified an API intended for usage outside the project, which will be used at events such as hack days/hackathons. The features of this API are based on our own technical experience.

2 Demonstrators

The NewsReader project is working on a shortlist of demonstrators:

- **Car ownership** – using a large subset of articles focused on the automotive industry to study car manufacturer takeover activity. An example use is in mergers and acquisitions departments. Millions of source news articles to extract this information from have been supplied to the project by LexisNexis;
- **TechCrunch** – CrunchBase is a wiki-like database of technology companies, TechCrunch is a substantial blog reporting events relating to those companies. Together these form a great training dataset. The goal of this demonstrator would be to derive the contents of the database from the news article text of the blog. Then data can be derived from other technology news stories. This is useful for Venture Capitalists, researchers and Government planners;
- **Bankers Accuity/ABN Amro** – for banks, and other commercial entities, there is a requirement to understand in detail other small banks with whom they are thinking of doing business. They need to trust them. Bankers Accuity is a supplier of such data. This demonstrator aims to supply some of the data required in this process by scraping the contents of bank websites and company reports, and structuring that information using the NewsReader technology;
- **Dutch Parliament** – parliamentary enquires are provided with particular sets of information by the parliament's information department. This use case will examine those sources of information, to understand what politicians and party bureaus are basing their decisions on.

3 Available annotation data in NewsReader

We anticipate that the Newsreader system will have data available as described in the Grounded Annotation Format[1] (GAF) [Fokkens et al. 2013] as well as the NLP Annotation Format (NAF). Detailed descriptions of these formats can be found in "3.1 Annotation module" and "D4.1 Resources and linguistic processors". NAF is currently under development and will be the standard output format for linguistic processors in NewsReader. It includes information such as:

- Parts of speech labels;
- Sentiment features;
- Dependency relations;
- Chunks;
- Named entities;
- Opinions (distinct from sentiment which is general sentiment info);
- Events;
- Time;
- Factuality;

GAF provides RDF conform representations [Carroll and Klyne 2004, Guha and Brickley 2004] of information that is of interest to end-users. Activities such as chunking and parts of speech labeling are more relevant to the underlying linguistic processing rather than the anticipated use cases. We anticipate that the primary interest for end-users will be in the Events, Named entities, Opinions and Sentiment features. This information must be derived from NAF. The process of generating GAF representations from NAF representations is work in progress. The first module that carries out conversions from NAF to GAF will be described in Deliverable 2.1. We anticipate that GAF will provide the necessary information for use in weighting searches, clustering information and new visualisations of narrative. This includes:

- Summaries, plots, narratives
- Clustering of statements about the same event
- Importance of an event
- Evidence / opinion divergences
- Uncertainties

4 User Applications

We describe below a set of hypothetical user applications which we use to establish the functionality required of the NewsReader API. It is not intended that the NewsReader team will write these applications, it is simply a mechanism for thinking about the requirements of the API.

4.1 Sober Search

1" http://groundedannotationframework.org/

"Sober search" looks like a traditional advanced search engine. There is an input box for the search term or terms with options to restrict the search to specific time periods or search within specific document sets or to search within specific entity types, for example searching for Apple as a company entity rather than apple the fruit.

Such a search returns lists of:

- documents, ordered by event density;
- events, ordered somehow by importance;

In the case of a) it is the original newspaper articles or excerpts thereof with links to the full article, so the user would be able to get a Google News type format output. In the case of b) it is sets of RDF triples being returned, so to make it accessible they would be re-rendered as human language sentences (albeit with hyperlinks on the terms).

Required functionality:

- Prior to search: obtain available types and categories of data to build the advanced search interface:
  - Supply a list of named entity types;
  - Supply a list of document subsets;
- Search: allow search by keyword(s) with logical operators (OR, AND, NOT) and selection by named entity type and document subset.
- Returned article list including, for each article:
  - Source (URL to original)
  - Author
  - Date
  - Publisher
  - Excerpt
  - HTML excerpt with hyperlinks and highlighting of search terms.
  - Article ranking based on NewsReader technology i.e. event density, uncertainty
  - Article ranking based on events
  - Opinion annotation

4.2 Network explorer

The annotation that Newsreader will add is intrinsically network-like in character. Users should be able to examine this network-like character. We envisage this interaction starting from a seed search, for example a specific event or entity. This would bring up an overview display showing a selection of nodes which relate to the search made. Selecting one of these nodes will display linked nodes or provide more detail of the selected node. The user should have the ability to rearrange the display of nodes and also to hide nodes, and their children from view.

Detail for a selected node may include links back to the originating news articles, or perhaps images or DBpedia-like[2] articles on the node, where available.

Additional required functionality:

- Event network to some specified depth, i.e. a number of links from the seed event

2" http://dbpedia.org/

4.3 Timeline

Important to the human understanding of events is the timeline: placing events in order of time. The demonstrators for the NewsReader project are concerned with economic and financial matters, therefore it would be useful if structured data such as share prices and so forth could be displayed alongside the annotations of unstructured data also found in Newsreader.

Additional required functionality:

- None - functionality already covered by applications 4.1 and 4.2 above.

4.4 Serendipity

The Serendipity application would enable the user to discover information without requiring a specific stimulus as input, it would be configured, optionally, with some topics of interest and receive periodic updates from Newsreader as to new or "interesting" content. Material would be presented as some type of bubbling display with the user able to click onto a bubble to learn more if it seemed interesting. This would be more compelling with the addition of images, maps and even sound or video. This new content may be provided via an RSS feed. It is anticipated that the application would cycle over material received from the RSS feed rather than only responding to updates.

Additional required functionality:

- RSS feed of "Sober search" results.
- Ability to provide references to maps / images / sound / video

4.5 Table View

The Table View would display named entity and event information in a table format, i.e. circumventing any issues with generating natural language-like summaries.

Additional required functionality:

- None - functionality covered by application 4.1.

4.6 Watcher / Alerter

Strategic users of news applications will often be interested in dashboard or traffic light displays of events. That is to say that rather than seeing detail they wish to see alerts of events matching a certain search criteria. This is similar to LexisNexis' existing dashboard.

Examples of this might include companies looking at the amount of news coverage they are receiving relative to their competitors and the sentiment expressed in that coverage (i.e. positive or negative).

This implies a notification system and a sentiment system, or that the user application will make repeated requests.

Additional required functionality:

- Reporting of sentiment
- RSS feed
- Email notifications – or is this client functionality?

4.7 Search my documents

In addition to providing search and visualization of news articles from the outside world users may wish to carry out searches on private data from their organizations, or from sources not yet in NewsReader.

Required functionality:

- Ability to upload content

5 Required Functionality summary

This section summarizes the functionality required to deliver the user applications described in section 4.

Basic functionality, which all applications require:

- Prior to search: obtain available types and categories of data would need to be obtained to build the advanced search interface:
  - Supply a list of named entity types;
  - Supply a list of document subsets;
- Search: allow search by keyword(s) and selection by named entity type and document subset
  Returned data:
  Article list;
  Source (URL to original)
  Author
  Date
  Excerpt
  HTML excerpt with hyperlinks and highlighting of search terms.
  Ranking based on NewsReader technology
  Article ranking based on events
- Event network to some specified depth

Advanced functionality, required by particular applications:

- Search with logical operators (OR, AND, NOT)
- Returned data: Opinion annotation
- Convert natural language queries to search queries
- RSS feed of “interesting content”
- Reporting of sentiment
- Ability to upload content
- Ability to provide references to maps / images / sound / video

6 Timeliness

Although in principle Newsreader will offer a more contemplative approach to the news then we might expect from current news search applications we might anticipate that users will also expect the same functionality as their current systems.

Responses to simple searches should be returned in <10s since this is consistent with acceptable user responsiveness for desktop applications. Applications which require very complex queries can batch them, provided this is clearly conveyed to the user.

New news should be incorporated into the system on a timescale of approximately 6 hours, that is to say that the annotations which the NewsReader project will generate should be applied to new news items and made available to the API within 6 hours

7 Developer API

7.1 Query interface

We anticipate two APIs, a low level one which accesses the NewsReader architecture via the native SPARQL/RDF interface. However, for commercial or hobbyist developers, SPARQL/RDF has a steep learning curve therefore the second API will be a stateless API over HTTPS which returns JSON objects. GET requests should be used for queries – i.e. when the state on the server is not being altered. Query parameters go in the URL string after the ?, for example:

http://www.newsreader.eu/api/sobersearch?q=fred&category=people

POST requests should be used for API calls which modify the state of the server, such as uploading new news articles, or a “search on subset request” where the server would give a unique ID to the subset raised in a previous query. Occasionally, something that is naturally a GET request may be too long for a URL in which case a POST request should be used.

The JSON objects should not simply be a wrapper for the raw RDF but should present the underlying data in the simplest, most natural form. For example, a simple search query would make a response along the following lines:

```
{ “sparql_query”: “http://dfjnsdkjnfknfdgvknf”
    [
```

```
    {
    “id”: 1,
    “excerpt”: “Some excerpt from the first article…”,
    “excerpt_html”: “Some <blink>excerpt</blink> from the first article…”,
    “article_url”: http://my.article.com/01234”,
    “ranking_method1”: 1,
    “ranking_method2”: 2,
    “date”: 2011-11-03,
    “author”: Fred,
    “rdf”: ArticleURI
    },
    {
    “id”: 2,
    “excerpt”: “Some excerpt from the second article…”,
    “excerpt_html”: “Some <blink>excerpt</blink> from the second article…”,
    “article_url”: http://my.otherarticle.com/01234”,
    “ranking_method1”: 2,
    “ranking_method2”: 7,
    “date”: 2011-11-03,
    “author”: Fred,
    “rdf”: ArticleURI
    }
    …
    ]
}
```

If authentication is required we propose a simple system of getting an API key via another channel with this added to the request. This is a requirement particularly for hack days, as we've found more complex systems such as OAuth are a large barrier to developers using an API.

There are several libraries designed to smooth the process of generating a simpler API on top of a SPARQL/RDF endpoint these include: the Linked Data API [1] with implementations by Elda[2] from Epimorphics or Puelia [3] in PHP. More flexibly and particularly useful for working with Ruby on Rails are RDF.rb [4] and Swirrl's Tripod [5].

The API should allow for the paging of results. This means that for any queries that return a large number of entries, such as a search, they are returned in chunks of a specified size and chunk number.

In middle and later stages of the project, changes to the API should be backwards-compatible in most cases. Where a change is being made that is not compatible, it should instead be introduced as a new endpoint, and the old one deprecated before being later removed.

API backwards-compatibility is not so important in the very early stages of the project, where it is more important to move fast and develop a good API.

7.2 Upload interface

Alongside the ability to query Newsreader there should also be API facilities for uploading documents into Newsreader for processing and incorporation into future searches.

8 References

[1] https://code.google.com/p/linked-data-api/
[2] http://www.epimorphics.com/web/tools/linked-data-api.html
[3] https://code.google.com/p/puelia-php/
[4] https://github.com/ruby-rdf/rdf
[5] https://github.com/Swirrl/tripod

[Carroll and Klyne 2004] Jeremy J. Carroll and Graham Klyne. 2004. Resource description framework (RDF): Concepts and abstract syntax. W3C recommendation, W3C, February.
http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[Fokkens et al. 2013] Fokkens, Antske, Marieke van Erp, Piek Vossen, Sara Tonelli, Willem Robert van Hage, Luciano Serafini, Rachele Sprugnoli and Jesper Hoeksema (2013) GAF: A Grounded Annotation Framework for Events. *Proceedings of the first Workshop on EVENTS: Definition, Detection, Coreference and Representation.* Atlanta, USA.

[Guha and Brickley 2004] Ramanathan V. Guha and Dan Brickley. 2004. RDF vocabulary description language 1.0: RDF schema. W3C recommendation, W3C, February. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/