Application and System Requirements

Deliverable D1.4

Final

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 D1.4
Status & Version	FINAL
Contractual Date of Delivery	January 2015
Actual Date of Delivery	30 January 2015
Type	Report
Security (distribution level)	Public
Number of Pages	20
WP Contributing to the deliverable	WP01
WP Responsible	SCW
EC Project Officer	Susan Fraser

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
0.7	15 January 2015	Update to D1.4	Ian Hopkinson	3.1, 4.2, 4.3, 4.5, 7.3, 8
0.8	28 January 2015	Processed comments after internal review	Ian Hopkinson	1, 2, 3, 4, 7, 8
0.8	30 January 2015	Checked by coordinator	VUA	-

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

No table of figures entries found.

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 assume 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 around $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

Changes with respect to the draft "Application and System requirements" D1.3
• Removed Bankers Accuity example; • Added World Cup demonstrator used for first Hack Day;

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 of such data is in mergers and acquisitions departments. 6 Million 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;
- **World Cup** – for the first Hack Day held in London in 2014 we wanted a topical dataset with wide interest. For this we chose news articles relating to the 2014 World Cup, held in Brazil a short time after the Hack Day. We used a

combination of articles from LexisNexis, and articles scraped from the BBC (who also attended the Hack Day) and the Guardian;

- **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

Changes with respect to the draft "Application and System requirements" D1.3
• Wording changes to reflect that RDF is now being generated from the underlying NAF annotation, as described in D2.1. At draft this had only been anticipated.

The Newsreader system 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;
- Attribution/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 has been derived from NAF. The first module that carries out conversions from NAF to GAF has been described in Deliverable 2.1. We

[1] http://groundedannotationframework.org/

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

3.1 Annotation update

Changes with respect to the draft "Application and System requirements" D1.3
• This is a new section which highlights our changing viewpoint on user requirements;

In the period since the draft Requirements document was written the exact form that the annotation provided by NewsReader have evolved, and how users interact with news search systems has crystallised.

The key insight is that NewsReader is event centric whilst the user requirements as described in the draft of this document are much more document centric.

In practice the most interesting output from the NewsReader system is the output from semantic role labelling for events available in the RDF layer. This provides a "frame" for an event and roles within that event. Initially events and roles were labelled using FrameNet but latterly these frames are stored in the project's Event Situation Ontology. More recently situation graphs for events have been implemented which are the results of using reasoning algorithms on the event data. They, for example, will identify someone joining or leaving a company.

The Hack Days have highlighted the importance of timeline presentations of the newsreader data.

4 User Applications

Changes with respect to the draft "Application and System requirements" D1.3
• We have added some text and illustrations to the Network Explorer, Timeline, Serendipity and Table user applications section which show how we have implemented this functionality in the Decision Support Tool Suite and elsewhere;

Below, we describe 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

"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:

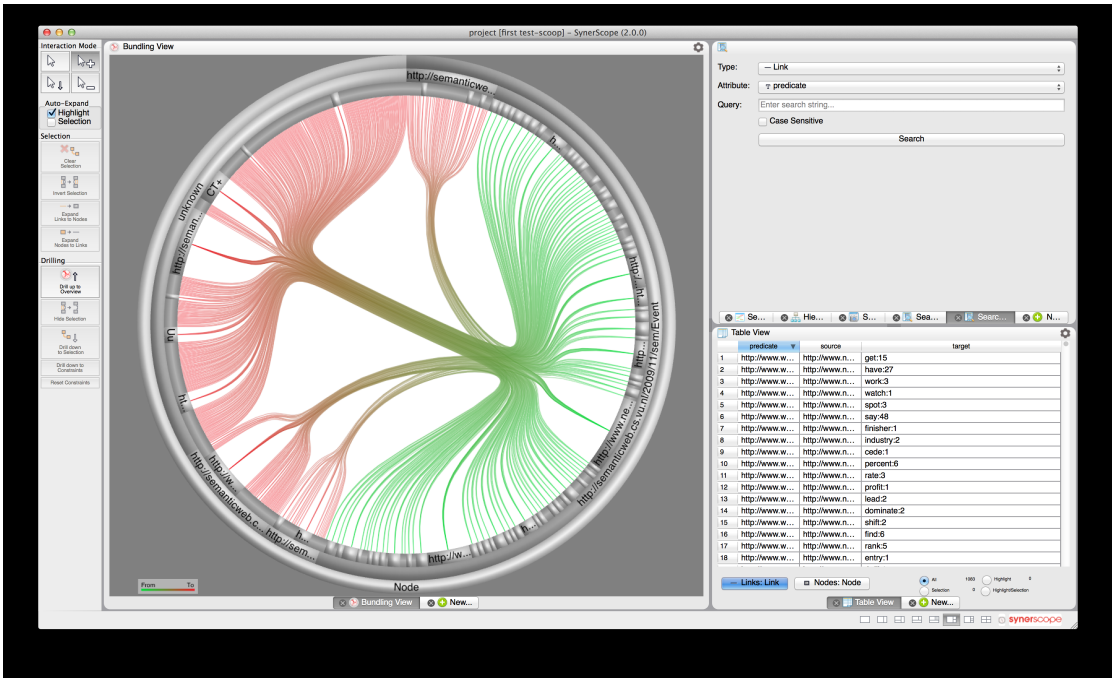
- Prior to search: obtain available types and categories of data to build the advanced search interface:
  - Supply a list of named entity types;
  - Supply a list of document subsets;
- Search: allow search by keyword(s) with logical operators (OR, AND, NOT) and selection by named entity type and document 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.

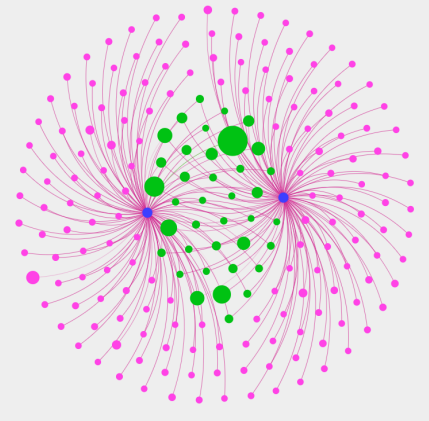
Detail for a selected node may include links back to the originating news articles, or perhaps images or DBpedia-like[2] articles on the node, where available.

The visualization below shows output from the Decision Support Tool Suite (DSTS) described in D7.3. This is a network like visualization of events around the car company BMW found in the "cars" data set. The DSTS allows users to explore this network view of events.



The image below shows a visualization of the network of interactions involving Sepp Blatter and David Beckham in the collection of articles concerning the World Cup which was used during the NewsReader Hack Day in London, 2014. The blue nodes with represent Blatter and Beckham. The green nodes represent individuals who occur in events involving Beckham and events involving Blatter, other nodes show individuals who have events with only one of them. The size of the nodes reflects the number of events the individual has been involved in. The "live" version of this visualization allows us to identify the individuals with most contacts with these "seed" figures. (http://stevenmaude.github.io/newsreader-network-vis/).

2 http://dbpedia.org/



The data for this visualization were obtained using a simple API described in section 7.1. The input parameters for this visualization are the URIs of the actors of interest. The visualization allows us to identify other key figures in the news surrounding the World Cup.

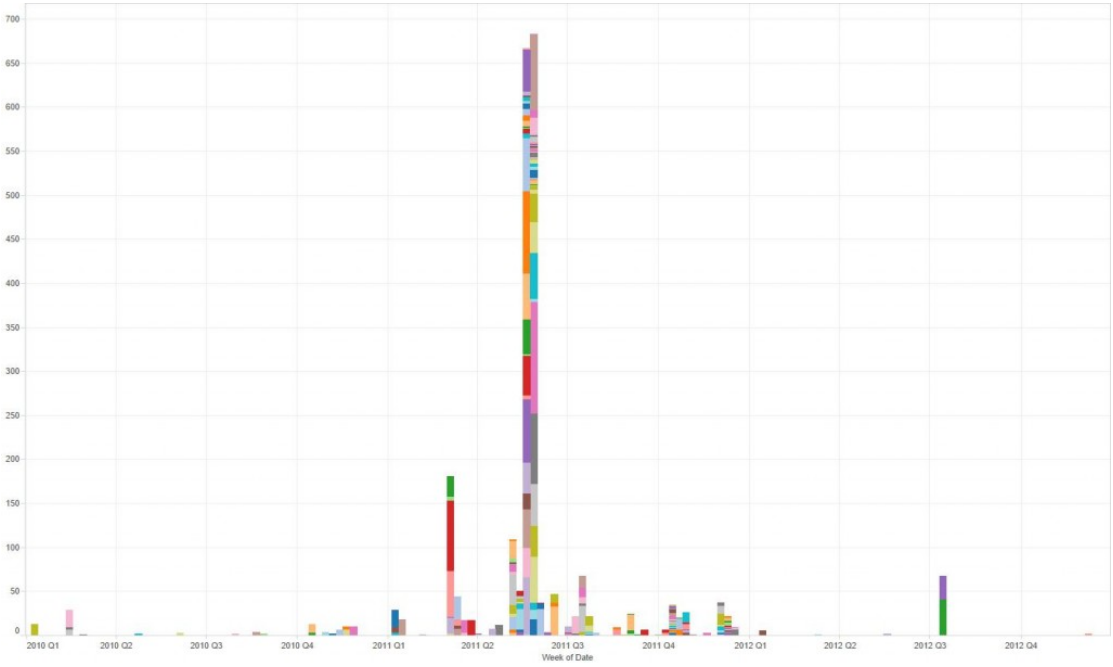
Additional required functionality:

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

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.

We have prototyped some of this functionality using the simple API described in section 7.3 shown in the image below. An interactive version of this visualization is available (http://public.tableausoftware.com/profile/ian.hopkinson#!/vizhome/blatter_and_beckham/NewsEvents).



This visualization shows a count of events mentioning Mohammed Bin Hammam as a function of time in the World Cup Hack Day dataset. The events are colour coded by the article in which an event occurred therefore larger bars correspond to news articles which contain more events. We can also code and filter the results by the event type.

In addition the Decision Support Tool Suite (DSTS) also has timeline functionality.

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.

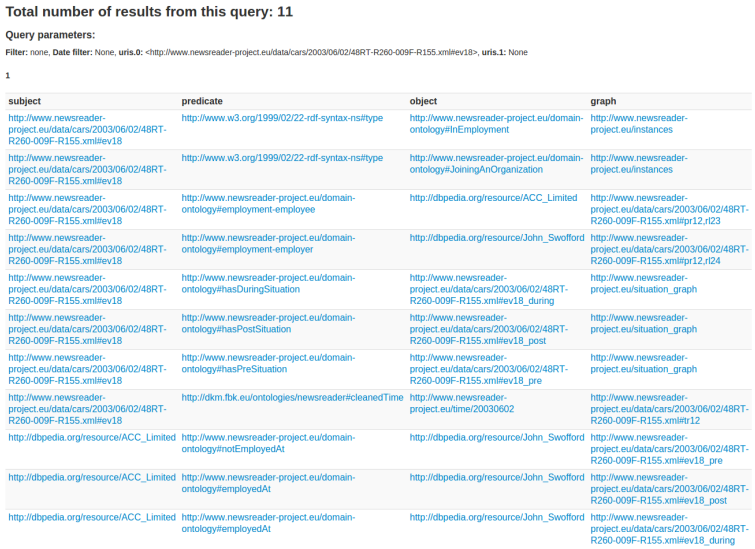
Additional required functionality:

- 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.

An example of this type of output is shown below. This is output from the Simple API for an “event precis” query which shows the most useful data for an event. The table format is the default method by which data are returned from this API for most query. Table-shaped data is easier to display in a generic fashion than the underlying graph-shaped data that the KnowledgeStore holds.



Total number of results from this query: 11

Query parameters:

Filter: none, **Date filter:** None, **uris.0:** <http://www.newsreader-project.eu/data/cars/2003/06/02/48RT-R260-009F-R155.xml#ev18>, **uris.1:** None

1

subject	predicate	object	graph
http://www.newsreader-project.eu/data/cars/2003/06/02/48RT-R260-009F-R155.xml#ev18	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.newsreader-project.eu/domain-ontology#InEmployment	http://www.newsreader-project.eu/instances
http://www.newsreader-project.eu/data/cars/2003/06/02/48RT-R260-009F-R155.xml#ev18	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.newsreader-project.eu/domain-ontology#JoiningAnOrganization	http://www.newsreader-project.eu/instances
http://www.newsreader-project.eu/data/cars/2003/06/02/48RT-R260-009F-R155.xml#ev18	http://www.newsreader-project.eu/domain-ontology#employment-employee	http://dbpedia.org/resource/ACC_Limited	http://www.newsreader-project.eu/data/cars/2003/06/02/48RT-R260-009F-R155.xml#pr12,rl23
http://www.newsreader-project.eu/data/cars/2003/06/02/48RT-R260-009F-R155.xml#ev18	http://www.newsreader-project.eu/domain-ontology#employment-employer	http://dbpedia.org/resource/John_Swofford	http://www.newsreader-project.eu/data/cars/2003/06/02/48RT-R260-009F-R155.xml#pr12,rl24
http://www.newsreader-project.eu/data/cars/2003/06/02/48RT-R260-009F-R155.xml#ev18	http://www.newsreader-project.eu/domain-ontology#hasDuringSituation	http://www.newsreader-project.eu/data/cars/2003/06/02/48RT-R260-009F-R155.xml#ev18_during	http://www.newsreader-project.eu/situation_graph
http://www.newsreader-project.eu/data/cars/2003/06/02/48RT-R260-009F-R155.xml#ev18	http://www.newsreader-project.eu/domain-ontology#hasPostSituation	http://www.newsreader-project.eu/data/cars/2003/06/02/48RT-R260-009F-R155.xml#ev18_post	http://www.newsreader-project.eu/situation_graph
http://www.newsreader-project.eu/data/cars/2003/06/02/48RT-R260-009F-R155.xml#ev18	http://www.newsreader-project.eu/domain-ontology#hasPreSituation	http://www.newsreader-project.eu/data/cars/2003/06/02/48RT-R260-009F-R155.xml#ev18_pre	http://www.newsreader-project.eu/situation_graph
http://www.newsreader-project.eu/data/cars/2003/06/02/48RT-R260-009F-R155.xml#ev18	http://dkm.fbk.eu/ontologies/newsreader#cleanedTime	http://www.newsreader-project.eu/time/20030602	http://www.newsreader-project.eu/data/cars/2003/06/02/48RT-R260-009F-R155.xml#tr12
http://dbpedia.org/resource/ACC_Limited	http://www.newsreader-project.eu/domain-ontology#notEmployedAt	http://dbpedia.org/resource/John_Swofford	http://www.newsreader-project.eu/data/cars/2003/06/02/48RT-R260-009F-R155.xml#ev18_pre
http://dbpedia.org/resource/ACC_Limited	http://www.newsreader-project.eu/domain-ontology#employedAt	http://dbpedia.org/resource/John_Swofford	http://www.newsreader-project.eu/data/cars/2003/06/02/48RT-R260-009F-R155.xml#ev18_post
http://dbpedia.org/resource/ACC_Limited	http://www.newsreader-project.eu/domain-ontology#employedAt	http://dbpedia.org/resource/John_Swofford	http://www.newsreader-project.eu/data/cars/2003/06/02/48RT-R260-009F-R155.xml#ev18_during

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

Changes with respect to the draft "Application and System requirements" D1.3
• Added a section 7.3 The Simple API which describes, briefly, an API implemented along the lines of that proposed in the earlier sections;

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.

7.3 The Simple API

Since the first draft of this document was prepared we have implemented a Simple API for developers along the lines of that proposed above. Details of this API have been described in a publication by Hopkinson, Maude and Rospocher[Hopkinson, Maude and Rospocher, 2014]. In addition the KnowledgeStore also has a lower level SPARQL/RDF query API which can be accessed directly or via the Simple API. In practice we discovered that the libraries mentioned above to wrap RDF/SPARQL were not mature. The Simple API was therefore constructed by making templated SPARQL queries direct to the SPARQL/RDF endpoint.

8 Update and summary

Changes with respect to the draft "Application and System requirements" D1.3
• This section summarises updates some general findings from the Hack Days we have run;

As part of the Hack Day process we implemented an API with the broad intention of delivering the user requirements outlined above. We were able to track the queries that the Hack Day participants made. The typical behaviour was to query to find events, most often using a specific FrameNet term (the "verb" of the event) and then to follow that up with a "describe" query which provides all of the available information regarding the event. The users were typically only interested in a small part of the information provided by the "describe" query. So we have implemented some simpler queries which provide particular details of an event.

This focus on the events has been influenced to some degree by the licensing conditions surrounding the material that Lexis Nexis is able to supply. We must respect the licenses under which they obtain material and therefore, in general, it has not been possible to supply direct links to original articles, nor to supply excerpts of those articles. In addition the

mindset of the NewsReader team is very "event" oriented since this is the focus of the core NewsReader technologies. We have mitigated the issues caused by licensing by putting in place redirects which lead users to the articles on the Lexis Nexis website, allowing access to those with appropriate credentials. Furthermore, Lexis Nexis has been able to negotiate more permissive licenses with some of its news providers.

In the final year of the project we hope to modify the API we have created to fit better with the user requirements described here.

9 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/

[Hopkinson, Maude and Rospocher, 2014] Hopkinson, Ian, Steven Maude and Marco Rospocher (2014), A simple API to the KnowledgeStore, *Proceedings of the ISWC Developers Workshop 2014*, vol 1268, p7-12.